UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

FRANCISCO GODIENSKI HANSEN

DESENVOLVIMENTO DE UMA SOLUÇÃO PARA AUTOMAÇÃO DE UM SISTEMA DE CULTIVO HIDROPÔNICO

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2020

FRANCISCO GODIENSKI HANSEN

DESENVOLVIMENTO DE UMA SOLUÇÃO PARA AUTOMAÇÃO DE UM SISTEMA DE CULTIVO HIDROPÔNICO

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, *Câmpus* Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientadora: Profa. Andreia Scariot Beulke

PATO BRANCO
2020

Ministério da Educação
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
UTFPR - CAMPUS PATO BRANCO
DIRETORIA GERAL - CÂMPUS PATO BRANCO
DIR. DE GRAD. E EDUCAÇÃO PROFISSIONAL-PB
DEP. ACADÊMICO DE INFORMÁTICA DAINF-PB

TERMO DE APROVAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

Desenvolvimento de uma Solução para Automação de um Sistema de Cultivo Hidropônico

POR

Francisco Godienski Hansen

Este trabalho de conclusão de curso foi apresentado no dia 23 de novembro de 2020, como requisito parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, pela Universidade Tecnológica Federal do Paraná. O acadêmico foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Banca examinadora:

Profa. MSc Andreia Scariot Beulke	**Profa. MSc Mainara Cristina Lorencena**	**Profa. MSc Adriana Ariati**
Professora orientadora	Professora convidada	Professora convidada

Prof. Dr. Edilson Pontarolo	**Profa. Dra. Mariza Miola Dosciatti**
Coordenador do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas	Responsável pela Atividade de Trabalho de Conclusão de Curso

Documento assinado eletronicamente por (Document electronically signed by) **ANDREIA SCARIOT BEULKE, PROFESSOR(A) ORIENTADOR(A)**, em (at) 03/12/2020, às 19:59, conforme horário oficial de Brasília (according to official Brasilia-Brazil time), com fundamento no (with legal based on) art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por (Document electronically signed by) **ADRIANA ARIATI, PROFESSOR MAGISTERIO SUPERIOR-SUBSTITUTO**, em (at) 04/12/2020, às 09:48, conforme horário oficial de Brasília (according to official Brasilia-Brazil time), com fundamento no (with legal based on) art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por (Document electronically signed by) **MARIZA MIOLA DOSCIATTI, PROFESSOR ENS BASICO TECN TECNOLOGICO**, em (at) 07/12/2020, às 07:44, conforme horário oficial de Brasília (according to official Brasilia-Brazil time), com fundamento no (with legal based on) art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por (Document electronically signed by) **MAINARA CRISTINA LORENCENA, PROFESSOR MAGISTERIO SUPERIOR-SUBSTITUTO**, em (at) 07/12/2020, às 08:13, conforme horário oficial de Brasília (according to official Brasilia-Brazil time), com fundamento no (with legal based on) art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por (Document electronically signed by) **EDILSON PONTAROLO, COORDENADOR(A) DE CURSO/PROGRAMA**, em (at) 07/12/2020, às 10:28, conforme horário oficial de Brasília (according to official Brasilia-Brazil time), com fundamento no (with legal based on) art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site (The authenticity of this document can be checked on the website) https://sei.utfpr.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador (informing the verification code) **1771262** e o código CRC (and the CRC code) **AAF74362**.

RESUMO

A utilização inadequada dos recursos naturais, o excesso de uso de fertilizantes, pesticidas e herbicidas e a necessidade do uso de grandes porções de terra pela agricultura tradicional impactam no meio ambiente de maneira cada vez mais perceptível. Isso, aliado ao crescimento da população e à demanda cada vez maior de alimento, é necessário que as técnicas de cultivo evoluam. Uma forma de cultivo que vem ganhando força e mercado é a agricultura orgânica, que visa reduzir e aproveitar da melhor maneira possível os recursos utilizados. Uma técnica de agricultura orgânica é a hidroponia, plantio feito em água, que proporciona melhores rendimentos para as culturas, melhor aproveitamento dos recursos e insumos e maior versatilidade do sistema. Essa versatilidade torna possível implantar um sistema de cultivo em um ambiente controlado, proporcionando a produção contínua de certos tipos de cultura. Neste trabalho, foi desenvolvido um sistema para automatizar e monitorar os sistemas de cultivo hidropônico. O sistema foi desenvolvido para dispositivos móveis utilizando a linguagem Flutter, com um servidor Java utilizando o *framework* Spring para gerenciamento das requisições, e o código para controlar *hardware* foi escrito utilizando a linguagem Arduino, que é baseada em C e serve para programar microcontroladores. As principais funcionalidades são a leitura dos dados dos sensores, o cadastro e monitoramento de culturas e plantios, e a automação do sistema hidropônico.

Palavras chave: Hidroponia. Dispositivos Móveis. Automação.

ABSTRACT

The inappropriate use of natural resources, excess usage of fertilizers, pesticides and herbicides, and the need to use large portions of terrain by the traditional agriculture impact the environment in increasingly noticeable ways. This, allied to population growth and the increased demand of food, makes the evolution of the traditional agriculture needed. A form of agriculture that is gaining strength and market is the organic agriculture, which aims to reduce and utilize the resources with the best efficiency possible. One technique of organic agriculture is the hydroponics, planting done in water, which provides better yield of the cultures, better use of resources and supplies and greater versatility of the systems. This versatility makes it possible to implant a cultivation system in a controlled environment, providing continuous production of certain types of culture. In this work, a system to automate and monitor hydroponic cultivation systems was developed. The system was developed as a mobile application, using the Flutter language, with a Java server using Spring framework to manage the requests and the code to control the hardware was written using the Arduino language, which is based in the C language and is used to program microcontrollers. The main functionalities are the reading of data from the sensors, the registration and monitoring of the cultures and crops and the automation of the hydroponic system.

Keywords: Hydroponics. Mobile Devices. Automation.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGEM DE CÓDIGO

LISTA DE FÓRMULAS

LISTA DE ABREVIATURAS E SIGLAS

DFT	*Deep Flow Technique*
ER	Entidade Relacionamento
HTTP	*Hyper Text Transfer Protocol*
IDE	*Integrated Development Environment*
IoT	I*nternet of Things*
MQTT	*Message Queue Telemetry Transport*
NFT	*Nutrient Film Technique*
pH	potencial Hidrogeniônico
PVC	*PolyVinyl Chloride*
SN	Solução Nutritiva
UML	*Unified Modeling Language*
UUID	*Universally Inique Identifier*
WWAP	*World Water Development Report*
WWDR	*World Water Development Report*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, os objetivos e a justificativa da realização deste trabalho. O texto é finalizado com a apresentação dos capítulos subsequentes.

1.1 CONSIDERAÇÕES INICIAIS

Os impactos causados pela agricultura convencional ao meio ambiente são evidentes e cada vez mais perceptíveis, entre os quais estão o uso ineficiente da água, a necessidade de grandes porções de terra para o plantio e a alta quantidade de fertilizantes, pesticidas e herbicidas utilizados, que acabam contaminando lençóis freáticos, rios e lagos. Por causa desses malefícios, outro tipo de cultivo tem ganhado força e mercado, como a agricultura orgânica (BARBOSA et al., 2015).

Além desses impactos, para suprir a demanda, a produção de alimentos deve crescer à medida que a população cresce. Porém, segundo Kampman, Brower e Schepers (2008, p. 8), apenas 18% das áreas destinadas para uso agropecuário são utilizados para agricultura e segundo a *World Water Development Report* (WWAP) (2009, p. 98) 70% da água doce captada é utilizada para a agricultura. Portanto, para produzir alimento de forma sustentável, as técnicas de cultivo necessitam evoluir.

A agricultura orgânica visa efetuar o cultivo com o menor impacto possível ao ambiente, reduzindo o uso de produtos químicos e fazendo melhor uso dos recursos utilizados no processo. Uma técnica de cultivo orgânico bastante difundida é a hidroponia, que traz vários benefícios em relação à agricultura convencional. Barbosa *et al.* (2015) definem os maiores benefícios desse tipo de cultivo: o melhor rendimento da cultura, a eficiência no aproveitamento de água e dos insumos e a versatilidade dos sistemas, indo desde sistemas rudimentares de fundo de quintal, normalmente implantados por hobbistas, até sistemas comerciais altamente sofisticados. Vale ressaltar também que se o sistema hidropônico for implantado em um ambiente controlado é possível ter produção contínua, ao longo do ano todo (BRECHNER; BOTH, 2018).

Esse contexto sustenta a necessidade e a importância de implantar sistemas de cultivo que fazem uso de menor quantidade de recursos, causem menor impacto ambiental e tenham retorno semelhante ou maior que a agricultura convencional. É esperado que esses sistemas possuam certo nível de automatização a fim de reduzir o contato humano, evitar falhas e aumentar o rendimento econômico do cultivo.

A tecnologia pode ser uma aliada promissora no desenvolvimento de sistemas para automatização, controle e gerenciamento de processos relacionados à produção agrícola, em suas diversas fases e atividades. Nesse sentido, o conceito de Internet das Coisas ou *Internet of Things* (IoT), que consiste em uma rede de comunicação entre diversos sistemas e equipamentos pode ser vista, também, como forma de contribuir para a melhoria de processos da agricultura, em específico de hidroponia, por ser o contexto deste trabalho. Essa rede consiste, normalmente, de uma infraestrutura de comunicação e processamento que interliga vários objetos e sensores, podendo comunicar-se com outras redes, com o mínimo de interação humana e elevado nível de autonomia (FILHO, 2016, p.11; PUNDIR; SHARMA; SINGH, 2016, p. 960).

Diante do exposto, este trabalho propõe o desenvolvimento de um sistema de automação e controle de um sistema de cultivo hidropônico que utiliza a técnica de *Nutrient Film Technique* (NFT) - técnica de filme de nutrientes, em tradução livre - que provê os melhores resultados para hortaliças folhosas, como a alface. A técnica de NFT consiste na circulação de uma Solução Nutritiva (SN) por um sistema de canais rasos em um sistema de laço fechado (TOGNONI; PARDOSSI, p. 147, 1998; ALBRIGHT; LANGHANS, 1996, p. 10).

1.2 OBJETIVOS

A seguir são apresentados o objetivo geral e os específicos deste trabalho.

1.2.1 Objetivo Geral

Desenvolver um sistema para automação e monitoramento de sistemas de cultivo hidropônico.

1.2.2 Objetivos Específicos

- Permitir ao usuário configurar os fatores operacionais do sistema hidropônico, entre eles a temperatura esperada do ambiente, a umidade relativa do ar desejada para o ambiente, o tempo de exposição da cultura à luz, a temperatura da SN e, para monitoramento, o potencial hidrogeniônico (pH) da SN.
- Gerenciar os fatores operacionais configurados pelo usuário, bem como automatizar processos, visando um melhor resultado final das culturas. Os processos a serem

automatizado são: Controle da circulação da SN pelo sistema, monitoramento do pH, controle da temperatura da SN, controle da temperatura e umidade do ar, e controle do tempo de exposição da cultura à luz;

- Permitir o monitoramento em tempo real dos dados extraídos dos sensores do sistema hidropônico.

1.3 JUSTIFICATIVA

A crescente expansão dos perímetros urbanos das grandes cidades e a crescente expansão da população mundial faz com que os alimentos sejam produzidos em locais cada vez mais afastados dos locais de consumo. Segundo Kampman, Brower e Schepers (2008, p. 8) previsão é que a população vivendo em centros urbanos ultrapasse os 7 mil milhões até 2050, o que torna necessária a tomada de medidas para garantir a segurança alimentar da população, assegurando a qualidade e a quantidade de alimentos necessários para suprir a demanda da população, com o mínimo de desperdício.

Estima-se que nas próximas décadas, seja necessário aumentar em até 70% da produtividade para suprir a demanda por alimento (FAO, 2015). O aumento da produtividade de métodos convencionais de agricultura pode causar uma grande gama de impactos negativos ao meio ambiente.

Entende-se por métodos convencionais de agricultura o plantio no solo, a céu aberto, com irrigação e uso de nutrientes e pesticidas. Entre os impactos causados pelos métodos tradicionais estão o mau uso da água, a necessidade por grandes porções de terra, a erosão do solo e o uso excessivo de nutrientes e agrotóxicos, que com as chuvas são transferidos para rios e lençóis freáticos (BARBOSA et al., 2015).

Uma alternativa aos métodos convencionais de cultivo é a hidroponia. O sistema de cultivo hidropônico tem se consolidado no mundo, trazendo maiores retornos com menos impacto ao ambiente (JENSEN, 1999), porém o sistema não automatizado requer que o produtor faça medições e ajustes periodicamente.

Este trabalho propõe o desenvolvimento de uma solução de automação para um sistema hidropônico, visando monitorar e gerenciar diversos fatores que compõem o sistema, e, ao detectar a necessidade de ajustes, intervir para que o sistema se mantenha com os parâmetros previamente configurados, garantindo, assim, a qualidade esperada da cultura de acordo com os parâmetros definidos. O sistema também será responsável por efetuar os cálculos necessários para definir quais ajustes serão realizados, facilitando a rotina do produtor e diminuindo a

probabilidade de erros, garantindo que o sistema tenha o melhor retorno possível para a cultura de interesse e o melhor aproveitamento dos recursos disponibilizados, reduzindo o impacto ao ambiente.

1.4 ESTRUTURA DO TRABALHO

O texto apresentado a seguir está organizado em capítulos. O Capítulo 2 apresenta o referencial teórico que fundamenta a proposta deste trabalho centrado em: hidroponia, sensores, atuadores e Internet das coisas. No Capítulo 3 estão os materiais e métodos de desenvolvimento da proposta deste trabalho. Os resultados obtidos estão apresentados no Capítulo 4. Por fim, estão as referências utilizadas para o desenvolvimento do trabalho.

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentados conceitos sobre o cultivo hidropônico, os sensores utilizados nas medições dos parâmetros de monitoramento do sistema hidropônico, tais como: sensor de temperatura e umidade do ar, sensor de temperatura da água e sensor de pH da água. Também são apresentados os dispositivos para acionamento e controle do fluxo de água do sistema e do controle de temperatura e de umidade do ambiente.

2.1 HIDROPONIA

Esta seção apresenta os conceitos relacionados ao cultivo hidropônico, bem como os principais métodos e os parâmetros que devem ser levados em conta nesse tipo de cultivo.

2.1.1 Conceitos Gerais

Hidroponia é um conceito que tem como significado o “trabalho na água”, do grego *hydro*=água e *ponos*=trabalho. O conceito define a técnica de cultivo protegido, em que o solo é substituído por uma solução aquosa contendo os elementos minerais essenciais ao desenvolvimento da planta, com fim de produzir culturas mais fortes e saudáveis (FURLANI et al, 2009).

2.1.2 Vantagens e Desvantagens

Segundo Delfin (2012) entre as vantagens de um cultivo sem solo, destacam-se:

- Uso de espaço de terra que não poderia ser utilizado pela agricultura convencional;
- Maior produtividade que o cultivo no solo;
- Menor uso de água;
- Reduzido uso de fertilizantes;
- Menos poluição.

Para Delfin (2012) as principais desvantagens são:

- Custo alto de implantação;

- Necessário conhecimento de práticas agrícolas, sem as quais a produtividade reduzirá;
- Necessário conhecimento e experiência no uso de nutrientes, pois a falta ou excesso deles afeta negativamente as plantas;
- Necessária dedicação e perseverança.

2.1.3 Culturas

Segundo Delfin (2012), devido ao princípio de absorção mineral ser o mesmo em cultivo com solo e no sistema sem solo, uma grande quantidade de plantas pode ser cultivada em sistemas sem solo, entre elas as leguminosas de baixo e médio porte, vegetais folhados, raízes, tubérculos, bulbos, plantas medicinais e aromáticas, ornamentais e flores.

A alface é um dos produtos mais cultivados em hidroponia, porém há outras espécies que são bastante utilizadas, entre elas: tomate, pepino, pimentão, ervas, berinjela e morango (DELFIN, 2012).

2.1.4 Técnicas

Os tipos de sistemas hidropônicos podem ser divididos em dois: sistemas hidropônicos em água e sistemas hidropônicos em substrato. No sistema hidropônico de substrato, a raiz da planta se desenvolve em um substrato inerte e a aplicação da SN é realizada, geralmente, por meio do sistema de irrigação por gotejamento e é mais utilizado para culturas que possuem o sistema radicular e parte aérea mais desenvolvida, pois o substrato garante sustentação à planta (DELFIN, 2012; FURLANI et al., 2009).

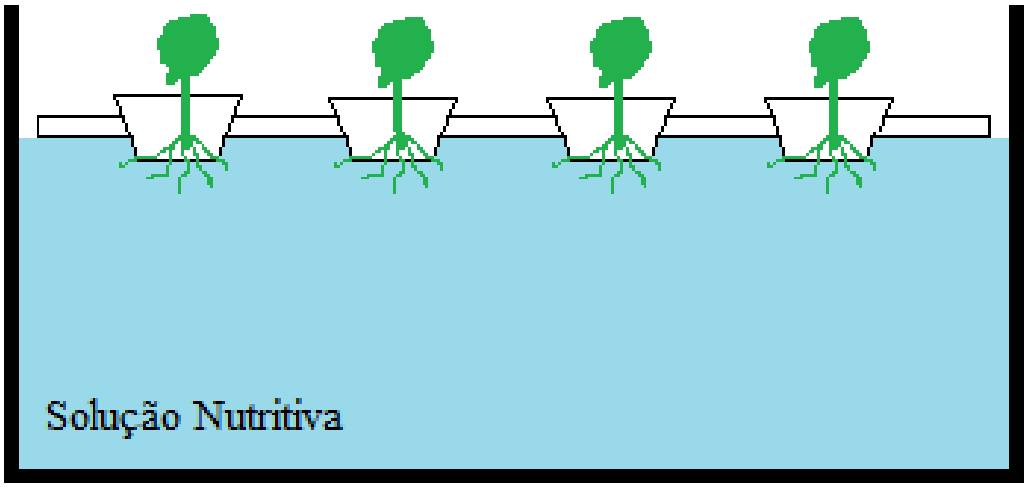
No sistema hidropônico em água, por sua vez, a planta fica com a raiz em contato direto com a SN, entre os quais se destacam o Sistema *Nutrient Film Technique* e o Sistema *Deep Filme Technique* (DFT), também chamado de *floating* ou Sistema de Raiz Flutuante (DELFIN, 2012; FURLANI et al., 2009).

2.1.5 Sistema DFT

A Solução Nutritiva é preparada em um reservatório com profundidade entre 5 e 20 cm e as raízes das plantas ficam parcialmente submersas. Na construção do sistema geralmente são utilizadas placas de poliestireno expandido, que fica flutuando sobre a SN. Na parte inferior da

placa há furos por onde saem as raízes das plantas para dentro da água. Como a Solução Nutritiva fica parada é necessário oxigená-la. Essa oxigenação pode ser feita manualmente utilizando um misturador plástico, cerca de duas vezes ao dia, ou então automaticamente, utilizando um compressor de ar. Ao utilizar essa técnica é necessário prestar atenção às raízes das plantas, pois se elas apresentarem coloração escura é um indicador de má oxigenação, o que limita a absorção de nutrientes e de água, afetando assim o crescimento das plantas (DELFIN, 2012; FURLANI et al., 2009).

Figura 1 – Diagrama de um Sistema DFT



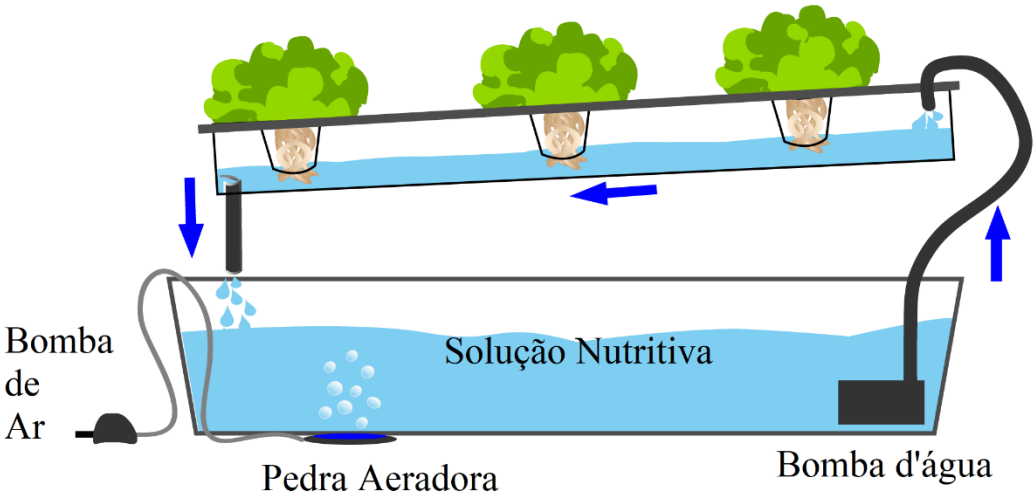
Fonte: Autoria própria.

2.1.6 Sistema NFT

No Sistema NFT original desenvolvido pelo Dr. Allen Cooper em 1965, a Solução Nutritiva circulava por canais de concreto forrados com um filme de polietileno com uma inclinação de 1%, porém ao longo dos anos o sistema sofreu alterações. Hoje em dia ele é mais frequentemente encontrado na forma de canais de cloreto de polivinila (*PolyVinyl Chloride* - PVC) (DELFIN, 2012; FURLANI et al., 2009).

O sistema consiste de um reservatório de Solução Nutritiva, ao qual está acoplado um sistema de bombeamento que distribui a SN aos canais de cultivo e escoa por gravidade ao sistema de retorno ao tanque, formando assim uma fina lâmina que irriga as raízes (FURLANI et al., 2009). Segundo Delfin (2012), esse é o sistema oficial de hidroponia no Brasil. A Figura 2 ilustra o funcionamento básico do sistema NFT.

Figura 2 - Diagrama de um Sistema NFT

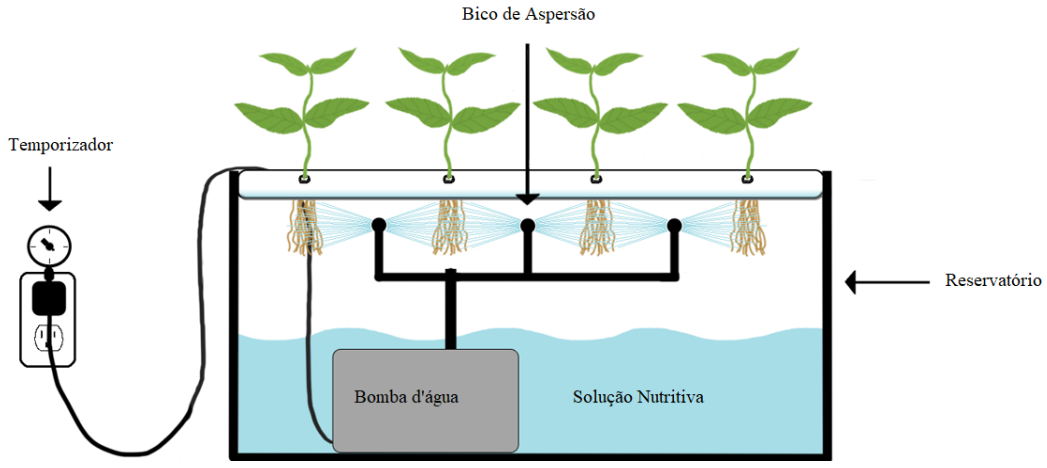


Fonte: NOSOILSOLUTIONS (2019).

2.1.7 Outros Sistemas

Há ainda outras técnicas de cultivo hidropônico que podem ser implementadas, porém essas possuem complexidade maior de implantação e manutenção. Entre elas destacam-se Técnica de Aeroponia (Figura 3). Nessa técnica as raízes ficam suspensas e são irrigadas por um sistema de aspersores que produz uma névoa da Solução Nutritiva (DELFIN, 2012).

Figura 3 - Diagrama de um Sistema de Aeroponia

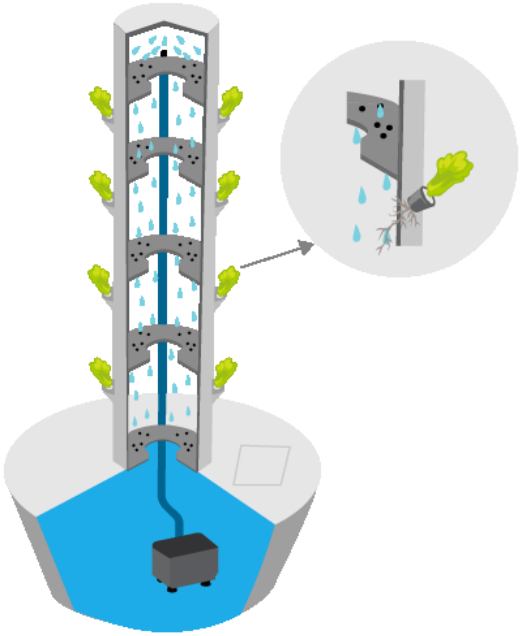


Fonte: Adaptado de Genesis (2019).

Segundo Delfin (2012), existe também o sistema de cultivo em colunas, no qual as culturas são plantadas em um sistema de colunas (Figura 4). O sistema permite uma alta produção por unidade de área, porém é limitado a plantas de pequeno porte e tem um custo de

implantação elevado. As plantas em um sistema de colunas necessitam de alta luminosidade para que sua taxa fotossintética não seja reduzida, resultando em baixa produtividade. Também é preferível implementá-lo em um sistema aberto, para prevenir doenças causadas por fungos.

Figura 4 - Sistema de cultivo em colunas



Fonte: Green e Vibrant (2019).

2.2 SENSORES

Sensores são dispositivos que alteram suas características devido a algum estímulo físico, representando esses estímulos com um sinal proporcional à medição. Um sensor pode realizar a medição de forma direta, transformando um estímulo físico (umidade, temperatura, pH, etc.) em um estímulo elétrico, permitindo a leitura do meio físico por meio de medidas que podem ser visualizadas em um meio digital (THOMAZINI; ALBUQUERQUE, 2005).

Há diversos tipos de sensores que podem ser utilizados para efetuar as medições necessárias em um sistema hidropônico. A seguir estão apresentados os tipos de sensores que são utilizados neste projeto.

2.2.1 Sensor de Temperatura

Os sensores de temperatura alteram seus parâmetros conforme a temperatura aumenta ou diminui. A medição é realizada utilizando as alterações nas características do silício, presente no sensor, de acordo com a mudança de temperatura (THOMAZINI; ALBUQUERQUE, 2005).

2.2.2 Sensor de Umidade

O sensor de umidade funciona com o princípio capacitivo de um filme fino, feito de uma lâmina de tântalo e outra de cromo, em que a capacidade deste filme muda de acordo com a umidade relativa do ambiente. Desta variação de capacidade, é feita a conversão eletronicamente, obtendo-se assim a umidade relativa (THOMAZINI; ALBUQUERQUE, 2005).

2.2.3 Sensor de Potencial Hidrogeniônico (pH)

A medição de pH é feita utilizando dois eletrodos, sendo um de vidro e outro de referência. Uma combinação de materiais de vidro constitui uma superfície que é sensível às variações de pH, porém, pelo fato do vidro possuir um potencial de assimetria, é impossível relacionar a medida do eletrodo diretamente com o pH de uma solução. Portanto é necessário calibrar este eletrodo, usando uma solução tampão, de pH previamente conhecido.

2.3 ATUADORES

Atuadores são dispositivos que recebem um sinal do sistema controlador e alteram o valor de uma variável já conhecida. Entre os atuadores podem ser citados os relés, os motores e as solenoides (THOMAZINI; ALBUQUERQUE, 2005). No sistema proposto são usados os seguintes atuadores:

a) Relés - são interruptores operados eletricamente, muitos deles utilizam eletromagnetismo, ativando ou desativando um eletroímã para ligar ou desligar um circuito. Nele, o circuito em operação é completamente isolado do circuito de ativação, permitindo que seja possível utilizar um circuito de baixa voltagem (geralmente entre 5v e 12v) para ativá-lo, e voltagens maiores no circuito de operação (MAKERLAB, 2019). Na hidroponia são utilizados para ligar e desligar as bombas d'água e o sistema de iluminação.

b) Bomba d'água - serve para levar água de um ponto a outro com facilidade. Na hidroponia é utilizada para circular a água dentro do sistema.

c) Sistema de Iluminação – as plantas necessitam de iluminação para fazerem a fotossíntese e se desenvolverem. O sistema de iluminação artificial é utilizado para garantir às plantas o período de iluminação necessário para o melhor desenvolvimento das mesmas.

2.4 INTERNET DAS COISAS

A Internet das Coisas, do inglês *Internet of Things* (IoT), é um termo que surgiu como um novo conceito de rede, que permite a comunicação entre os mais diversos tipos de aparelhos. É uma nova visão para a Internet, com a Internet passando a estar presente não somente em computadores, mas em objetos utilizados no cotidiano (FAZION FILHO, 2016).

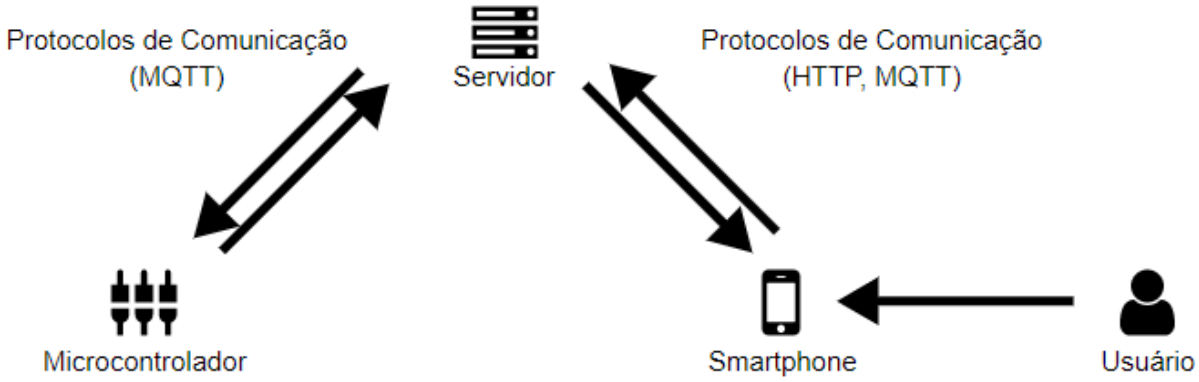
Segundo Huang (2010), a Internet é um sistema físico, composto de diversos computadores, interligados por um meio de comunicação, cuja função é a transmissão de informações. A Internet denota não somente a rede de computadores, mas também a forma que o compartilhamento dos dados é feito na rede física. Ainda segundo Huang (2010), a Internet das Coisas, por sua vez, depende diretamente da Internet e não constrói nenhuma outra estrutura física. Trata-se apenas de uma forma de compartilhamento de informações de um produto por meio da Internet.

Os avanços da tecnologia vêm expandindo as barreiras da Internet. O acesso à Internet por meio de banda-larga está ficando mais barato, os dispositivos estão ficando com capacidade de processamento cada vez maior e, ao mesmo tempo, reduzindo de tamanho. Isso muda não somente a forma como as pessoas acessam a Internet, mas permite uma nova gama de possibilidades (COETZEE, 2011). Segundo Singh (2014), a IoT está evoluindo a passos largos e está no processo de mudar a atual Internet estática para uma Internet totalmente integrada, que mudará a forma como as pessoas trabalham, pensam e vivem.

2.4.1 Arquitetura da IoT

Em um sistema IoT existem vários elementos que se comunicam entre si. Nesta seção são abordados os elementos que compõe um sistema IoT e que são utilizados no desenvolvimento deste trabalho. Esses elementos são: um servidor, um microcontrolador, as tecnologias e os protocolos de comunicação entre as partes e os usuários. Os usuários interagem com o sistema por meio de uma interface visual, implementada em uma aplicação para *smartphones*. Esses elementos são representados na Figura 5.

Figura 5 - Elementos utilizados no trabalho



Fonte: Autoria própria.

2.4.2 Protocolos de Comunicação

Para haver qualquer tipo de troca de dados na Internet, deve ser utilizado um protocolo de comunicação. Para páginas *web*, o mais utilizado é o *Hyper Text Transfer Protocol* (HTTP). Contudo, para o uso com a IoT ele tem algumas limitações. Há um *overhead* de processamento para cada requisição, pelos muitos cabeçalhos e regras. O HTTP é um protocolo síncrono e unidirecional, ou seja, o cliente deve iniciar a requisição e aguardar o retorno, gerando, assim, uma alta latência nas requisições. Também pelo fato de ser unidirecional, torna-se algo custoso fazer um *broadcast* para todos os dispositivos na rede.

Diferentemente dos microcomputadores, os dispositivos utilizados na IoT possuem limitações quanto a seu poder computacional, o que limita o uso do protocolo HTTP. Para essas aplicações, existe outro protocolo de troca de informações, que possui baixo custo operacional: o *Message Queue Telemetry Transport* (MQTT) (SANTOS et al., 2016).

Segundo SANTOS et al. (2016), o principal objetivo do MQTT é reduzir o *overhead* das requisições de transferência de dados, otimizando os recursos disponíveis nos dispositivos. As mensagens do MQTT são divididas em tópicos, em que cada tópico contém dados específicos de um assunto. A estrutura do MQTT é composta por três partes, o *publisher,* o *subscriber* e o *broker.* O *broker* funciona como uma espécie de servidor e gerencia o tráfego de dados. O *subscriber* registra-se ao *broker* para receber dados de um determinado tópico e o *broker* os avisa toda vez que um *publisher* publicar algo naquele tópico. Os dispositivos da IoT em geral têm função de *publisher*, publicando dados relativos às suas medições (SANTOS et al, 2016).

2.4.3 Servidor

O servidor é um computador que gerencia o armazenamento de dados e a comunicação entre os dispositivos que a ele se conectam. Funciona como o centro da aplicação, é acessado remotamente pelos usuários e dispositivos. Por receber todo o tráfego da aplicação, o servidor deve ser altamente escalável e suportar o recebimento de múltiplas requisições simultâneas.

2.4.4 Microcontrolador

Um microcontrolador é uma espécie de microcomputador de tamanho reduzido. Por ser utilizado para fins bem específicos, tem baixo poder computacional. São compostos de uma unidade de processamento, memória, unidades de comunicação, tanto via rede cabeada como sem fio. A esses microcontroladores, acoplados a algum objeto, juntamente com unidades de atuadores ou de sensores, dá-se o nome de *Smart Object*, ou Objeto Inteligente (SANTOS et al., 2016).

2.4.5 Usuários

Usuário é todo aquele que interage com o sistema por meio de um dispositivo eletrônico. A interação pode ser feita por um *smartphone* (celular) ou de um computador por meio de um sistema web ou desktop. A forma mais comum de interação se dá por meio de *smartphones*, forma da qual também é o contexto deste trabalho.

O usuário é quem opera um sistema IoT, é ele quem configura, analisa os dados e monitora o sistema. Os dispositivos IoT servem aos usuários, enviando e recebendo dados.

3 MATERIAIS E MÉTODO

Este capítulo apresenta as tecnologias, as ferramentas, os dispositivos e os componentes e demais materiais para o desenvolvimento deste trabalho. É, ainda, apresentado o método para esse desenvolvimento.

3.1 MATERIAIS

O Quadro 1 apresenta as tecnologias utilizadas no planejamento e implementação deste trabalho.

Quadro 1 - Materiais Utilizados

Ferramenta/Tecnologia	Versão	Referência	Finalidade
Visual Paradigm	15.2	https://visual-paradigm.com	Modelagem do Sistema
IntelliJ IDEA	2019.1.2	https://jetbrains.com/idea	Desenvolvimento do Servidor
Android Studio	3.4.1	https://developer.android.com/studio	Desenvolvimento do Aplicativo Android
PostgreSQL	9.6	https://postgresql.org	Banco de Dados do Servidor
SQLite	3	https://sqlite.org	Banco de Dados do Aplicativo Android
Arduino IDE	1.8.9	https://arduino.cc	Desenvolvimento do Sistema dos Microcontroladores
Spring Framework	5	https://spring.io	Framework de Desenvolvimento do Servidor
Java	8	https://www.oracle.com/java/	Linguagem de Desenvolvimento do Servidor
Flutter	1.22	https://flutter.dev/	Linguagem de Desenvolvimento para Smartphones
NodeMCU	-	https://www.nodemcu.com	Microcontrolador
Sensor PH4502C	-	-	Sensor de pH
Sensor DS18B20	-	-	Sensor de temperatura à prova d'água
Sensor DHT22	-	-	Sensor de temperatura e humidade
Bomba d'água	-	-	-
Fita de LEDs azuis e vermelhos	-	-	-

Fonte: Autoria própria.

3.1.1 Visual Paradigm

Sistema de desenvolvimento de diagramas para a modelagem de softwares. Possui tanto a modelagem *Unified Modeling Language* (UML) como a modelagem de Entidade Relacionamento (ER).

Há várias versões disponíveis: a Community, gratuita, porém limitada e que não pode ser utilizada comercialmente; e outras versões pagas, com mais funcionalidades e que podem ser utilizadas comercialmente (VISUAL PARADIGM, 2019).

3.1.2 Intellij IDEA

Desenvolvido pela empresa JetBrains é uma *Integrated Development Environment* (IDE) de desenvolvimento multiplataforma, com ênfase no Java. É uma IDE muito intuitiva, com recursos que simplificam bastante alguns processos de desenvolvimento.

Possui duas versões: a Community que é gratuita, porém com algumas limitações; e a Ultimate, versão paga, com todos os recursos necessários para desenvolver qualquer tipo de aplicação Java (JETBRAINS, 2019).

3.1.3 Android Studio

Desenvolvida pela JetBrains em parceria com a Google, é a IDE oficial de desenvolvimento de aplicativos para a plataforma Android. É disponibilizada gratuitamente.

Baseado no Intellij IDEA, o Android Studio adiciona mais funcionalidades que aumentam a produtividade no desenvolvimento de aplicativos Android (GOOGLE, 2019).

3.1.4 PostgreSQL

Sistema de Banco de Dados relacional *open source*, com mais de 30 anos de desenvolvimento ativo, tem uma forte reputação por sua confiabilidade, robustez e desempenho (POSTGRES, 2019).

3.1.5 SQLite

O SQLite é uma biblioteca baseada em C que implementa um mecanismo de banco de dados leve, robusto, de alta confiabilidade e portável. É o banco de dados padrão das aplicações Android (SQLITE, 2019).

3.1.6 Arduino IDE

IDE de desenvolvimento oficial para microcontroladores Arduino. Utiliza linguagem baseada em C e é compatível com todas as versões do Arduino. Pode ser estendido por meio de *plug-ins* para ser compatível com outros microcontroladores, como é o caso do NodeMCU (ARDUINO, 2019).

3.1.7 Spring Framework

Framework de desenvolvimento para Java que provê um modelo de programação e configuração compreensivo para aplicações Java modernas, para ser disponibilizado em qualquer plataforma de implantação (SPRING, 2020).

3.1.8 NodeMCU

Placa de desenvolvimento que combina o chip ESP8266 da Espressif Systems, com uma interface USB-Serial e um regulador de tensão 3.3v. Utiliza o padrão 802.11 b/g/n para conexão Wi-fi, possui 11 portas GPIO e 512k de memória flash (FILIPEFLOP, 2019).

3.1.9 Sensor PH4502C

Sensor para medir o pH de uma solução. Opera na tensão elétrica de 5v e entre as temperaturas de 0ºC a 60ºC. Tem a possibilidade de medição de pH do 0 a 14. Esse sensor deve ser calibrado de tempos em tempos para manter a precisão (DIYMORE, 2019).

3.1.10 Sensor DS18B20

Esse sensor é utilizado para medição de temperatura, por ser à prova d'água é utilizado principalmente em ambientes úmidos. Opera em temperaturas entre -55ºC e 125ºC e com voltagem mínima de 3.0v e máxima de 5.5v (DALLAS, 2019).

3.1.11 Sensor DHT22

Esse sensor é utilizado para medição de temperatura e umidade. Não é à prova d'água, então não pode ser molhado. Utilizado para medição da temperatura e umidade do ar. Trabalha nas faixas de 0% a 100% de umidade relativa do ar, com variação de 2%, e entre temperaturas de -40ºC a 80ºC, com variação de 0.5ºC (AOSONG, 2019).

3.2 MÉTODO

O desenvolvimento do sistema de automação da estufa hidropônica foi feito com base no modelo sequencial linear de Pressman (2006), que consiste no levantamento de requisitos, na modelagem destes requisitos e implementação. A seguir essas etapas são descritas:

a) Requisitos: a ideia de desenvolvimento do sistema surgiu a partir do interesse em hortas *indoor*, em um ambiente pequeno, com baixa iluminação e que tivesse baixo custo de implantação. A definição dos requisitos foi feita de acordo com os conhecimentos adquiridos por meio de pesquisa sobre a hidroponia e visando implementar o sistema de forma modular e, de fácil implantação e manutenção e com o menor custo possível.

b) Análise: a análise consistiu na modelagem dos requisitos levantados. O sistema consistirá de um aplicativo para gerenciamento, um servidor para a persistência de dados e de um sistema para leitura dos sensores e controle dos atuadores, a ser implantado nos microcontroladores.

c) Implementação: o desenvolvimento foi feito com base nos materiais e ferramentas apresentados na seção 3.1, visando o menor custo de implementação e a modularização do sistema. Foi dividido em etapas, sendo estas:

- Servidor de aplicação: servidor que gerenciará a persistência dos dados, tanto vindos do aplicativo, como vindos dos microcontroladores. Também foi efetuada a estruturação do banco de dados.
- Aplicação e registro de usuários: desenvolvimento inicial da aplicação, focado no desenvolvimento da interface de registro e *login* do usuário na aplicação.
- Cadastro de dispositivos, projetos e culturas: desenvolvimento das interfaces dos principais cadastros da aplicação.
- Controle dos microcontroladores e sensoriamento: implementadas as bibliotecas de controle, leitura dos sensores e controle dos atuadores.

- Integração: desenvolvimento da integração da aplicação com os dados recebidos pelo microcontrolador, desenvolvimento do gráfico de temperatura e umidade.

4 RESULTADOS

Este capítulo apresenta o que foi obtido como resultado do trabalho. Consiste na descrição do escopo, modelagem e implementação do aplicativo proposto.

4.1 ESCOPO DO SISTEMA

O sistema gerencia e automatiza os processos de um sistema hidropônico, desde o controle e o monitoramento da qualidade da Solução Nutritiva, monitorando o pH e controlando a temperatura da mesma, a temperatura e umidade do ar, bem como o tempo de exposição à luz da cultura, visando, assim, a possibilidade de simular *habitats* de diferentes regiões do planeta, tornando o ambiente propício para o cultivo de diversos tipos de plantas, ou então tornar propício o ambiente para o potencial máximo de produtividade de determinada cultura em qualquer época do ano.

A primeira parte da implementação foi o sistema que faz a integração dos microcontroladores e regula o sistema hidropônico com base na configuração selecionada pelo usuário. A segunda parte do desenvolvimento consiste em um aplicativo desenvolvido utilizando a linguagem Flutter para configurar o sistema hidropônico, permitindo a configuração dos seguintes parâmetros:

- Temperatura do Ambiente (ºC);
- Umidade desejada do ambiente (Umidade relativa do ar);
- Tempo de exposição à luz (horas/dia);
- Monitoramento do PH da SN;
- Temperatura da SN (ºC).

4.2 MODELAGEM DO SISTEMA

O Quadro 2 apresenta os requisitos funcionais levantados para o projeto proposto.

Quadro 2 - Requisitos funcionais do sistema

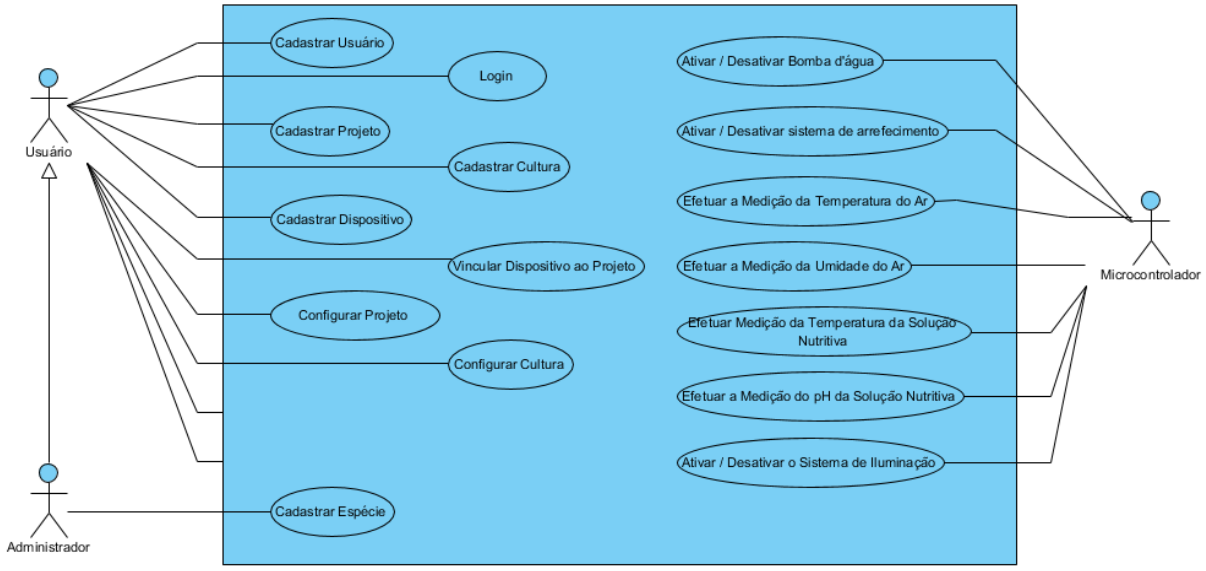
Id.	Nome	Responsável	Descrição
RF01	Cadastrar Usuário	Usuário	O usuário deverá registrar-se para utilizar o sistema. Os projetos ficarão vinculados a um usuário.
RF02	Login	Usuário	O usuário deverá autenticar-se no sistema para poder utilizá-lo.

RF03	Cadastrar Projeto	Usuário	O projeto contém dispositivos e culturas e é atrelado ao usuário.
RF04	Cadastrar Cultura	Usuário	A cultura é atrelada a um projeto. Composta de uma espécie e das configurações dos parâmetros a serem mantidos pelo sistema. Somente uma cultura ativa por projeto.
RF05	Cadastrar Dispositivo	Usuário	Cada dispositivo terá um identificador único (UUID), e estará atrelado diretamente ao usuário e pode ser vinculado a um projeto.
RF06	Cadastrar Espécie	Administrador / Usuário	Cadastro de espécies de plantas a serem utilizados nas culturas. Composto de nome comum e nome científico. Se for cadastrado por um usuário, deve ser aprovado pelo Administrador.
RF07	Vincular Dispositivo ao Projeto	Usuário	Vinculação do Dispositivo ao Projeto. O dispositivo só pode ser atrelado a um projeto por vez.
RF08	Configurar Projeto	Usuário	Composto de nome do projeto e dos dispositivos que fazem parte do mesmo.
RF09	Configurar Cultura	Usuário	Configurações de Temperaturas a serem mantidas, tempo de exposição à luz, pH esperado e umidade relativa do ar esperada.
RF10	Ativar/Desativar bomba d'água	Servidor / Microcontrolador	Servidor envia o comando ao microcontrolador para ativar ou desativar a bomba d'água.
RF11	Ativar/Desativar sistema de arrefecimento	Servidor / Microcontrolador	Servidor envia o comando ao microcontrolador para ativar ou desativar o sistema de arrefecimento.
RF12	Efetuar Medição da Temperatura do Ar	Microcontrolador	O microcontrolador lê o sensor de temperatura do ar e envia o valor ao servidor.
RF13	Efetuar Medição da Umidade do Ar	Microcontrolador	O microcontrolador lê o sensor de umidade do ar e envia o valor ao servidor.
RF14	Efetuar a Medição da Temperatura da Solução Nutritiva	Microcontrolador	O microcontrolador lê o sensor de temperatura da SN e envia os dados ao servidor.
RF15	Efetuar a Medição do pH da Solução Nutritiva	Microcontrolador	O microcontrolador lê o sensor de pH da SN e envia o valor ao servidor.
RF16	Ativar / Desativar o Sistema de Iluminação	Servidor / Microcontrolador	O servidor envia o comando de ativar ou desativar o sistema de iluminação ao microcontrolador.

Fonte: Autoria própria.

O diagrama de casos de uso contido na Figura 6 ilustra as funcionalidades principais do sistema, que são realizadas pelos atores Administrador, Usuário e Microcontrolador.

Figura 6 - Diagrama de Casos de Uso



Fonte: Autoria própria.

O Quadro 3 apresenta a descrição de caso de uso para as funcionalidades que constam na Figura 6 e que se referem aos cadastros.

Quadro 3 - Casos de Uso de Cadastros

Caso de Uso: Cadastrar projetos, dispositivos, cultura. **Descrição:** O Usuário efetuará o cadastro, incluindo os dados necessários a serem preenchidos em determinado cadastro. **Evento Iniciador:** Acessar o cadastro a ser inserido. **Atores:** Usuário **Pré-condição:** O Usuário deve ter o aplicativo instalado em seu dispositivo móvel, ter efetuado o registro, e o *login*. **Sequência de Eventos:** 1. O Usuário acessará a tela do cadastro desejado; 2. Pressiona o botão incluir; 3. O usuário preenche os dados necessários ao cadastro; 4. O usuário clica em salvar. **Pós Condição:**

Dados salvos no banco de dados.	
Fluxo Alternativo	**Descrição**
4. Dados não são válidos	4.1. Ao validar os dados o sistema constata que há dados inválidos. É mostrada uma mensagem de erro e permanece na tela do cadastro.

Fonte: Autoria própria.

O Quadro 4 representa os casos de uso presentes na Figura 6 que representam a configuração do projeto e da cultura.

Quadro 4 - Casos de Uso de Configuração

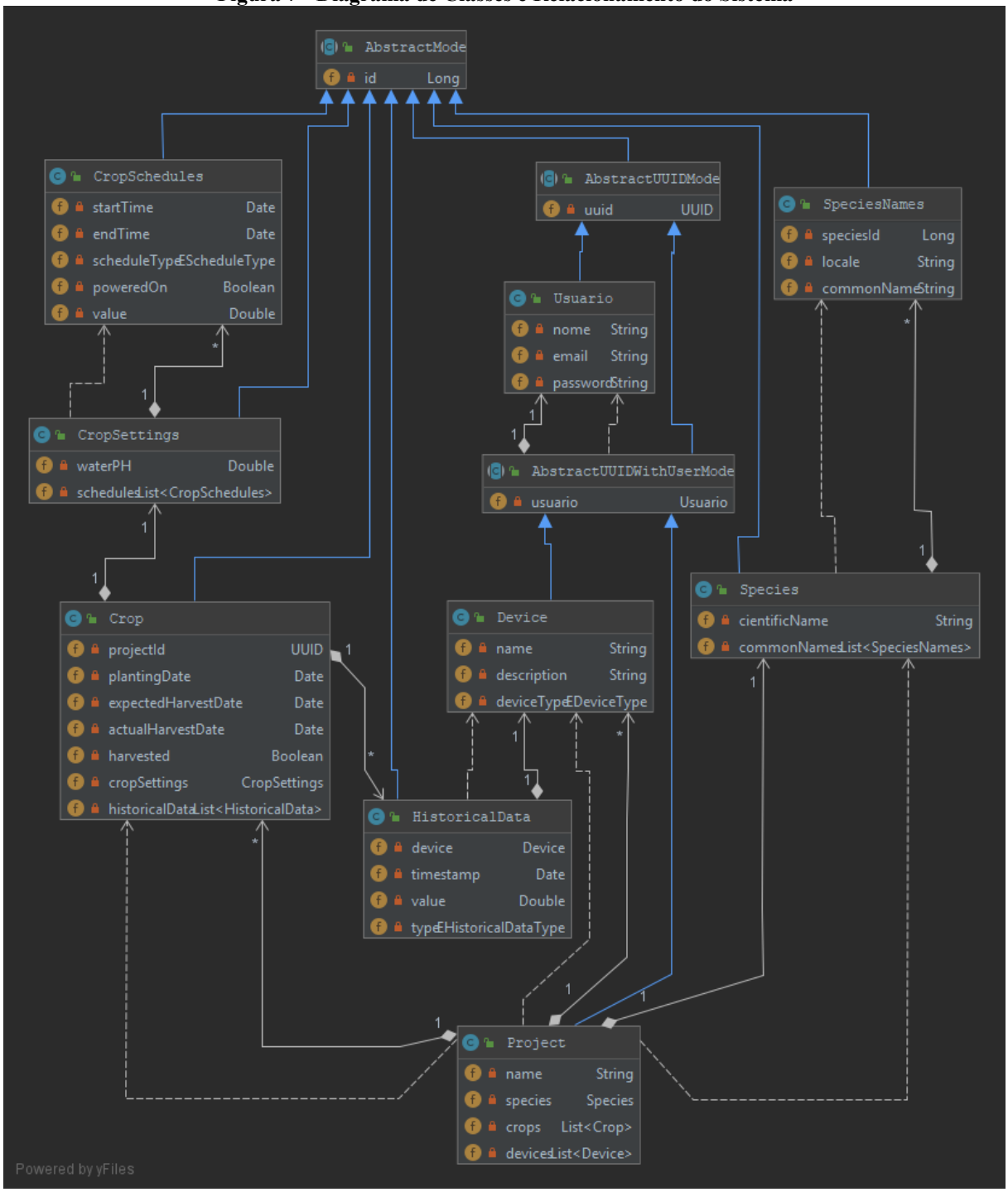
Caso de Uso: Configurar projetos, cultura. **Descrição:** O Usuário efetuará a configuração do projeto e da cultura de acordo com os parâmetros que espera-se que o sistema hidropônico mantenha. **Evento Iniciador:** Acessar o projeto/cultura a ser configurado. **Atores:** Usuário **Pré-condição:** O Usuário deve ter o aplicativo instalado em seu dispositivo móvel, ter efetuado o registro, o *login* e ter um projeto já cadastrado. **Sequência de Eventos:** 1. O Usuário acessará a tela da configuração; 2. Em seguida pressiona o botão editar 3. O usuário preenche os dados necessários à configuração 4. O usuário clica em salvar. 5. O Servidor envia as novas configurações ao microcontrolador **Pós Condição:** Dados salvos no banco de dados, microcontrolador reconfigurado.

Fluxo Alternativo	**Descrição**
4. Configuração Inválida	4.1. Ao validar, se for verificada que uma das configurações é inválida, o sistema apresentará uma mensagem de erro e permanecerá na tela de configuração.

Fonte: Autoria própria.

A Figura 7 apresenta o diagrama das entidades e relacionamentos que representam a estrutura do banco de dados do sistema.

Figura 7 - Diagrama de Classes e Relacionamento do Sistema



Fonte: Autoria própria.

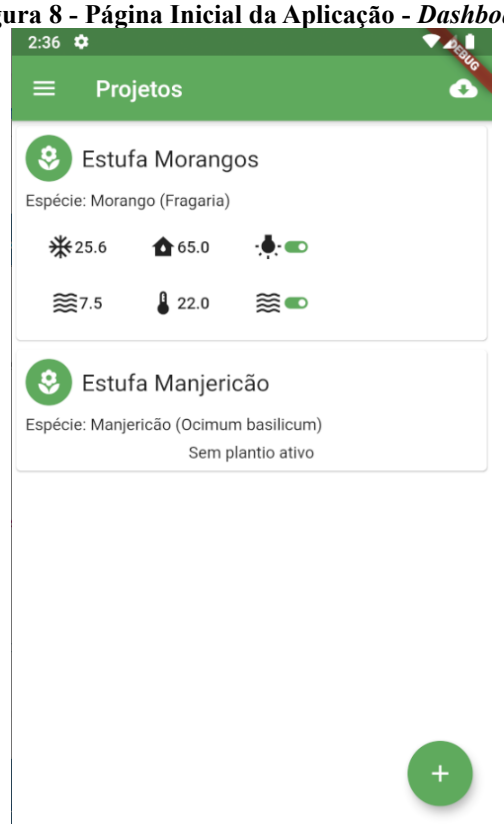
4.3 APRESENTAÇÃO DO SISTEMA

Esta seção apresenta as funcionalidades do sistema por meio da apresentação de suas telas. As telas apresentadas nesta seção são *mock-ups*, ou protótipos de como o aplicativo foi implementado e demonstram suas principais funcionalidades.

A Figura 8 apresenta a tela principal da aplicação, que foi desenvolvida em formato de *dashboard*. O *dashboard* contém uma lista com os projetos cadastrados pelo usuário, no formato de um *card*, com a espécie da cultura atual e o valor das últimas medições do projeto. Ao arrastar um item da lista para a esquerda, é mostrado um botão que permite a exclusão do projeto.

No canto inferior direito há um botão que quando clicado redireciona para a tela de cadastro de um novo projeto. No canto superior esquerdo há um botão que ao ser clicado abre o menu lateral que permite navegar entre os cadastros. Esses dois botões são padrões para todas as telas de listagem.

Figura 8 - Página Inicial da Aplicação - *Dashboard*

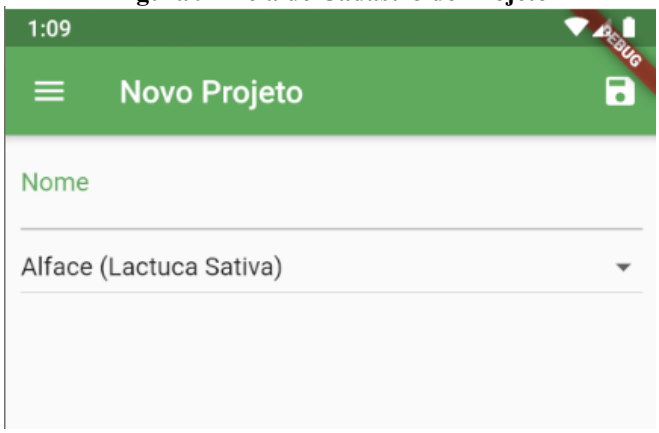


Fonte: Autoria própria.

A Figura 9 apresenta a tela de cadastro de um novo projeto, na qual é possível informar o nome do projeto e selecionar a espécie que será utilizada no mesmo.

Por padrão, as telas de cadastro possuem em seu canto superior direito um botão utilizado para salvar o registro, e em seu canto superior esquerdo, um botão que é utilizado para descartar as alterações e retornar à tela anterior.

Figura 9 - Tela de Cadastro do Projeto



Fonte: Autoria própria.

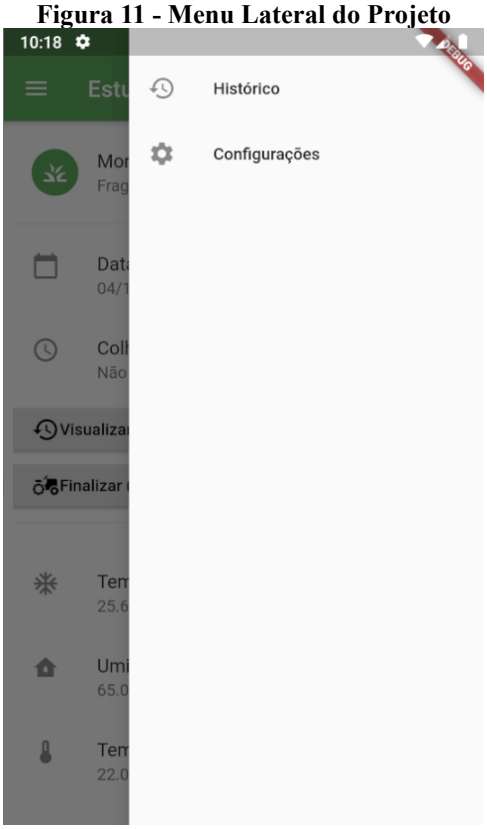
A Figura 10 apresenta a tela do projeto, para projetos sem um plantio ativo, na qual é possível visualizar a espécie definida para o projeto e há dois botões: um para cadastrar um novo plantio e outro para acessar o histórico de plantios efetuados com este projeto.

Figura 10 - Tela do Projeto - Sem plantio ativo

Fonte: Autoria própria.

Na parte superior da tela de projetos, do lado direito, há um botão que é utilizado para abrir o menu do projeto, no qual existe um botão que redireciona para a tela de configurações do projeto. Quando há um plantio ativo no projeto, neste menu também é mostrado um botão utilizado para acessar o histórico de medições do projeto. Este menu está exibido na Figura 11.

Figura 11 - Menu Lateral do Projeto

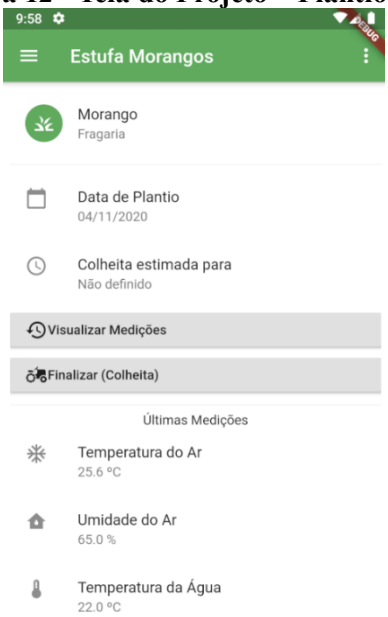


Fonte: Autoria própria.

As Figura 12 e 13 apresentam a tela do projeto que possui um plantio ativo, que pode ser acessada clicando no projeto apresentado no *dashboard.* Nessa página, é possível visualizar a espécie do projeto, a data de plantio, e a data estimada para a colheita. Abaixo disso, há um botão que quando clicado redireciona para a tela de histórico de medições do projeto. Abaixo deste botão há o botão “Finalizar (Colheita)” que redireciona o usuário para a tela de colheita do plantio, para finalizá-lo.

Abaixo dos botões há uma lista com as últimas medições de cada parâmetro efetuadas no sistema.

Figura 12 - Tela do Projeto – Plantio ativo



Fonte: Autoria própria.

Figura 13 - Tela do Projeto – Plantio ativo

Fonte: Autoria própria.

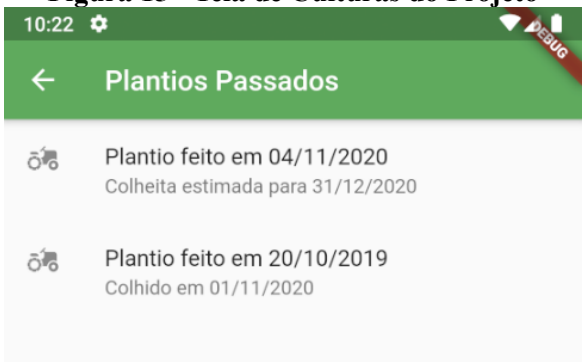
A Figura 14 mostra a tela de dispositivos conectados a um projeto. Essa página contém a lista dos dispositivos, com uma descrição e o tipo do dispositivo e mostra também o estado atual de conexão do dispositivo, mostrando um ícone vermelho caso haja falha de comunicação com o dispositivo. É possível acessar esta tela ao clicar no botão “Dispositivos Conectados”, na tela de projetos. Nessa tela, também é possível vincular outros dispositivos ao projeto.

Figura 14 - Lista de Dispositivos Conectados

Fonte: Autoria própria.

A Figura 15 apresenta a tela de plantios do projeto, na qual é possível visualizar a lista dos plantios do projeto, tanto os já colhidos como o que estão ativos.

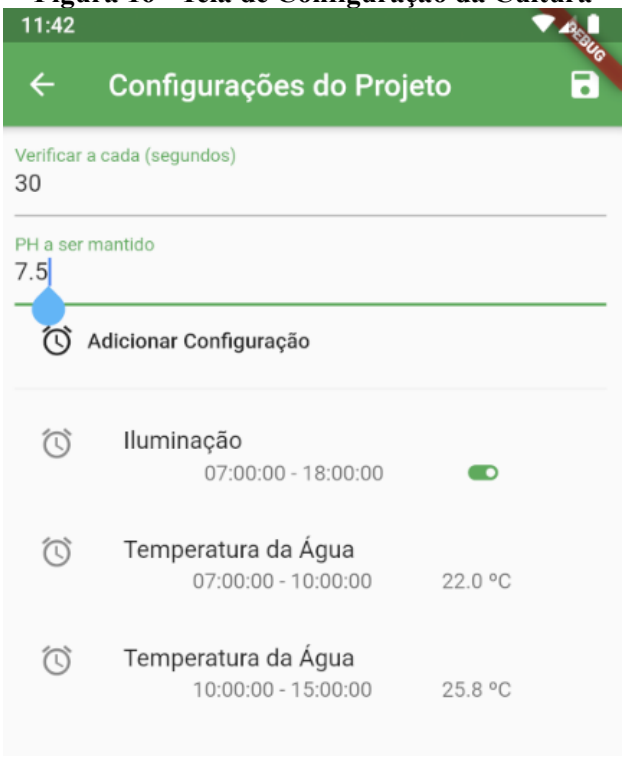
Figura 15 - Tela de Culturas do Projeto



Fonte: Autoria própria.

A Figura 16 apresenta a tela de configuração da cultura, sendo possível definir o tempo entre consulta dos sensores e o pH esperado durante o período de cultivo. Também é possível efetuar agendamentos, definindo a temperatura, umidade e iluminação esperados para determinado período do dia, bem como o período em que a configuração estará ativa.

Figura 16 - Tela de Configuração da Cultura

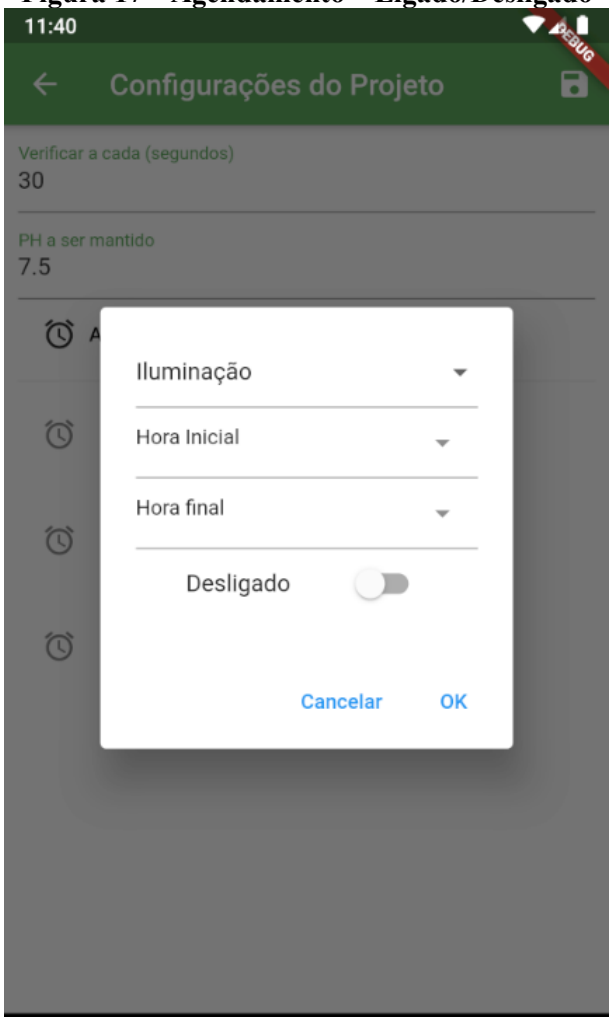


Fonte: Autoria própria.

As figuras seguintes representam a inserção de agendamentos na configuração da cultura. No primeiro campo da tela é possível selecionar o tipo do agendamento, o que alterará o tipo do campo de valor de acordo com o tipo de agendamento. Os campos de valores possíveis são campo de estado e de valor numérico.

A Figura 17 representa um agendamento para qual o dispositivo configurado possua somente dois estados que pode ser ligado e desligado.

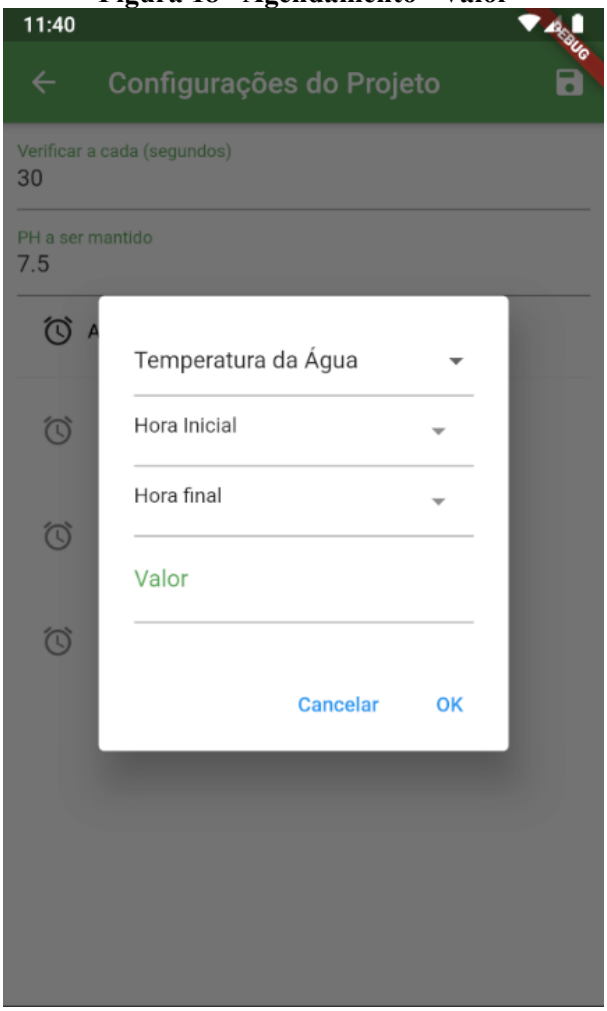
Figura 17 – Agendamento – Ligado/Desligado



Fonte: Autoria própria.

A Figura 18 representa a configuração de um agendamento cujos parâmetros a serem monitorados são representados por um valor numérico, por exemplo, temperatura e umidade.

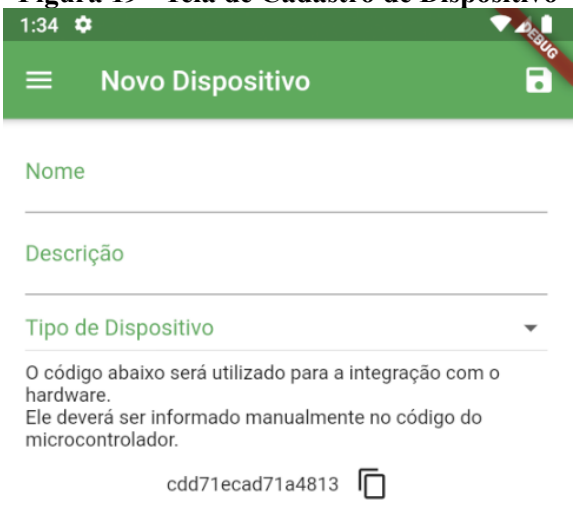
Figura 18 - Agendamento - Valor



Fonte: Autoria própria.

A Figura 19 representa a tela de cadastro de um novo dispositivo, na qual é possível definir uma descrição para o dispositivo e o tipo que será cadastrado. Cada dispositivo possuirá um identificador único, que será utilizado na comunicação via MQTT. Esse identificador é baseado em um *universally unique identifier* (UUID), que é gerado utilizando o *timestamp* e o endereço MAC do computador que a gera (UUIDGENERATOR, 2019).

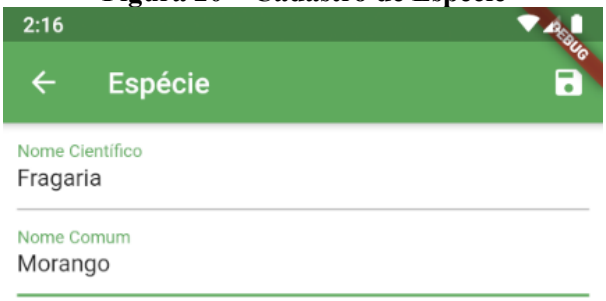
Figura 19 - Tela de Cadastro de Dispositivo



Fonte: Autoria própria.

A Figura 20 apresenta a tela de cadastro de espécies, na qual é possível definir o nome científico e comum da espécie.

Figura 20 – Cadastro de Espécie



Fonte: Autoria própria.

4.4 IMPLEMENTAÇÃO DO SISTEMA

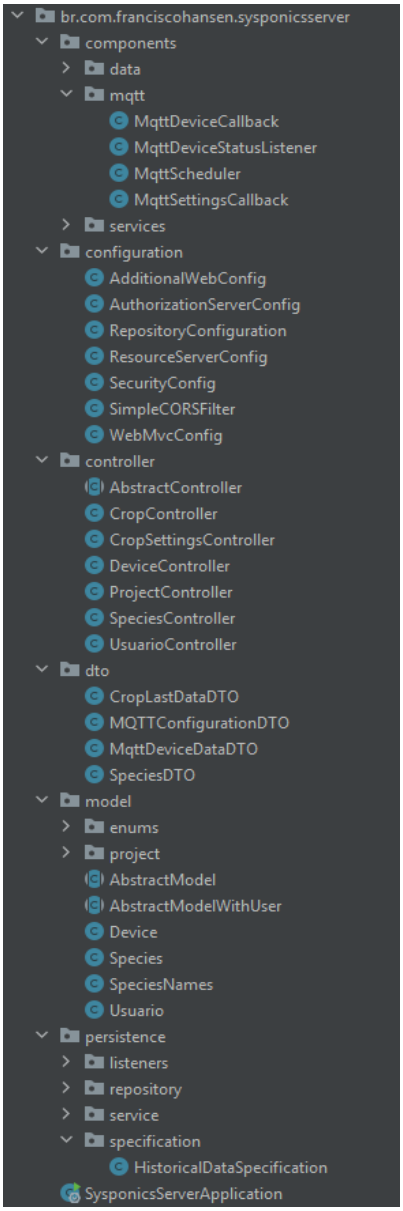
Nesta seção são apresentadas partes da codificação do sistema em listagens de código com o objetivo de mostrar como foi organizada a implementação do sistema.

4.4.1 Codificação do Servidor

A Figura 21 apresenta a estrutura de pacotes utilizada no servidor, onde o pacote base é o *br.com.franciscohansen.sysponics*, e seus subpacotes estão divididos de acordo com sua

função. O pacote *model* guarda os objetos das entidades a serem persistidas no banco de dados. O pacote *persistence* guarda as classes da camada de persistência da aplicação, ou seja, as classes que fazem leitura e gravação na base de dados. O pacote *controller* guarda as classes dos *controllers* da aplicação, ou seja, as classes que expõem métodos acessíveis via requisições HTTP. O pacote *configuration* guarda as classes utilizadas na configuração da aplicação e o pacote *components* guarda os *beans* de componentes que implementam funcionalidades ao sistema, e que podem ser instanciados utilizando a anotação *@Autowired* em outros *beans*. O pacote *dto* contém as classes de objetos auxiliares utilizadas pelo aplicativo e pelo microcontrolador.

Figura 21 - Estrutura de Pacotes do Servidor



Fonte: Autoria própria.

4.4.2 Configuração do Servidor

Esta seção trata da codificação utilizada para efetuar a configuração do Servidor da Aplicação.

A Listagem 1 apresenta a classe *SecurityConfig*, que é utilizada para efetuar as configurações de segurança do Servidor. O método *configure (HttpSecurity http)* define que os *endpoints /oauth/authorize e /oauth/confirm_access* serão utilizados para efetuar a autenticação do usuário, e que qualquer requisição efetuada em outro *endpoint* deverá ser autenticada. O método *configure (AuthenticationManagerBuilder auth)* define que o serviço utilizado para efetuar a validação dos dados do usuário será o *userDetailsService*, e que as senhas utilizadas pelo sistema serão codificadas utilizando o *BCryptPasswordEncoder*.

Listagem 1 - Classe SecurityConfig

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

  @Autowired
  @Qualifier( "usuarioServiceImpl" )
  private UserDetailsService userDetailsService;

  @Override
  protected void configure( HttpSecurity http ) throws Exception {
    http
            .requestMatchers()
            .antMatchers( "/oauth/authorize", "/oauth/confirm_access" )
            .and()
            .authorizeRequests()
            .anyRequest().authenticated()
            .and()
            .csrf().disable();
  }

  @Override
  protected void configure( AuthenticationManagerBuilder auth ) throws
Exception {
    auth.userDetailsService( userDetailsService )
            .passwordEncoder( new BCryptPasswordEncoder( 10 ) );
  }

  @Override
  @Bean
  public AuthenticationManager authenticationManagerBean() throws Exception {
    return super.authenticationManagerBean();
  }

  @Bean
  public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
  }
}
```

Fonte: Autoria própria.

A classe *AuthorizationServerConfig*, exibida na Listagem 2 define que o servidor também será utilizado como um servidor de autorização, e define os demais parâmetros utilizados para autenticação do usuário, em seus métodos de configuração. O servidor de autorização não precisa necessariamente ficar junto ao servidor de aplicação. Pode se utilizar um serviço de terceiros para efetuar a autorização dos usuários também.

Listagem 2 - Classe AuthorizationServerConfig

```
@Configuration
@EnableAuthorizationServer
public class AuthorizationServerConfig extends
AuthorizationServerConfigurerAdapter {

    @Autowired
    @Qualifier("authenticationManagerBean")
    private AuthenticationManager authenticationManager;

    @Autowired
    @Qualifier("usuarioServiceImpl")
    private UserDetailsService userDetailsService;

    @Override
    public void configure(ClientDetailsServiceConfigurer clients) throws
Exception {
        clients.inMemory()
                .withClient("sysponics-server")
                .secret(new BCryptPasswordEncoder().encode("sysponics2019"))
//senha da aplicação
                .autoApprove(true)
                .authorizedGrantTypes("authorization_code", "refresh_token",
"password", "client_credentials")
                .scopes("openid");
    }

    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints)
throws Exception {
        endpoints
                .authenticationManager(authenticationManager)
                .userDetailsService(userDetailsService);
    }

    @Override
    public void configure(AuthorizationServerSecurityConfigurer oauthServer)
throws Exception {
        oauthServer
                .tokenKeyAccess("permitAll()")
                .checkTokenAccess("permitAll()")
                .passwordEncoder(new BCryptPasswordEncoder(10));
    }

}
```

Fonte: Autoria própria.

4.4.3 O Pacote Model

As classes do pacote *model* devem ser anotadas com a anotação @*Entity*, para que seja possível persisti-las no banco de dados. As colunas das tabelas são definidas nos campos das classes, utilizando a anotação @*Column*. Os campos do tipo "data" devem ser anotados com a anotação @*Temporal*, que define se eles conterão data, hora ou ambos. Quando o campo é do tipo enumerado, é possível anotá-lo com a anotação @*Enumerated*, para salvá-lo como texto com o valor da enumeração, por padrão é salvo o índice da enumeração no banco de dados.

Quando é necessário referenciar outra entidade, anota-se o campo com o tipo de relação entre as entidades, usando @*OneToOne* para relações um-para-um, @*OneToMany* para relações um-para-muitos e @*ManyToOne*, para relações muitos-para-um. Caso seja necessário, também é possível utilizar a anotação @*ManyToMany*, para definir relações muitos-para-muitos. Também é necessário anotar o campo com a forma como será feita a relação entre as entidades. Pode ser realizada por meio de chave estrangeira, pela anotação @*JoinColumn*, ou no caso de listas, é possível criar uma tabela de ligação, utilizando a anotação @*JoinTable*.

A Listagem 3 contém a classe *HistoricalData*, que exemplifica o uso das anotações. Nessa classe é possível verificar a presença da anotação @*Where*, que pré-define um filtro para todas as consultas da entidade. Nesse caso, o filtro é para que sejam retornadas somente as entidades cujas quais o campo *deleted* seja nulo ou falso. É utilizado isso, pois ao efetuar a exclusão de alguma entidade, efetuamos o soft-delete das mesmas, ou seja, a entidade não é apagada do banco de dados, apenas marcada como tal, para que não seja mais mostrada na aplicação. É realizado desta maneira pois o JPA tem algumas limitações no gerenciamento de chaves estrangeiras, o que pode ocasionar problemas, caso uma entidade seja apagada do banco de dados. Porém, a utilização dessa marcação permite que dados sejam restaurados facilmente, em caso de exclusão acidental, sendo necessário somente setar o campo *deleted* como falso.

Listagem 3 - Classe HistoricalData

```
@Entity
@Where( clause = " ( (deleted is null) or (deleted=false) )" )
public class CropSettings extends AbstractModel {

  @OneToOne
  @JoinColumn( name = "crop_id", referencedColumnName = "id" )
  @JsonIgnore
  @JsonDeserialize
  private Crop crop;

  private Double waterPH;

  @Column
  private Long millisToVerify;

  @OneToMany( orphanRemoval = true, cascade = CascadeType.ALL, mappedBy =
"cropSettings" )
  private List<CropSchedules> schedules = new ArrayList<>();
}
```

Fonte: Autoria própria.

4.4.4 Codificação do Microcontrolador

Esta seção apresenta a codificação para o microcontrolador e para a leitura dos sensores.

A Listagem 4 apresenta o mapeamento das portas do microcontrolador. Pelo fato das mesmas não seguirem uma sequência numérica, é necessário mapeá-las manualmente. A referência entre os números de portas e sua referência visual impressa no microcontrolador pode ser encontrada na documentação disponibilizada pelo fabricante de cada placa.

Listagem 4 - Mapeamento das portas do microcontrolador NodeMCU

```
#ifndef NODEMCU_H
#define NODEMCU_H

#define D0    16
#define D1    5
#define D2    4
#define D3    0
#define D4    2
#define D5    14
#define D6    12
#define D7    13
#define D8    15
#define D9    3
#define D10   1

#endif
```

Fonte: Autoria própria.

A Listagem 5 representa a codificação para leitura dos valores do sensor de temperatura e umidade DHT22, apresentado no Quadro 1. Esta codificação é implementada no

microcontrolador. O método *setup()* configura a inicialização do microcontrolador e o método *loop* contém o código que faz a leitura dos dados do sensor. A variável *dht* faz a comunicação com o sensor, e sua implementação é feita pelo fabricante do sensor. O método *readHumidity()* retorna a umidade relativa, e o método *readTemperature()* retorna a temperatura atual do sensor.

Listagem 5 - Leitura de Temperatura e Umidade utilizando DHT22

```
#include <DHT.h>;
#define DHTPIN 7            //Porta do Microcontrolador utilizada
para conexão
#define DHTTYPE DHT22       // DHT 22  (AM2302)
DHT dht(DHTPIN, DHTTYPE);   //Inicialização do DHT22
float hum;  //Guarda o valor da umidade
float temp; //Guarda o valor da temperatura

void setup(){
  Serial.begin(9600);
  dht.begin();
}
void loop(){
    delay(2000);
    //Ler variáveis de umidade e temperatura
    hum = dht.readHumidity();
    temp= dht.readTemperature();
}
```

Fonte: Autoria própria.

A

Listagem **6** apresenta a codificação utilizada para fazer a medição do sensor de pH que é implementada no microcontrolador. Para fazer o cálculo do pH são realizadas 10 leituras de valor do sensor, utilizando o método *analogRead()*.

Esses 10 valores são ordenados do menor para o maior, e então o maior e menor valor são descartados. Estes dois valores são descartados pois o sensor demora a estabilizar e trazer leituras consistentes de valores. Os valores restantes são somados e é feito uma média dos mesmos. Essa média de valores é utilizada para calcular o resultado em *millivolts*, utilizando a Fórmula 1. Então é necessário converter de *millivolt* para pH, utilizando a Fórmula 2.

Fórmula 1 - Conversão da média de valores para *millivolt*

$$\frac{\frac{média \times 5}{1024}}{6}$$

Fonte: PH4502C

Fórmula 2 - Conversão de millivolt para pH

$$millivolts \times 3.5$$

Fonte: PH4502C

Listagem 6 - Leitura de pH utilizando o sensor PH4502C

```
//pH meter Analog output to Arduino Analog Input 0
#define SensorPin 0
unsigned long int avgValue;  //Armazena o valor médio do pH
float b;
int buf[10],temp;
void setup(){
  pinMode(13,OUTPUT);
  Serial.begin(9600);
}

void loop(){
  //Ler 10 valores do sensor para fazer uma média entre eles
  for(int i=0;i<10;i++){
    buf[i]=analogRead(SensorPin);
    delay(10);
  }
  //Ordena os valores do menor para o maior
  for(int i=0;i<9;i++){
    for(int j=i+1;j<10;j++){
      if(buf[i]>buf[j]){
        temp=buf[i];
        buf[i]=buf[j];
        buf[j]=temp;
      }
    }
  }
  avgValue=0;
  //Pega o valor médio dos 6 valores do meio
  for(int i=2;i<8;i++){
      avgValue+=buf[i];
  }
  //Converte o valor analógico para millivolt
  float phValue=(float)avgValue*5.0/1024/6;
  //Converte de millivolt para pH
  phValue=3.5*phValue;
  delay(800);
  digitalWrite(13, LOW);
}
```

Fonte: Autoria própria.

A Listagem 7 apresenta a codificação utilizada para a medição do sensor de temperatura da água. No método *setup,* é chamado o método *sensors.begin()* para inicializar a biblioteca de sensores disponibilizada pelo fabricante do mesmo. No método *loop(),* primeiro é chamado o método *sensors.requestTemperatures(),* para solicitar à biblioteca que efetue a leitura dos valores do sensor, e então o método *sensors.getTempCByIndex()*, que retorna a temperatura já convertida para *Celsius*.

Listagem 7 - Leitura da Temperatura da Água

```
#include <OneWire.h>
#include <DallasTemperature.h>
#define ONE_WIRE_BUS D6;
void setup() {
    Serial.begin(9600);
    sensors.begin();
}

void loop() {
    sensors.requestTemperatures();
    temp = sensors.getTempCByIndex(0);
    Serial.println(temp);
}
```

Fonte: Autoria própria.

A Listagem 8 apresenta o código utilizado para a leitura do nível de água do sistema. No método *setup*() define-se que a porta do microcontrolador estará em modo *INPUT_PULLUP*, utilizando o método *pinMode*(). Para efetuar a leitura do sensor, no método *loop*(), é chamado o método *digitalRead*(), que fará a leitura do estado da porta que o sensor está conectado.

Listagem 8 - Nível da água

```
#define LEVEL_SENSOR D7
int level;
void setup(){
    Serial.begin(9600);
    pinMode(LEVEL_SENSOR, INPUT_PULLUP);
}

void loop(){
    level = digitalRead(LEVEL_SENSOR);
}
```

Fonte: Autoria própria.

A Listagem 9 apresenta a codificação utilizada para acionar e desativar os atuadores. Primeiro, a porta do atuador é definida como *OUTPUT*, no método *setup(),* isso permitirá que o estado seja escrito para a porta, ao contrário do que ocorre quando a porta está em modo *INPUT_PULLUP*. Para ativar o atuador, é necessário definir o estado da porta como *HIGH*, e

para desligar, como *LOW*. Para definir o estado de cada porta, é utilizado o método *digitalWrite()*.

Listagem 9 - Ativar/Desativar Atuadores

```
#define RELAY1 D5;
int state;
void setup(){
    Serial.begin(9600);
    pinMode( RELAY1, OUTPUT );
    state = LOW;
}

void loop(){
    if( state==HIGH ) {
        digitalWrite(RELAY1, LOW); //Para desligar
        state = LOW;
    }else{
        digitalWrite(RELAY1, HIGH); //Para ligar
        state = HIGH;
    }
}
```

Fonte: Autoria própria.

A Listagem 10 apresenta a codificação para configurar conexão do microcontrolador com o *broker* MQTT. As chamadas aos métodos *Serial.print()* ou *Serial.println()* servem para imprimir texto na porta serial onde o microcontrolador está conectado, a qual funciona como uma espécie de *feedback* ou *log* quando está vinculada a IDE de desenvolvimento Arduino, tornando assim o *debug* das aplicações mais fácil.

Para configurar a conexão, é necessário copiar o endereço do *broker*, e passa-lo, juntamente com a porta para o método *MQTT.setServer()*. O método *MQTT.setCallback()* é utilizado para definir o método que será chamado ao se receber uma mensagem em qualquer tópico que a aplicação esteja inscrita.

Listagem 10 - Configuração da Conexão

```
void mqttSetup(MQTTConnectionData data) {
    Serial.println();
    Serial.print("Connecting to MQTT on ");
    Serial.print(data.broker().c_str());
    Serial.print(":");
    Serial.print(data.port());
    Serial.println();
    char broker[data.broker().size() + 1];
    strcpy(broker, data.broker().c_str());
    MQTT.setServer(broker, data.port());   //informa qual broker e
porta deve ser conectado
    MQTT.setCallback(mqtt_callback);
}
```

Fonte: Autoria própria.

A Listagem 11 apresenta o código utilizado para manutenção da conexão com o *broker*. Ao entrar no método, em um *loop*, é verificado se o MQTT possui conexão com o *broker*. Caso não haja conexão, o MQTT é reconfigurado e o método *MQTT.connect()* é chamado, passando como parâmetro o identificador da conexão, bem como usuário e senha utilizados para conectar. Quando a conexão é efetuada com sucesso, a aplicação então se inscreve nos tópicos de seu interesse, chamando o método *subscribeToTopics(),* e então invoca o método *publishToConfig()* para enviar uma requisição ao servidor, solicitando as configurações a serem utilizadas por aquele dispositivo.

Listagem 11 - Manutenção da Conexão

```
void reconnectMQTT(MQTTConnectionData data) {
    while (!MQTT.connected()) {
        Serial.println("Connecting to MQTT");
        char broker[data.broker().size() + 1];
        strcpy(broker, data.broker().c_str());
        Serial.println(data.broker().c_str());
        MQTT.setServer(broker, data.port());
        Serial.println("BEFORE SET CALLBACK");
        MQTT.setCallback(mqtt_callback);
        Serial.println("AFTER SET CALLBACK");
        if (MQTT.connect(data.identifier().c_str(), "pnktyzmy",
"5fzaDbMmpFxN")) {
            Serial.println("Connected to MQTT");
            subscribeToTopics( data );
            publishToConfig(data);
        } else {
            Serial.println("Falha ao reconectar no broker.");
            Serial.println("Havera nova tentatica de conexao em 2s");
            delay(2000);
        }
    }
}
```

Fonte: Autoria própria.

Para efetuar a troca de dados entre o microcontrolador e o *broker*, é necessário conectá-lo à mesma rede que o servidor está, ou, caso o *broker* esteja hospedado na internet, é necessário conectar o microcontrolador à uma rede com acesso à internet. A conexão utilizada pelo NodeMCU é sem fio, e a Listagem 12 mostra o código utilizado para conectá-lo à rede *wireless*. Para efetuar a conexão, é chamado o método *WiFi.begin(),* passando como parâmetro o nome da rede, e a senha utilizada para conectar. Então aguarda-se até que a conexão seja efetuada, nesse ponto o *status* da conexão será *WL_CONNECTED*.

Listagem 12 - Configuração da conexão wireless

```
void reconnectWifi(WifiConnectionData data) {
    if (WiFi.status() == WL_CONNECTED) {
        return;
    }
    Serial.println("Connecting WiFi");
    WiFi.begin(data.ssid().c_str(), data.password().c_str());
    while (WiFi.status() != WL_CONNECTED) {
        delay(100);
    }
    Serial.println("Connected WiFi");
}
```

Fonte: Autoria própria.

A Listagem 13 representa o código necessário para efetuar o envio de uma mensagem a um tópico do *broker*. Para isso é necessário efetuar a chamada do método *publish()* do MQTT, passando como parâmetro o tópico ao qual estamos postando a mensagem, o *buffer* com os dados, e o tamanho da mensagem.

Listagem 13 - Envio de mensagem ao *broker*

```
void sendMessage(DynamicJsonDocument data, std::string const &topic) {
    char buffer[512];
    size_t n = serializeJson(data, buffer);
    char c_topic[topic.size() + 1];
    strcpy(c_topic, topic.c_str());
    Serial.println(c_topic);
    Serial.println(buffer);
    if (getMQTT().publish(c_topic, buffer, n)) {
        Serial.println("Data sent to broker");
    } else {
        Serial.println("Erro ao publicar mensagem");
    }
}
```

Fonte: Autoria própria.

4.4.5 Codificação do Aplicativo

Esta seção apresenta a codificação utilizada para desenvolver o aplicativo. O desenvolvimento de aplicações Flutter é baseado em *Widgets*, que são objetos utilizados para representar os dados em tela. O desenvolvimento em Flutter baseia-se em alguns conceitos de desenvolvimento funcional, que utiliza funções para seu fluxo de trabalho, diferente da programação orientada a objetos.

No Flutter há dois tipos de *Widgets*, os com estado, ou *Stateful*, e os sem estado, ou *Stateless*. Os *Widgets* no Flutter são imutáveis, o que muda são seus estados. Portanto, os *Widgets* sem estado tornam-se imutáveis por padrão, não sendo possível alterá-los após sua inicialização.

Os *Widgets* com estado, requerem que seja implementado um estado (*State*) para eles, e este estado pode ser alterado chamando a função *setState*(). Ao contrário da programação orientada a objetos, ao alterar o estado de um *Widget*, a alteração não é pontual, somente em uma propriedade. O *Widget* inteiro é instanciado novamente com o novo estado.

A Listagem 14 mostra a classe *SysponicsApp*, que é a classe principal do projeto. Essa classe é uma classe do tipo *Stateless*, e ela gerencia a inicialização, os temas e as rotas internas do aplicativo. As rotas são utilizadas para navegar entre as telas do aplicativo, cada tela tem uma rota única.

Listagem 14 - Classe SysponicsApp

```
void main() => runApp(SysponicsApp());

class SysponicsApp extends StatelessWidget {
  initLocale() async {
    await initializeDateFormatting('pt_BR');
  }

  @override
  Widget build(BuildContext context) {
    Intl.defaultLocale = 'pt_BR';
    initLocale();

    return MaterialApp(
      title: 'Sysponics',
      theme: ThemeData(
        brightness: Brightness.light,
        primaryColor: CustomColors.APP_BRAND_COLOR,
        hintColor: CustomColors.APP_BRAND_COLOR,
      ),
      initialRoute: LoginForm.route,
      routes: {
        LoginForm.route: (context) => LoginForm(),
        ProjectListForm.route: (context) => ProjectListForm(),
        ProjectPage.route: (context) => ProjectPage(),
        ProjectForm.route: (context) => ProjectForm(),
        DeviceListForm.route: (context) => DeviceListForm(),
        DeviceForm.route: (context) => DeviceForm(),
        ProjectPageCropsWidget.route: (context) => ProjectPageCropsWidget(),
        CropHistoricalDataWidget.route: (context) =>
CropHistoricalDataWidget(),
        ProjectSettingsPage.route: (context) => ProjectSettingsPage(),
        ProjectNewCropPage.route: (context) => ProjectNewCropPage(),
        SpeciesList.route: (context) => SpeciesList(),
        SpeciesForm.route: (context) => SpeciesForm(),
        ProjectHarvestCropPage.route: (context) => ProjectHarvestCropPage(),
      },
    );
  }
}
```

Fonte: Autoria própria.

O desenho de uma tela no Flutter é desenvolvido por meio de *Widgets*, no qual vários *widgets*, aninhados um dentro do outro, definem a disposição dos campos e componentes visuais dentro da tela.

Uma tela tem como componente pai um *Scaffold*, que guarda todas as propriedades que uma tela padrão de um aplicativo possui. O parâmetro *body* do *Scaffold* contém o corpo da tela, ou seja, a parte onde são mostrados os dados na tela. Esse parâmetro é obrigatório.

Em um *Scaffold* é possível também definir um *AppBar*, que é a barra superior da tela, que é geralmente encontrada em aplicativos, e que tem como função principal a navegação entre telas, e acesso a menus. O *AppBar* não é obrigatório para uma tela. No *AppBar* podem ser definidos os botões de ação e o título da tela.

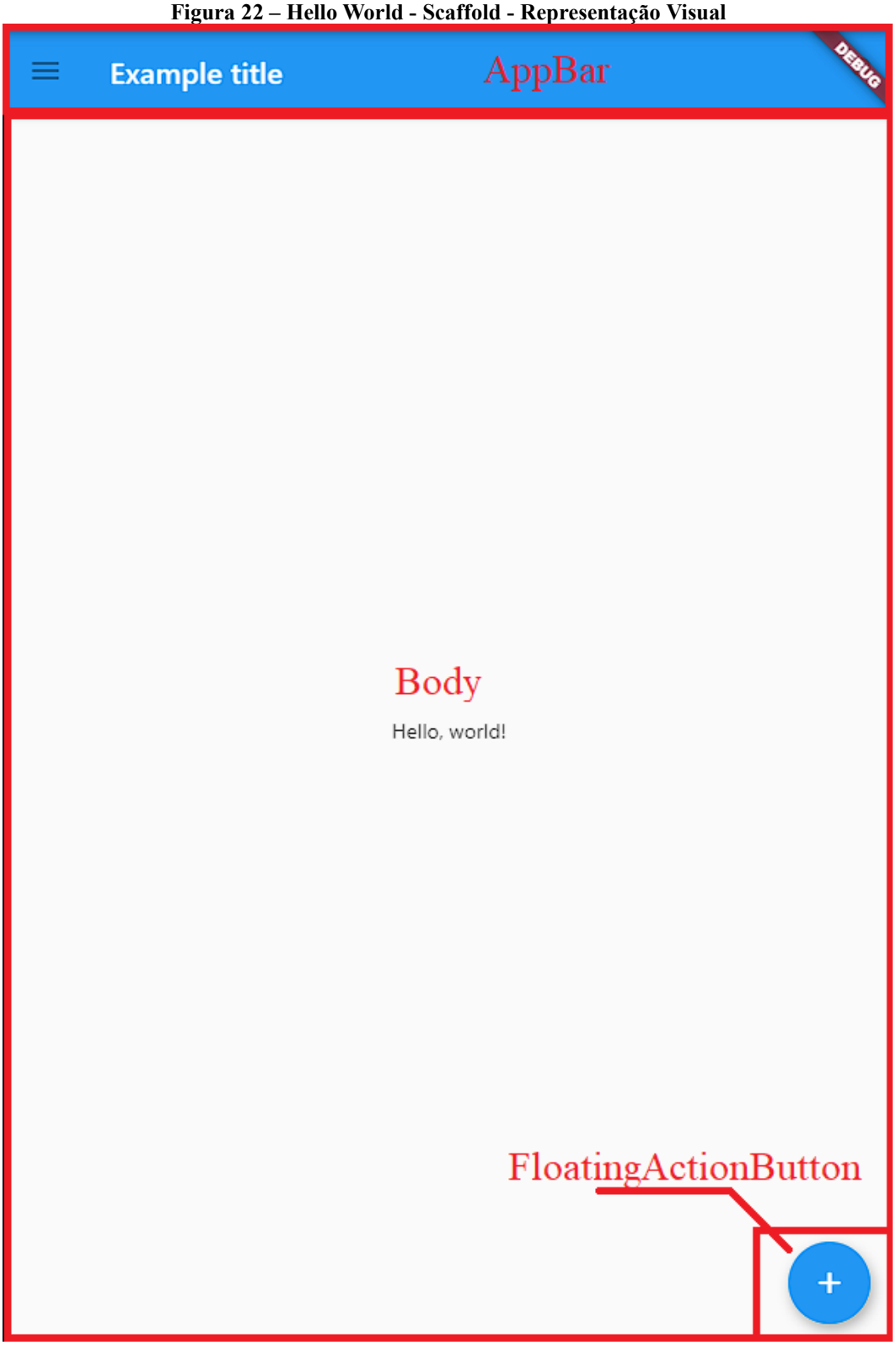
A Listagem 15 apresenta um exemplo de como utilizar o *Scaffold* para montar uma tela contendo uma barra superior, um texto em seu centro escrito *Hello World*, e um botão em seu canto inferior direito. A Figura 22 mostra a representação visual desta tela.

Listagem 15 - Hello World - Scaffold

```
class TutorialHome extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        leading: IconButton(
          icon: Icon(Icons.menu),
          tooltip: 'Navigation menu',
          onPressed: null,
        ),
        title: Text('Example title'),
        actions: <Widget>[
          IconButton(
            icon: Icon(Icons.search),
            tooltip: 'Search',
            onPressed: null,
          ),
        ],
      ),
      body: Center(
        child: Text('Hello, world!'),
      ),
      floatingActionButton: FloatingActionButton(
        tooltip: 'Add',
        child: Icon(Icons.add),
        onPressed: null,
      ),
    );
  }
}
```

Fonte: FLUTTER (2020).

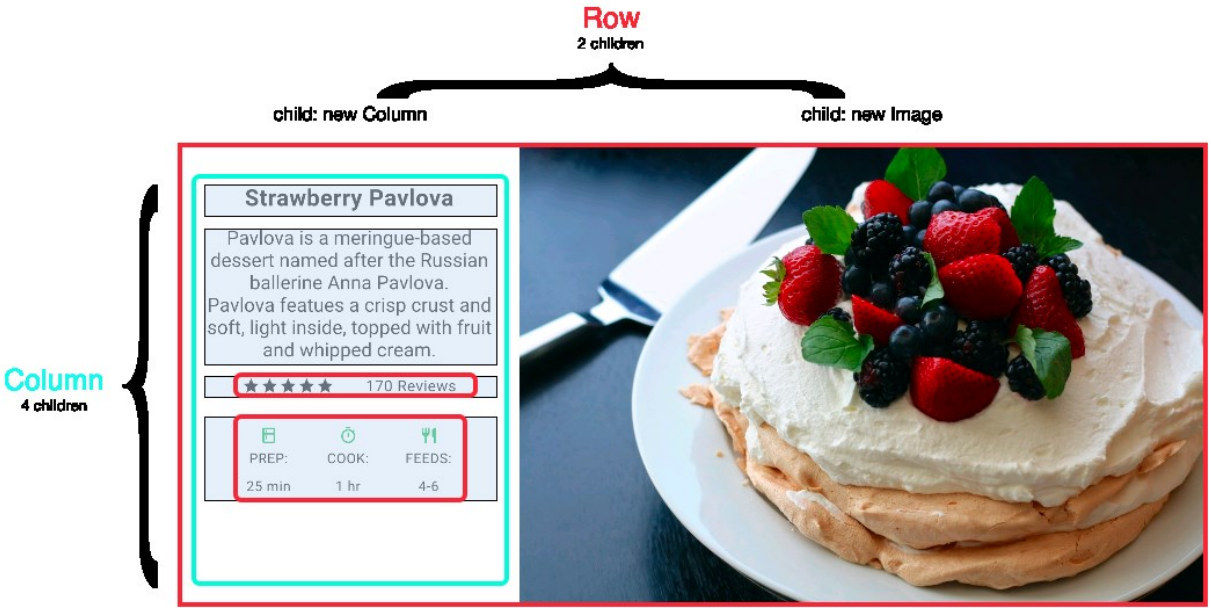
Figura 22 – Hello World - Scaffold - Representação Visual



Fonte: Autoria própria.

Para a organização dos componentes em tela, existem *widgets* auxiliares, que gerenciam o alinhamento e espaçamento dos itens, são eles: *Container, Column e Row*. O Container é um *widget* que tem apenas um filho, e gerencia os espaçamentos e margens no entorno deste item. O *Column* é um *widget* que possui uma lista de filhos, e forma uma coluna com esses filhos, colocando-os um embaixo do outro. O *Row* é um *widget* similar ao *Column*, mas ao invés de formar uma coluna, ele forma uma linha, colocando os filhos um ao lado do outro. A Figura 23 exemplifica os *widgets* *Row* e *Column*, onde os *Rows* estão circulados em vermelho e as Columns estão circulados em ciano.

Figura 23 - Rows e Columns - Visualização



Fonte: FLUTTER (2020).

Para listar objetos, há dois *Widgets* principais, o *ListView* e o *PagedListView*. O *ListView* serve para listas simples, onde sabe-se de antemão o tamanho da lista. Já o *PagedListView* monta a lista de forma paginada. É útil para trazer dados de um servidor, onde não se sabe o tamanho total da lista de antemão, e onde também é possível diminuir os tempos de requisição e resposta, visto que ao paginar os dados, o fluxo de informação entre cada página acaba se reduzindo.

A Listagem 16 representa a codificação necessária para implementar um *PagedListView*. É necessário ter um *PagingController*, que é o componente que gerenciará a transição entre uma página e outra. O *PagingController* precisa de um *PageRequestListener*, que é chamado toda vez que a página é mudada.

Listagem 16 - PagedListView

```
final PagingController<int, SpeciesDTO> _pagingController =
PagingController(firstPageKey: 0);

@override
void initState() {
  _pagingController.addPageRequestListener((pageKey) {
    fetchPage(pageKey);
  });
  super.initState();
}

void fetchPage(int p) {
  SpeciesDataRetriever().speciesOfLocale(page: p).then((list) {
    final isLastPage = list.length < Constants.DEFAULT_PAGE_SIZE;
    //Se for a última página, chama o appendLastPage, senão chama o
appendPage
    if (isLastPage) {
      _pagingController.appendLastPage(list);
    } else {
      _pagingController.appendPage(list, p + 1);
    }
  }).catchError((error) {
    _pagingController.error = error;
  });
}

Widget pagedListView() {
  return PagedListView<int, SpeciesDTO>(
    pagingController: _pagingController,
    builderDelegate: PagedChildBuilderDelegate<SpeciesDTO>(
      itemBuilder: (context, item, index) {
        return ListTile(
          title: Text(item.commonName),
        );
      },
    ),
  );
}
```

Fonte: Autoria própria.

Para desenhar uma tela de formulário, utiliza-se o *Widget Form*. O *Form* precisa de uma chave, definida no parâmetro *key*, que é um objeto do tipo *GlobalState<Form>*. Esta chave é utilizada para fazer as validações do Form.

Os campos a serem preenchidos dependem do tipo de dados que eles representarão. Para texto e número, é utilizado o *widget TextFormField*. Para campo tipo data, é utilizado o *DateFormField*, e para campos onde é necessário selecionar um valor de uma lista, é utilizado o *DropDownButton*. Os campos do tipo *TextFormField* necessitam de um *TextEditingController* para gerenciar o preenchimento dos dados em seus objetos. Os campos do tipo *DateFormField* e *DropDownButton* preenchem os dados em seus objetos no método *onChange*. Nesse caso é necessário chamar o método *setState* para que a alteração surta efeito.

A Listagem 17 representa a codificação utilizada para montar uma tela de formulário. O parâmetro *validator* serve para definir um método de validação do conteúdo do campo. Este método será chamado ao fazer a validação do formulário, que é realizada invocando o método *currentState.validate*() da chave do formulário.

Listagem 17 - Formulário

```
final _formKey = GlobalKey<FormState>();
Species _species = Species(commonNames: List.empty(growable: true));
List<SpeciesNames> _names = List.empty(growable: true);

final TextEditingController _cientificNameFilter = TextEditingController();
final TextEditingController _commonNameFilter = TextEditingController();

@override
void initState() {
  addListeners();
}

void addListeners() {
  _cientificNameFilter.addListener(() {
    _species.cientificName = _cientificNameFilter.text.isEmpty ? '' :
_cientificNameFilter.text;
  });
  _commonNameFilter.addListener(() {
    if (_names.length == 0) {
      _names.add(SpeciesNames(locale: 'pt_br', commonName: _commonNameFilter.text.isEmpty
? '' : _commonNameFilter.text));
    } else {
      _names[0].commonName = _commonNameFilter.text.isEmpty ? '' : _commonNameFilter.text;
    }
  });
}

@override
Widget build(BuildContext context) {
  if (ModalRoute.of(context).settings.arguments != null) {
    _species = ModalRoute.of(context).settings.arguments;
    _cientificNameFilter.text = _species.cientificName;
    _names = _species.commonNames;
  }
  return Scaffold(
    backgroundColor: CustomColors.APP_BACKGROUND,
    appBar: AppBar(
      title: Text('Espécie'),
      actions: [
        IconButton(
          icon: Icon(Icons.save),
          onPressed: () {
            if (_formKey.currentState.validate()) {
              _species.commonNames = _names;
              SpeciesDataRetriever().save(_species).then((value) {
                Navigator.pop (context);
              }).catchError((error) {
                Scaffold.of(context).showSnackBar(
                  SnackBar(
                    content: Text(error),
                  ),
                );
              });
            }
          },
        )
      ],
    ),
    body: Form(
```

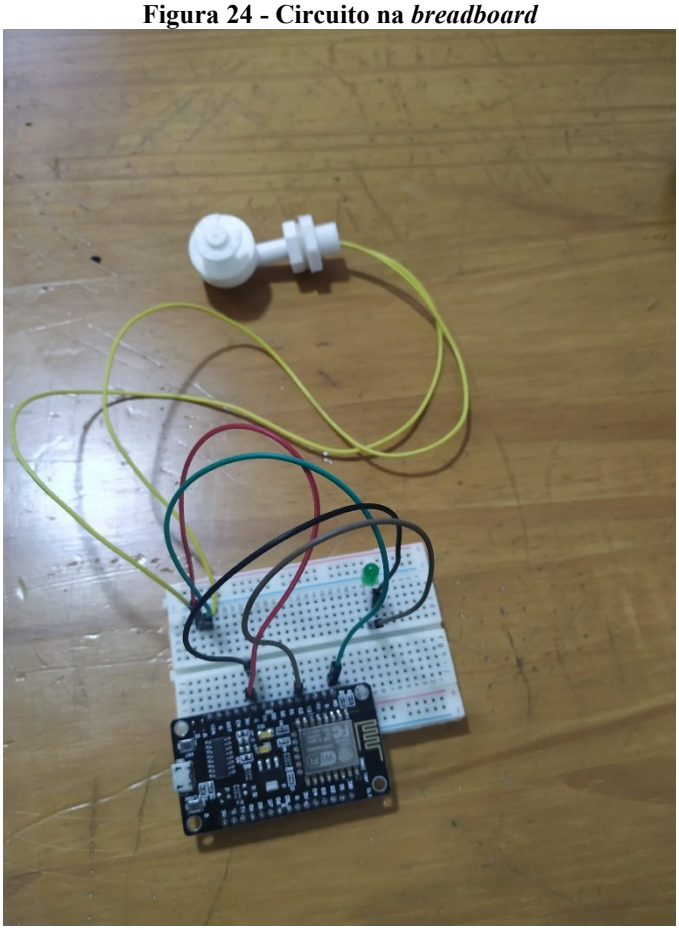
Fonte: Autoria própria.

4.5 HARDWARE E PROTOTIPAGEM

Nesta sessão são apresentadas partes da prototipagem e da montagem do hardware utilizado no projeto.

A Figura 24 - Circuito na *breadboard* mostra o circuito montado em uma *breadboard.* Uma *breadboard* é uma pequena placa de prototipagem, utilizada para efetuar testes rápidos do circuito. Ela permite que os parâmetros do circuito sejam alterados com facilidade, sem a necessidade de efetuar a solda dos componentes. A prototipagem do circuito de todos os componentes foi feita desta forma.

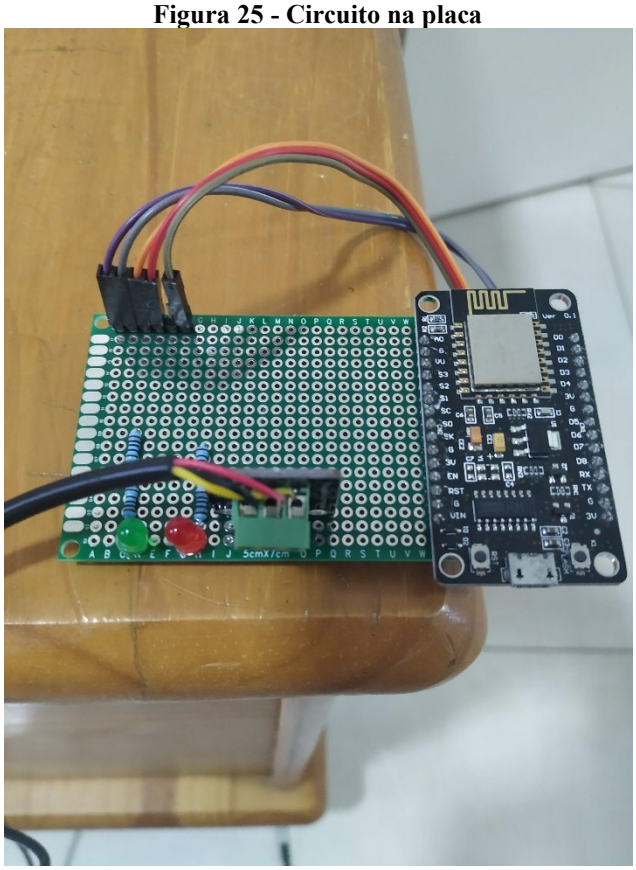
Figura 24 - Circuito na *breadboard*



Fonte: Autoria própria.

A Figura 25 - Circuito na placa mostra o circuito montado na placa com os componentes já soldados. Para a solda foi utilizado fio de estanho de 1mm, com o estanhador em cerca de 300ºC.

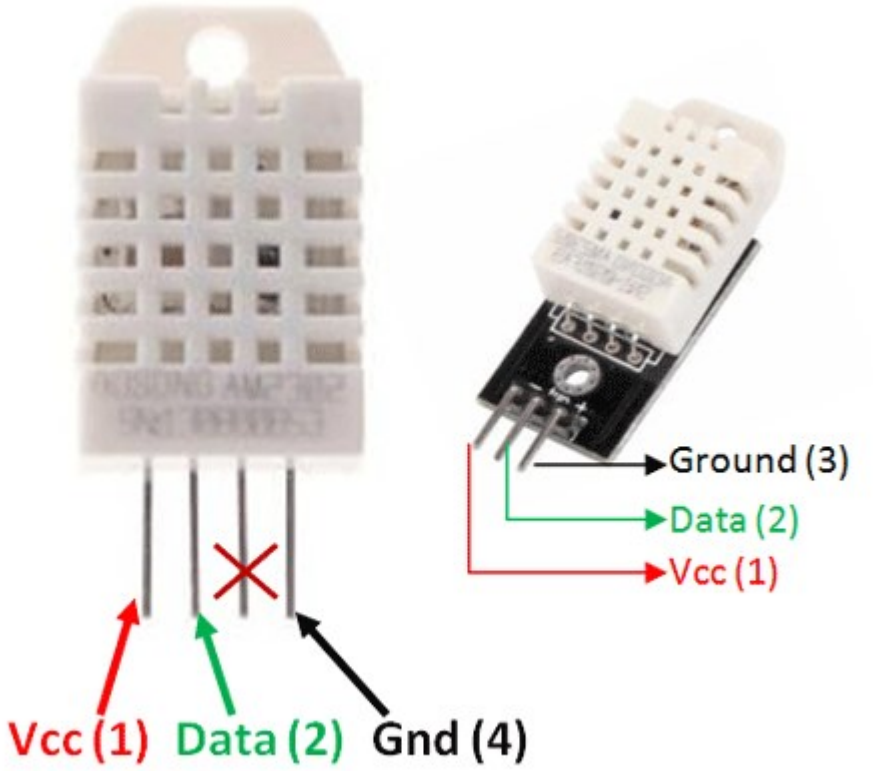
Figura 25 - Circuito na placa



Fonte: Autoria própria.

Os circuitos foram montados de acordo com a especificação de cada fabricante, que varia para cada tipo de sensor. O que é padrão para todos é que os pinos G ou GND são sempre ligados ao pino G/GND do microcontrolador. Esse pino é o terra, e caso haja mais de um componente que necessite aterramento no circuito, todos podem utilizar o mesmo pino. Os pinos V ou VCC são ligados nos pinos que possuem tensão constante no microcontrolador, respeitando sempre as recomendações dos fabricantes, para evitar a queima dos componentes. No caso do NodeMCU, os pinos que possuem tensão constante são os pinos marcados como 3.3 ou VIN. Esses pinos possuem tensão constante de 3.3 volts, o que é suficiente para alimentar os sensores mais simples. Para sensores que possuem um módulo que faça a interface entre o sensor e a placa controladora, pode ser necessário alimentação externa. Para isso é necessário consultar a especificação do fabricante e verificar o recomendado. No caso dos sensores deste projeto, o único que necessitou alimentação externa foi o sensor de pH.

Figura 26 - Sensor DHT22

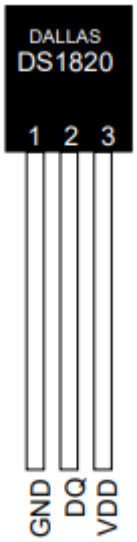


Fonte: Components 101 (2020).

Conforme a Figura 26, para efetuar a ligação do sensor DHT22, utilizado para medir temperatura e umidade do ar, é necessário ligar o *Ground*/*GND* ao pino G/GND do microcontrolador, o pino VCC deve ser ligado ao pino 3.3v do NodeMCU, e o pino Data deve ser ligado ao pino lógico do microcontrolador. No caso do NodeMCU, esses pinos são enumerados de D0 a D10.

Para o sensor DS18B20, que é o sensor de temperatura da água utilizado neste projeto, segundo a FFF, para efetuar a leitura dos dados, é necessário ligar o pino GND ao GND do microcontrolador, o pino VCC à porta 3.3V do NodeMCU, e o pino DQ à porta lógica do microcontrolador.

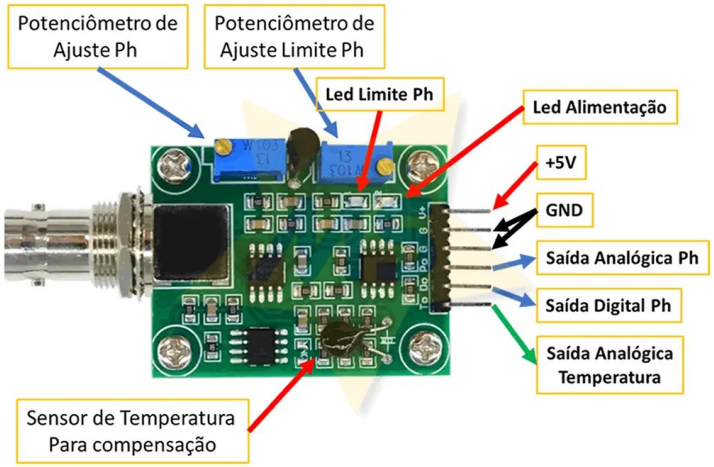
Figura 27 - Sensor DS1820B



Fonte: DALLAS (2019).

Conforme a Figura 27 para fazer a ligação do sensor PH4502C, é necessário ligar uma fonte de alimentação externa no pino 5V, ligar os pinos GND ao pino do GND do microcontrolador, ligar a saída digital à porta lógica do NodeMCU. Este sensor possui também uma saída analógica para efetuar as leituras, caso esta seja usada, é necessário também ligar a porta da saída analógica da temperatura à uma das portas logicas do microcontrolador. Essa temperatura será utilizada no cálculo para a compensação no cálculo do pH.

Figura 28 - Sensor PH4502C



Fonte: MARINOSTORE (2020).

5 CONCLUSÃO

O objetivo deste trabalho foi efetuar o planejamento e desenvolvimento de uma aplicação para automação e monitoramento de um sistema hidropônico, visando a modularização e o baixo custo de implantação do sistema.

Após o levantamento dos requisitos foi desenvolvido os casos de uso, a modelagem do banco de dados e a prototipagem da aplicação. Posteriormente, foi desenvolvida a estrutura do servidor, as classes de persistência e os *controllers*. Também foram desenvolvidas as rotinas para leitura da temperatura e umidade do ar, e para medição do pH da Solução Nutritiva. Foi também desenvolvido o protótipo e a implantação dos circuitos necessários para integrar o hardware e enviar os dados dos sensores ao servidor. Por fim, foi desenvolvido uma aplicação para *smartphones*, para que seja possível efetuar a consulta dos dados e a configuração do sistema.

A realização deste trabalho possibilitou grande aprendizado, devido à pesquisa efetuada sobre os sistemas hidropônicos de cultivo, sobre a automação utilizando microcontroladores, e com o desenvolvimento de aplicações utilizando os conceitos de IoT.

As principais dificuldades encontradas no desenvolvimento, além da extensão do projeto, que envolve três componentes principais: aplicação, servidor e hardware, foi o fato da necessidade de se utilizar três linguagens de desenvolvimento para a realização deste projeto, bem como a necessidade de aprender os conceitos de hardware e eletrônica para ser possível a implementação deste sistema. Estas dificuldades foram superadas por meio de pesquisa e experimentação.

Apesar do projeto haver suas limitações, conclui-se que o mesmo atingiu o objetivo proposto e, para trabalhos futuros, há a possibilidade de efetuar a integração com mais tipos de sensores, para tornar as medições mais precisas, e aprimorar a aplicação, para que os usuários possam compartilhar suas experiências entre eles. Também, para trabalhos futuros, sugere-se a implementação de um *dashboard*, para visualização dos dados, utilizando gráficos para uma visualização histórica dos dados obtidos, e também a possibilidade de gerar relatórios para o monitoramento dos dados.

REFERÊNCIAS

ALBRIGHT, L. D.; LANGHANS, R.W. **Controlled environmental agriculture scoping study.** Ithaca, EUA, set. 1996.

AOSONG Electronics, **Digital-output relative humidity & temperature sensor/module DHT22 (DHT22 also named as AM2302).** Disponível em: https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf. Acesso em: 06 mai. 2019.

ARDUINO, **What Is Arduino?** Disponível em: https://www.arduino.cc/en/Guide/Introduction. Acesso em: 06 jun. 2019.

AURELIO, **Novo Dicionário Aurélio da Língua Portuguesa**, Novo Dicionário Eletrônico Aurélio versão 5.0, Editora Positivo, 2004.

BARBOSA, G. L. et al. **Comparison of land, water, and energy requirements of lettuce grown using hydroponic vs. conventional agricultural methods**. International Journal of Environmental Research and Public Health, Estados Unidos, 16 jun. 2015.

BRECHNER, Melissa; BOTH, A. J., **Hydroponic Lettuce Handbook**, Estados Unidos, Disponível em: http://cea.cals.cornell.edu/attachments/Cornell%20CEA%20Lettuce%20Handbook%20.pdf. Acesso em: 22 ago. 2018.

COETZEE, Louis; EKSTEEN Johan. **The Internet of Things – Promise for the Future? An Introduction**, IST-Africa, 2011. Disponível em: https://www.researchgate.net/publication/232168440_The_Internet_of_Things_--_Promise_for_the_future_An_Introduction. Acesso em: 06 mai. 2019.

COMPONENTS101, **DHT22 Sensor Pinout, Specs, Equivalents, Circuits & Datasheet**, Disponível em: https://components101.com/sensors/dht22-pinout-specs-datasheet. Acesso em: 05 nov. 2020.

DALLAS Semiconductor, **DS18B20 Programmable Resolution 1-Wire® Digital Thermometer.** Disponível em: https://cdn.sparkfun.com/datasheets/Sensors/Temp/DS18B20.pdf. Acesso em: 06 mai. 2019.

DELFIN, Alfredo Rodriguez, **Advances of Hydroponics in Latin America**, 2012. Disponível em: https://www.researchgate.net/publication/283869323_Advances_of_hydroponics_in_Latin_America. Acesso em: 06 mai. 2019.

DIYMORE, **PH4502C PH 4502C Liquid PH Value Detection Detect Sensor Module Monitoring Control For Arduino.** Disponível em: https://www.diymore.cc/collections/ph-value-detect-sensor/products/diymore-liquid-ph-value-detection-detect-sensor-module-monitoring-control-for-arduino-m. Acesso em: 06 mai. 2019.

FAO, Food and Agriculture Organization, **Social protection and agriculture: breaking the cycle of rural poverty.** Roma, 2015. Disponível em:

http://www.fao.org/fileadmin/user_upload/newsroom/docs/SOFA-in-Brief2015.pdf. Acesso em: 06 mai. 2019.

FAZION FILHO, Mauro, **Internet das Coisas**. Palhoça: UnisulVirtual, 2016. Disponível em: https://www.researchgate.net/profile/Mauro_Fazion_Filho/publication/319881659_Internet_das_Coisas_Internet_of_Things/links/59c038d5458515e9cfd54ff9/Internet-das-Coisas-Internet-of-Things.pdf. Acesso em: 22 ago. 2018.

FILIPEFLOP, **Módulo WiFi ESP8266 NodeMCU**, Disponível em: https://www.filipeflop.com/produto/modulo-wifi-esp8266-nodemcu-esp-12/. Acesso em: 16 jun. 2019.

FURLANI, P.R.; SILVEIRA, L.C.P.; BOLONHEZI, D.; FAQUIN, V. **Cultivo Hidropônico de Plantas: Parte 1 - Conjunto hidráulico.** 2009. Artigo em Hypertexto. Disponível em: http://www.infobibos.com/Artigos/2009_1/hidroponiap1/index.htm. Acesso em: 22 abr. 2019.

FLUTTER, **Flutter – Beautiful native apps in record time**, Disponível em: https://flutter.dev/. Acesso em: 05 nov. 2020.

GENESIS, Advanced Aeroponic System, 2019. Disponível em: http://www.genesisindia.net.in/aero.html. Acesso em: 13 jun. 2019.

GOOGLE **Meet Android Studio** Disponível em: https://developer.android.com/studio/intro. Acesso em: 06 jun. 2019.

GREEN AND VIBRANT, **Vertical Hydroponics – A Base Introduction**, 2019. Disponível em: https://www.greenandvibrant.com/vertical-hydroponics. Acesso em: 13 jun. 2019.

HUANG, Yinghui, **Descriptive Models for Internet of Things, Dalian**, China, 2010.

JENSEN, Merle H., **Hydroponics Worldwide – A Technical Overview**, 1999, Disponível em: https://doi.org/10.17660/ActaHortic.1999.481.87. Acesso em: 13 jun. 2019.

JETBRAINS **IntelliJ IDEA: The Java IDE for Professional Developers** Disponível em: https://www.jetbrains.com/idea/. Acesso em: 06 jun. 2019.

KAMPMAN, Bettina; BROWER, Femke; SCHEPERS, Benno **Agricultural land availability and demand in 2020**: A global analysis of drivers and demand for feedstock, and agricultural land availability. Delft, Holanda, jun. 2008.

MAKERLAB, **5v Relay Module**, 2019 Disponível em https://www.makerlab-electronics.com/product/relay-module-spdt/. Acesso em: 06 mai. 2019.

MARINOSTORE, **Sensor de PH com módulo PH4502C**, Disponível em: https://www.marinostore.com/sensores/sensores/sensor-de-ph-com-modulo-ph4502c. Acesso em: 05 nov. 2020.

NOSOILSOLUTIONS, **What is Nutrient Film Technique**, 2019. Disponível em: http://www.nosoilsolutions.com/nutrient-film-technique-nft-hydroponics. Acesso em: 13 jun. 2019.

POSTGRES, **PostgreSQL: The world's must advanced open source database.** Disponível em: https://www.postgresql.org/. Acesso em: 06 jun. 2019.

PRESSMAN, Roger S. **Engenharia de Software.** 6ª Edição. São Paulo: McGraw-Hill do Brasil, 2006.

PUNDIR, Yogita; SHARMA, Nancy; SINGH, Yaduvir **Internet of Things (IoT)**: challenges and future directions. International Journal of Advanced Research in Computer and Communication Engineering, India, v.5, n.3, mar. 2016.

SANTOS, B. P. et al. **Internet das coisas: da teoria à prática.** Minicursos SBRC-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, 2016.

SINGH, Dhananjay; TRIPATHI, Gaurav; JARA, Antonio J. **A Survey of Internet-of-Things: Future Vision, Architecture, Challenges and Services,** 2014 IEE Forum on Internet of Thigs, 2014.

SQLITE, **SQLite**, Disponível em: https://www.sqlite.org/index.html. Acesso em: 06 jun. 2019.

SPRING, **Spring Framework**, Disponível em: https://spring.io/projects/spring-framework. Acesso em: 05 nov. 2020.

THOMAZINI, Daniel; ALBUQUERQUE, Pedro Urbando Braga de, **Sensores Industriais – Fundamentos e Aplicações**, Editora Erica, 2005.

TOGNONI, F.; PARDOSSI, A., Soil-less culture for greenhouse crops in the mediterranean countries. In: LARDEREL, J. A. de et al. **Methyl Bromide Alternatives for North African and Southern European Countries.** Roma, Itália: UNEP, 26-29 mai. 1998.

UUIDGENERATOR, **Online UUID Generator Tool**, Disponível em: https://www.uuidgenerator.net/version4. Acesso em 18 jun 2019.

VISUAL PARADIGM, **Ideal Modelling & Diagramming Tool for Agile Team Collaboration**, Disponível em: https://www.visual-paradigm.com/. Acesso em: 06 jun. 2019.

WORLD WATER ASSESSMENT PROGRAMME (WWAP). **The united nations world water development Report 3:** water in a changing world. Paris, França, Londres: UNESCO, 2009.