Universidade Tecnológica Federal do Paraná
Curso de Engenharia Eletrônica

Matheus Bernardi da Silva

Protótipo de um Sistema Computacional para Automação da Irrigação por Aspersão de um Plantio

Toledo
2025

Matheus Bernardi da Silva

Protótipo de um Sistema Computacional para Automação da Irrigação por Aspersão de um Plantio

Prototype of a Computer System for Automating Sprinkler Irrigation of a Plantation

Trabalho de Conclusão de Curso apresentado à disciplina de Trabalho de Conclusão de Curso 2 do Curso de Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná - UTFPR Campus Toledo, como requisito parcial para a obtenção do título de Bacharel em Engenharia Eletrônica.

Orientador Prof. Dr. Alessandro Paulo de Oliveira

Toledo
2025

Matheus Bernardi da Silva

Protótipo de um Sistema Computacional para Automação da Irrigação por Aspersão de um Plantio

Trabalho de Conclusão de Curso apresentado à disciplina de Trabalho de Conclusão de Curso 2 do Curso de Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná - UTFPR Campus Toledo, como requisito parcial para a obtenção do título de Bacharel em Engenharia Eletrônica.

Trabalho aprovado. Toledo, 13 de fevereiro de 2025:

Prof. Dr. Alessandro Paulo de Oliveira
UTFPR-TD
Orientador

Prof. Dr. Evandro Marcos Kolling
UTFPR-TD

Prof. Dr. José Dolores Vergara Dietrich
UTFPR-TD

Toledo
2025

A folha de aprovação assinada encontra-se na coordenação do curso

RESUMO

Em um contexto de crescente preocupação com a eficiência no uso dos recursos hídricos, este projeto tem como objetivo desenvolver um protótipo de automação de um sistema para monitorar um sistema de irrigação e permitir que o usuário altere parâmetros em tempo real. Por meio da integração de um gateway com um ESP32-LoRa, juntamente com um servidor web conectado a um banco de dados e um aplicativo móvel, será viável monitorar o estado da irrigação em tempo real, de forma remota. Isso proporcionará ao usuário uma forma eficaz de gerenciar e otimizar o uso da água nas plantações, baseado em seus conhecimentos sobre o seu plantio.

Palavras-chave: Web server; Automação de irrigação; Lora.

ABSTRACT

In a context of growing concern about the efficiency in the use of water resources, this project aims to develop an automation system to monitor and act on an irrigation system. By integrating a gateway with ESP32 and LoRa communication, a web server integrated with a database and a mobile application, it will be possible to remotely monitor the irrigation status in real time. This will provide users with an effective way to manage and optimize water use in plantations. In addition to simplifying the monitoring process, this approach also contributes to the preservation of natural resources and the sustainability of agricultural production.

Keywords: Web server; Irrigation automation; Lora.

LISTA DE ILUSTRAÇÕES

Figura 1 – Microcontrolador PIC16F877a 18
Figura 2 – Sensor S12 23
Figura 3 – Aspersor Tramontina Pro 24
Figura 4 – Válvula Solenoide 12 VDC 1/2" 25
Figura 5 – Pinagem SX1262 26
Figura 6 – Pinagem *DevKit* ESP32 LoRa V3 28
Figura 7 – Diagrama de blocos do sistema 32
Figura 8 – Heltec ESP32 LoRa V3 atuando como controlador *gateway* 33
Figura 9 – Página HTML gerada pelo ESP32 para conexão ao Wi-Fi 34
Figura 10 – Heltec ESP32 LoRa V3 atuando como controlador do nó 35
Figura 11 – Solenoide e adaptador do tipo engate rápido prontos para instalação . . 36
Figura 12 – Ligação elétrica do microcontrolador do nó 36
Figura 13 – Fluxograma do *firmware* do *gateway* 38
Figura 14 – Fluxograma do *firmware* do nó 41
Figura 15 – Amostra de solo colhida 42
Figura 16 – Caracterização do sensor 43
Figura 17 – Regressão exponencial para caracterização do sensor 45
Figura 18 – Funcionamento da classe de eventos 49
Figura 19 – Telas do aplicativo: configuração do *gateway* e da irrigação 50
Figura 20 – Telas do aplicativo: cadastramento e listagem de nós 51
Figura 21 – Telas do aplicativo: exibição do histórico de umidade 52
Figura 22 – Integração do sistema 54
Figura 23 – Montagem do nó em campo 56
Figura 24 – Dados de umidade após o teste 57
Figura 25 – Válvulas solenoides servo-operadas para refrigeração 61

LISTA DE TABELAS

Tabela 1 – Ordens via LoRa para acionamento e desligamento da irrigação 39
Tabela 2 – Valores do ADC e umidade do solo 46

LISTA DE CÓDIGOS FONTE

Listagem 1 – Criação e exportação do *schema* do dado de umidade 30
Listagem 2 – Código para caracterização do sensor 44
Listagem 3 – Rotas configuradas na aplicação com o Express.js 47
Listagem 4 – Função do *Controller* para salvar dados de umidade 47

LISTA DE ABREVIATURAS E SIGLAS

A/D	Analógico/Digital
ADC	Conversor analógico-digital
API	*Application Programming Interface*
CPU	*Central Processing Unit*
CSV	*Comma-separated values*
DB	*Database*
DIY	*Do It Yourself*
EEPROM	*Eletrically Erasable-Programmable Read-Only Memory*
IDE	*Integrated Development Environment*
IoT	*Internet of Things*
IP	*Internet Protocol*
JS	*JavaScript*
NoSQL	Não relacional
ODM	*Object Document Mapping*
PER	*Packet Error Rate*
pH	Potencial Hidrogeniônico
PIC	*Peripheral Interface Controller*
Pub/Sub	*Publish–subscribe pattern*
PWM	*Pulse Width Modulation*
RAM	*Random Access Memory*
REST	*Representational State Transfer*
ROM	*Read-Only Memory*
ULA	Unidade lógica e aritmética
NVS	Non-Volatile Storage

SUMÁRIO

1 INTRODUÇÃO 11

2 OBJETIVOS 13

2.1 Objetivos específicos 13

3 JUSTIFICATIVA 14

4 REFERENCIAL TEÓRICO 15

4.1 Métodos e sistemas de irrigação 15

4.1.1 Métodos de irrigação 15

4.1.2 Sistemas 16

4.1.2.1 Gotejamento 16

4.1.2.2 Microaspersão 16

4.2 Componentes de sensoriamento e atuação 16

4.2.1 Sensor de umidade do solo 16

4.2.2 Solenóide 17

4.2.3 Microcontrolador 17

4.3 Tecnologias utilizadas 18

4.3.1 Linguagem C/C++ 18

4.3.2 Modulação LoRa 18

4.3.3 Node.js 19

4.3.4 Conversor A/D 19

4.3.5 Typescript 19

4.3.6 React Native 20

4.3.7 Banco de dados 20

4.4 Trabalhos relacionados 21

5 MATERIAS E MÉTODOS 22

5.1 Dispositivos e componentes 22

5.1.1 Sensor de umidade 22

5.1.2 Aspersor Tramontina 23

5.1.3 Válvula Solenoide 12 VDC . . . 24
5.1.4 Transceptor LoRa . . . 25
5.1.5 ESP32 . . . 27
5.1.6 Web server . . . 28
5.1.7 Gateway . . . 30
5.2 Metodologia . . . 31
5.2.1 Visão geral . . . 31
5.2.2 Projeto do *gateway* . . . 32
5.2.2.1 Configuração do gateway pelo usuário . . . 33
5.2.3 Projeto do nó de sensoriamento e atuação . . . 34
5.2.4 Projeto do firmware . . . 37
5.2.4.1 Firmware do *gateway* . . . 37
5.2.4.2 Firmware do nó sensor/atuador . . . 38
5.2.5 Caracterização do sensor de umidade S12 . . . 41
5.2.6 Projeto do *web server* . . . 46
5.2.7 Projeto do *app mobile* . . . 50
5.2.8 Integração do sistema . . . 52

6 RESULTADOS . . . 55
6.1 Teste de acionamento . . . 56
6.2 Principais dificuldades encontradas . . . 58

7 CONCLUSÃO . . . 59
7.1 Trabalhos futuros . . . 59

REFERÊNCIAS . . . 62

1 INTRODUÇÃO

A água desempenha um papel fundamental na sustentação da vida na Terra, sendo um recurso essencial para a sobrevivência de todos os organismos vivos. Desempenha um papel vital na formação de tecidos e de regulação térmica, sendo crucical inclusive para a produção de alimentos. Nesse sentido, a irrigação, que consiste em técnicas variadas, desde sistemas simples até métodos mais avançados, busca suprir as necessidades hídricas das culturas, contribuindo para maximizar o rendimento das colheitas e garantir a segurança alimentar. Essa prática não apenas nutre as plantas, mas também desempenha um papel crucial na sustentabilidade agrícola, ao minimizar os efeitos adversos de períodos de seca e garantir a utilização eficiente dos recursos hídricos disponíveis (TESTEZLAF, 2012).

Através da aplicação controlada de água, a irrigação garante o fornecimento adequado de umidade para as plantas, mesmo em períodos de escassez de chuva, reduzindo significativamente os riscos de quebra de safra por seca. Além disso, ao manter o solo em condições ideais de umidade, a irrigação promove o desenvolvimento saudável das plantas, aumentando sua resistência a estresses ambientais e possibilitando um incremento substancial na produtividade das culturas. Dados demosntram que a irrigação é responsável por uma parcela significativa da produção de alimentos em nível mundial e nacional, destacando seu papel crucial no suprimento das demandas alimentares da população (TESTEZLAF, 2012). Assim, investir em sistemas de irrigação eficientes é essencial para garantir a sustentabilidade e o crescimento do agronegócio, contribuindo para o desenvolvimento econômico e social do país.

Há de se destacar também as melhorias proporcionadas por um sistema de irrigação eficiente. Resumidamente, pode-se destacar as seguintes melhorias: incremento na quantidade de colheitas agrícolas, produção durante períodos entressafras, assegurando o suprimento de acordo com as demandas de água, e diminuindo os perigos de perdas devido à falta de chuvas. Os métodos para atingir esses efeitos, são quatro: aspersão, superfície, localizada e subterrânea. No desenvolvimento desse projeto será utilizado o método de irrigação localizada (TESTEZLAF, 2012).

O foco deste trabalho é a prototipação de um sistema para automatizar e monitorar a irrigação de um pequeno plantio, como por exemplo, um plantio doméstico. Essas plantações, especialmente em ambiente urbano, têm se tornado mais frequentes ao longo dos últimos anos (NEVES; RODRIGUES, 2017), e costumam ser baseadas completamente na intervenção manual do usuário. A implementação do protótipo envolverá a integração de várias tecnologias para fornecer um sistema de monitoramento e controle da irrigação. O ESP32-LoRa será utilizado para o desenvolvimento de um *gateway*, responsável pela comunicação sem fio com os nós sensor/atuador distribuídos no campo. Estes nós consistirão em sensores de umidade do solo para monitorar as condições de umidade e controle

da irrigação de forma automática. Além disso, o ESP32 será responsável por se comunicar com o servidor *web* em *Node.js*, transmitindo os dados coletados para armazenamento e processamento. O servidor web, o cérebro do sistema, fornecerá uma interface de máquina para integração de um aplicativo de usuário acessível para visualização e gerenciamento dos dados de irrigação. Por fim, o aplicativo móvel baseado em *React Native* permitirá que os usuários monitorem e controlem remotamente o sistema de irrigação por meio de seus dispositivos móveis, podendo ajustar as configurações de irrigação conforme necessário. Essa integração de tecnologias proporcionará um sistema eficiente para otimizar o uso da água na agricultura, melhorando a produtividade e a sustentabilidade das culturas.

2 OBJETIVOS

O objetivo deste trabalho é desenvolver um protótipo de um sistema de monitoramento e controle computacional da irrigação para um plantio de dimensões reduzidas. O sistema visa proporcionar aos usuários uma solução eficaz e automatizada para gerenciar o uso da água no plantio, permitindo o monitoramento remoto das condições do solo, o controle da irrigação e a tomada de decisão com a informação sobre o cultivo.

2.1 OBJETIVOS ESPECÍFICOS

Para materializar o protótipo, o trabalho foi dividido em tópicos:

1. Desenvolver uma aplicação *backend* em *Node.JS* (*web server*), com exposição de API REST para o *gateway* e o aplicativo *mobile*.
2. Modelar um banco de dados (DB) para armazenamento de dados.
3. Desenvolver um aplicativo *mobile* em *React Native* que servirá de interface humanada.
4. Implementar um *gateway* com ESP32 e LoRa, que permita o fluxo bidirecional de informações com o *web server*.
5. Montar a unidade receptora/atuadora para microaspersão e monitoramento do solo.

3 JUSTIFICATIVA

Os sistemas de irrigação agrícolas comercializados são bastante concentrados em pivôs centrais, que possuem custo elevado para pequenas produções. Um sistema de irrigação localizada pode proporcionar um custo menor de implementação, e a não necessidade de mão de obra qualificada para instalação (automação para usuários DIY - do inglês *do it yourself*).

Outrossim, como ficou claro pelo exposto anterior, o desenvolvimento de tecnologias de automação e monitoramento remoto na agricultura tem o potencial de melhorar a eficiência operacional, reduzir a dependência da mão de obra (automação) e aumentar a competitividade dos produtores no mercado. Ao adotar sistemas de irrigação inteligentes, o produtor pode obter maior controle sobre o processo de irrigação, otimizando o uso dos recursos disponíveis e maximizando a produtividade das culturas. Portanto, investir no desenvolvimento e implementação de soluções como esta não apenas atende às demandas atuais por produção agrícola sustentável, mas também contribui para a modernização e aprimoramento do setor agrícola como um todo.

4 REFERENCIAL TEÓRICO

O desenvolvimento do trabalho exigiu a aplicação de diversos conhecimentos adquiridos ao longo do curso, como programação, incluindo programação em linguagem C/C++ e linguagens de alto nível, microcontroladores, sensores, atuadores e conversão A/D, além de conteúdos vistos através de enriquecimento, como banco de dados. Além disso, será necessário aprofundar o estudo sobre a IoT. Este capítulo apresenta uma revisão bibliográfica destes conceitos que são fundamentais para o projeto.

4.1 MÉTODOS E SISTEMAS DE IRRIGAÇÃO

Como foi abordado anteriormente, existem diferentes tipos de métodos de irrigação, com suas vantagens e desvatantagens, que podem atender ou não diferentes necessidades de aplicação. Além dos quatro métodos, faz-se necessário especificar os diferentes sistemas, e a diferença entre método e sistema. Do dicionário, observa-se que método significa modo de agir ou meio, e sistema significa o conjunto de elementos e organizados com funções específicas (EDITORA MELHORAMENTOS LTDA., 2024).

4.1.1 MÉTODOS DE IRRIGAÇÃO

Nesta seção será realizado uma revisão sobre os quatros métodos comumente utilizados na irrigação, que são:

- Aspersão: nesse método, o objetivo é aplicar a água sobre a folhagem, acima do solo, formando uma espécie de chuva artificial, através da qual a planta e o solo são umedecidos (TESTEZLAF, 2012).
- Superfície: nesse método, a água é aplicada sobre o solo com espalhamento através da ação da gravidade, como por exemplo, em um dique de irrigação, que limita e acumula a água em uma área, penetrando-a no solo, a exemplo da cultura de arroz (TESTEZLAF, 2012).
- Localizada: nesse método, a aplicação da água ocorre em uma área pré-determinada do solo, utilizando normalmente uma pequena vazão (TESTEZLAF, 2012). Como a proposta do projeto é de simplificação e baixo custo de instalação, este método será utilizado no protótipo.
- Subterrânea: nesse método, a aplicação da água ocorre sob a superfície, dentro do volume explorado pelas raízes das plantas, através, por exemplo, do sistema de gotejamento subterrâneo (instalações de linhas subterrâneas que permitem que a água atinja as raízes) (TESTEZLAF, 2012).

4.1.2 SISTEMAS

Nesta seção serão apresentados os sistemas comumente empregados no método selecionado para o projeto (irrigação localizada).

4.1.2.1 GOTEJAMENTO

O método de irrigação localizada através do sistema de gotejamento, propicia a aplicação de água (e em alguns caso de nutrientes, simultaneamente) através de um emissor de pequenas dimensões denominado gotejador (TESTEZLAF, 2012). A vazão do gotejador é pequena e, nesse sistema, a qualidade da água desempenha um papel crucial para evitar o entupimento dos gotejados, o que degrada a eficiência do sistema. Devem ser levados em consideração os fatores físicos (como suspensão de materiais sólido por litro), químicos (como pH e concentração dos sais dissolvidos por litro) e biológicos (como quantidade de bactérias por litro) (PAOLINELLI; DOURADO; MANTOVANI, 2021).

4.1.2.2 MICROASPERSÃO

O método de irrigação localizada através do sistema de microaspersão fornece água em forma de jatos ou aerosol, utilizando emissores denominados microaspersores ou *sprays* (TESTEZLAF, 2012). Nesse sistema, o problema de entupimento visto nos gotejadores é mitigado e ele pode ser instalado em tubulações aéreas, facilitando o trânsito em estufas, por exemplo. Se bem implementado, conhecendo-se a vazão, pode-se aspergir a quantidade necessária de água com precisão, fazendo um melhor uso dos recursos hídricos, através de difusores fixos ou giratórios, acionamentos manuais ou automazidados. Além das características elencadas anteriormentes, esse sistema permite o lançamento de defensivos e fertilizantes sobre as folhas com rapidez (TESTEZLAF, 2012). Neste projeto, o protótipo fará uso da microaspersão automatizada.

4.2 COMPONENTES DE SENSORIAMENTO E ATUAÇÃO

Nesta seção será realizada a abordagem dos componentes utilizados na aquisição dos sinais e posterior atuação, sendo esses: sensores, conversores, transmissores, controladores e atuadores.

4.2.1 SENSOR DE UMIDADE DO SOLO

O sensor de umidade do solo transforma a variação do nível de umidade no solo em variação de tensão realizando a aquisição analógica do nível de umidade no solo. É através dele que o sistema será capaz de determinar a ativação do mecanismo de irrigação quando estiver operando de forma automática, e também a coleta das informações que serão exibidas para o usuário.

Na Figura 2 é possível visualizar um sensor de umidade (modelo S12, com hastes resistentes à corrosão).

4.2.2 SOLENÓIDE

Quando houver determinação no sistema para atuação da irrigação, o envio de água será feito atráves de uma linha d'água controlada por uma válvula solenóide. Uma válvula solenóide é formada por dois componentes principais: um núcleo móvel (êmbolo) e seu obturador e o corpo com um orifício onde o obturador é posicionado. Esse sistema permite ou bloqueia a passagem da água dependendo da atração do núcleo móvel quando a bobina é energizada (PARKER HANNIFIN, 2002).

4.2.3 MICROCONTROLADOR

Um microcontrolador é um circuito integrado que engloba, juntamente com seus periféricos usuais, um microprocessador (CPU), todos integrados em um único *chip*. Esse componente é um circuito integrado projetado para executar tarefas específicas através de uma linguagem de comando, utilizando operações lógicas. Seu objetivo principal é executar a função armazenada em sua memória de código (memória *Flash*), para a qual possui entradas e saídas digitais e analógicas, bem como memórias RAM (*Random Access Memory*), ROM (*Read-Only Memory*), e variantes como a memória EEPROM (*Eletrically Erasable-Programmable Read-Only Memory*), além de uma unidade lógica e aritmética (ULA), cronômetros, portas de comunicação, registradores, entre outros recursos (NICOLOSI, 2004).

Esses dispositivos são programados com linguagens como *Assembly* ou C/C++ e oferecem uma gama de funcionalidades como conversão analógica/digital (A/D), PWM (*Pulse Width Modulation*), *timers* (cronômetros parametrizáveis), comunicação serial, etc.

A Figura 1 mostra um microcontrolador da família PIC (*Peripheral Interface Controller*), comumente utilizada no estudo e aprendizado desses dispositivos.

Figura 1 – Microcontrolador PIC16F877a

Fonte: Microchip (2013)

4.3 TECNOLOGIAS UTILIZADAS

Nesta seção serão abordadas as tecnologias utilizadas no desenvolvimento do *web server*, da aplicação *mobile* e dos *firmwares*.

4.3.1 LINGUAGEM C/C++

C é uma linguagem de programação estruturada de nível intermediário que permite a manipulação de bits, bytes e endereços, os quais são fundamentais para o funcionamento de computadores e microcontroladores. O código em C é muito utilizado por ser altamente portátil, facilitando a adaptação entre diferentes computadores (SCHILDT, 1996). Essa linguagem será utilizada nos nós e no *gateway* do protótipo.

O C++ introduz o paradigma de programação orientada a obejtos à linguagem C, inserindo as classes, que permitem a criação de novos tipos de dados, na qual o acesso a eles é restrito a um conjunto específico de propriedades. Essa entidade traz consigo os conceitos de atributos e métodos, para representar respectivamente, os dados membros e as funções membro daquela classe (PEREIRA, 1999). Dessa forma, há mais flexibilidade no desenvolvimento e definição de tipos.

4.3.2 MODULAÇÃO LORA

A tecnologia LoRa (abreviação de *Long Range*) é uma técnica de modulação de espectro expandido derivada de uma tecnologia conhecida como *Chirp Spread Spectrum* (CSS), desenvolvida pela Semtech. É conhecida por seu longo alcance e baixo consumo de energia, sendo amplamente utilizada em aplicações de IoT. Os chipsets LoRa da Semtech são incorporados em diferentes dispositivos e oferecem recursos para aplicações IoT, incluindo longo alcance, baixo consumo de energia e transmissão de dados segura. A tecnologia é utilizada em redes públicas, privadas ou híbridas, proporcionando maior alcance

do que as redes celulares. As implementações possibilitam integrações à infraestrutura existente e permitem aplicações IoT de baixo custo operadas por bateria (SEMTECH CORPORATION, 2024).

4.3.3 NODE.JS

Node.js é um ambiente de execução *JavaScript* (JS) de código aberto e multiplataforma, amplamente utilizado para diversos tipos de projetos. Ele utiliza a *engine JavaScript* V8, que é o núcleo do *Google Chrome*, fora do navegador, o que garante um alto desempenho ao Node.js. Como uma aplicação orientada a eventos assíncronos, o Node.js foi projetado para construir aplicativos de rede escalonáveis (OPENJS FOUNDATION, 2024). Essa tecnologia será utilizada como ambiente para o *web server*.

4.3.4 CONVERSOR A/D

O conversor analógico-digital (ADC) é um componente de sistemas de controle e processamento de sinais, responsável por converter sinais analógicos contínuos em representações digitais discretas. Esse processo é essencial para que variáveis analógicas do mundo real, como temperatura, pressão ou tensão, sejam interpretadas e processadas por dispositivos digitais. O ADC realiza a conversão em três etapas principais: amostragem, quantização e codificação. Durante a amostragem, o sinal contínuo é medido em intervalos regulares; a quantização transforma essas amostras em valores discretos; e a codificação converte esses valores em um formato binário utilizável por sistemas digitais.

A precisão de um ADC é limitada por sua resolução, definida pelo número de bits usados para representar os níveis de saída. Por exemplo, um ADC de seis bits tem 64 níveis distintos, enquanto um ADC de oito bits tem 256. Quanto maior o número de bits, menor será o degrau entre os valores adjacentes, resultando em uma conversão mais precisa do sinal analógico. Essa relação direta entre resolução e precisão exige uma escolha cuidadosa: resoluções mais altas aumentam a qualidade da conversão, mas também elevam o custo e a complexidade do circuito. Assim como ocorre nos ADCs, quanto maior a resolução maior será a proximidade entre o valor digital resultante e o sinal analógico original, influenciando diretamente a qualidade de sistemas de controle, instrumentação e comunicação. Neste projeto, será utilizado um conversor A/D embarcado no microcontrolador com resolução de 12 bits (TOCCI GREGORY L. MOSS, 2011).

4.3.5 TYPESCRIPT

O *TypeScript* foi criado pela *Microsoft* em 2010 para trazer tipos orientados a objetos ao *JavaScript*, ajudando programadores a adaptar programas para a *web*. Com o tempo, seu sistema de tipagem evoluiu para modelar o código JS nativo, resultando em um sistema poderoso com a inserção de tipagem flexível. Essa tecnologia oferece vantagens como

melhor desempenho do autocompletar na IDE, detecção precoce de erros e comunicação mais clara entre diferentes partes do programa (MICROSOFT, 2024). Será através do *JavaScript* com o auxílio do *TypeScript* que o *web server* será implementado.

4.3.6 REACT NATIVE

React Native é uma estrutura de código aberto para criar aplicativos Android e iOS utilizando React e os recursos nativos da plataforma. Com React Native, utiliza-se *JavaScript* para acessar as APIs da plataforma, para definir a aparência e o comportamento da interface do usuário usando componentes do React, que são pacotes de código reutilizável e aninhável (META, 2024). No desenvolvimento *mobile* utilizando React Native, a disposição dos elementos ocorre sobre um bloco básico de desenvolvimento denominado *View*. Um retângulo pequeno na interface do usuário, capaz de mostrar texto, imagens ou interagir com a entrada do usuário, até mesmo os elementos visuais mais simples, como uma linha de texto ou um botão, são considerados tipos de *Views*. Algumas Views podem conter outras Views. Esses pedaços de código podem ser reutilizados para gerar novos componentes com corpotamentos derivados (META, 2024).

Além da sua arquitetura baseada em componentes e *Views*, o *React Native* destaca-se por sua capacidade de ponte (*bridge*) com código nativo. Através dessa ponte, o código *JavaScript* interage diretamente com as APIs nativas do sistema operacional, permitindo acesso a funcionalidades como câmera, GPS e sensores. Essa comunicação assíncrona garante alta performance, pois a interface do usuário e a lógica do aplicativo são executadas em *threads* separadas. Outra característica importante é o *Fast Refresh*, que agiliza o desenvolvimento ao permitir a visualização instantânea de alterações no código, sem a necessidade de recompilação completa do aplicativo. Além disso, o *React Native* oferece uma vasta biblioteca de componentes e APIs pré-construídas, facilitando a criação de interfaces complexas e personalizadas (META, 2024).

4.3.7 BANCO DE DADOS

Um banco de dados (DB) é um sistema que armazena e gerencia conjuntos de dados de maneira organizada e eficiente. Ele facilita a inserção, consulta, atualização e remoção de informações de forma rápida e segura. Essa tecnologia pode ser dividida em bancos relacionais e não relacionais. Os bancos de dados relacionais foram inicialmente propostos para separar o armazenamento físico dos dados de sua representação conceitual e para fornecer uma base lógica para os bancos de dados (ELMASRI; NAVATHE, 2004). Por sua vez, banco não relacionais, também chamados NoSQL, surgiram para solucionar algumas demandas envolvendo exigências por maior produtividade, menores tempos e complexidade, entre outros. Para enfrentar alguns desses desafios, bancos de dados não relacionais foram rapidamente incorporados ao longo dos últimos anos. Contudo, muitos desses sistemas

são apenas soluções temporárias, fornecendo um conjunto limitado de recursos específicos (MONGODB, INC, 2021).

Em 2009, o banco de dados não relacional MongoDB foi introduzido no mercado como uma nova categoria de aplicação e, em pouco tempo, se tornou um das aplicaçãoes NoSQL mais populares entre os desenvolvedores. O MongoDB preserva os melhores aspectos dos bancos de dados relacionais e NoSQL, ao mesmo tempo em que oferece uma base tecnológica que capacita organizações a atender às demandas das aplicações modernas (MONGODB, INC, 2021). Será com o MongoDB que os dados de umidade captados pelo sensor serão armazenados

4.4 TRABALHOS RELACIONADOS

O objetivo desta seção é apresentar pesquisas e desenvolvimentos que se conectam com o presente trabalho, destacando, com brevidade, seus resultados e como se relacionam com as abordagens aqui exploradas. Por isso, é importante ressaltar que não se trata de uma revisão exaustiva da literatura sobre o tema. Dada a vastidão de pesquisas e projetos existentes, optou-se por focar em dois trabalhos que apresentam maior proximidade com os conceitos e metodologias discutidos neste artigo, oferecendo um contexto relevante do tema.

Diversos trabalhos exploram o uso de tecnologias IoT no contexto agropecuário, buscando soluções para otimizar processos e reduzir custos. Germano (2022) desenvolveu um protótipo de estação meteorológica utilizando a tecnologia LoRa para coleta de dados em plantios. A escolha do LoRa se justifica pela sua capacidade de cobrir extensas áreas com baixo custo, quando comparado a soluções que demandam maquinário e infraestrutura complexos. O protótipo, alimentado por bateria e com comunicação sem fio LoRaWAN, visa estudar o consumo energético e a área de cobertura da tecnologia. Os testes demonstraram resultados promissores quanto ao baixo consumo de energia, característica da transmissão LoRa. Em relação à distância de comunicação, o sistema obteve alcance de aproximadamente 4 quilômetros, considerando um nível mediano de interferências de obstáculos. Este trabalho demonstra o potencial do LoRa para aplicações agrícolas que necessitam de longo alcance e baixo consumo energético (GERMANO, 2022). Em relação à automação de sistemas de irrigação, Lima (2019) também explorou o uso de IoT para desenvolvimento de um protótipo, mostrando a relevância da temática no contexto do agronegócio. Ela utilizou o NodeMCU ESP8266 como microcontrolador, um servidor baseado no Adafruit IO e uma frequência de transmissão de 433 MHz. O protótitpo obteve, com uso de uma antena dipolo, alcance de 6,5 km (LIMA, 2019).

5 MATERIAS E MÉTODOS

Nessa seção aborda os componentes que foram utilizados na construção do sistema, descrevendo suas especificações e funcionalidades dentro do projeto, bem como a metodologia empregada.

5.1 DISPOSITIVOS E COMPONENTES

Nesta seção, detalham-se os métodos e metodologias empregados no desenvolvimento deste projeto, fornecendo-se uma visão do processo de pesquisa e implementação. Inicia-se descrevendo os componentes e ferramentas selecionados, justificando-se as escolhas com base em suas funcionalidades e adequação aos objetivos do projeto. Conjuntamente, exploram-se as especificidades de cada componente, detalhando-se como foram configurados e integrados para alcançar os resultados desejados. Abordam-se também as técnicas de coleta e análise de dados, quando aplicáveis, e os procedimentos de validação e teste utilizados para garantir a qualidade e confiabilidade dos resultados, explicitando a abordagem metodológica adotada e permitindo a reprodução do trabalho.

5.1.1 SENSOR DE UMIDADE

Para o sensoriamento da umidade no solo foi utilizado o sensor de umidade S12. Este sensor é resistente à corrosão através da cobertura de suas hastes e apresenta saída de tensão limitada à tensão de alimentação (até 12 V, com limite mínimo de 3,3 V) em nível alto, quando estiver seco, e 0 V em nível baixo, quando umedecido. Junto do sensor, um módulo de interface com saída analógica e digital complementa o conjunto facilitando os acoplamentos com outras placas. Esse produto possui as seguintes características:

- Tensão de operação: 3,3 a 12 VDC;
- Corrente: inferior a 20 mA;
- Dimensões sonda: 60 x 19 x 9 mm (C x L x A);
- Dimensões módulo: 36 x 15 x 7 mm (C x L x A);
- Comprimento do cabo de sonda: 1 m;
- Peso: 25 g;

A Figura 2 mostra o sensor de umidade e seu módulo. Uma das caracteríistias principais do sensor é a haste inoxidável que permite o uso prolongado no protótipo.

Figura 2 – Sensor S12

Fonte: Fermac - Robótica (2024)

5.1.2 ASPERSOR TRAMONTINA

Para a realização da microaspersão da água foi utilizado o aspersor giratório Tramontina Pro, fabricado pela Tramontina. Essa classe de aspersores possui compatibilidade com 3 diferentes diâmetros de mangueiras (1/2", 5/8"e 3/4"), e é capaz de trabalhar com diferentes raios de alcance, possui conexão do tipo engate rápido, alcance de 14,8 metros, área molhada de 688 m^2 e pode ser regulado horizontalmente de 30° a 360°. Esse produto possui as seguintes características:

- Diâmetro molhado: 29,6 m;
- Pressão nominal: 4 bar;
- Medidas: 300 x 125 x 50 mm;
- Peso: 106 g;

A Figura 3 mostra o microasperssor montado com a haste.

Figura 3 – Aspersor Tramontina Pro

Fonte: Ferramac (2025)

5.1.3 VÁLVULA SOLENOIDE 12 VDC

Uma válvula solenoide 12 VDC 1/2 polegada normalmente fechada foi utilizada no projeto, bloqueando a passagem de água em caso de falta de energia ou com o sistema em *idle*. Quando o servidor determinar a necessidade de água, a válvula solenoide é ativada, permitindo a passagem da água através de uma linha d'água controlada. Este dispositivo de controle assegura que a irrigação seja realizada de maneira eficiente, conectando-se facilmente à rede de água através de roscas de meia polegada. Esse produto possui as seguintes características:

- Material: plástico e metal;
- Tensão: 12 V;
- Potência nominal: 5 W;
- Modelo de operação: normalmente fechado;
- Pressão: 0.02 a 0.8 Mpa;
- Tamanho da rosca: 1/2";
- Temperatura do fluido: 0 a 100 ºC;

A Figura 4 mostra a válvula solenoide que será utilizada no projeto.

Figura 4 – Válvula Solenoide 12 VDC 1/2”

Fonte: Fermac - Robótica (2024)

5.1.4 TRANSCEPTOR LORA

A comunicação entre o módulo nó da rede (sensor ou atuador) com o *gateway* é baseada em LoRa, utilizando a frequência de 915 MHz, com o módulo SX1262. O SX1262 é um transceptor LoRa da Semtech, projetado para aplicações de longo alcance e baixo consumo de energia. Similar ao SX1276 e SX1268, ele oferece comunicação robusta e resistente a interferências, ideal para aplicações IoT. Utilizando a modulação LoRa, o SX1262 proporciona alta sensibilidade e potência de transmissão, com opções de bit rate programáveis. Assim como seus predecessores, o SX1262 demonstra superioridade em relação a técnicas convencionais em bloqueio e seletividade, otimizando o equilíbrio entre alcance, imunidade a interferências e consumo de energia (SEMTECH CORPORATION, 2021).

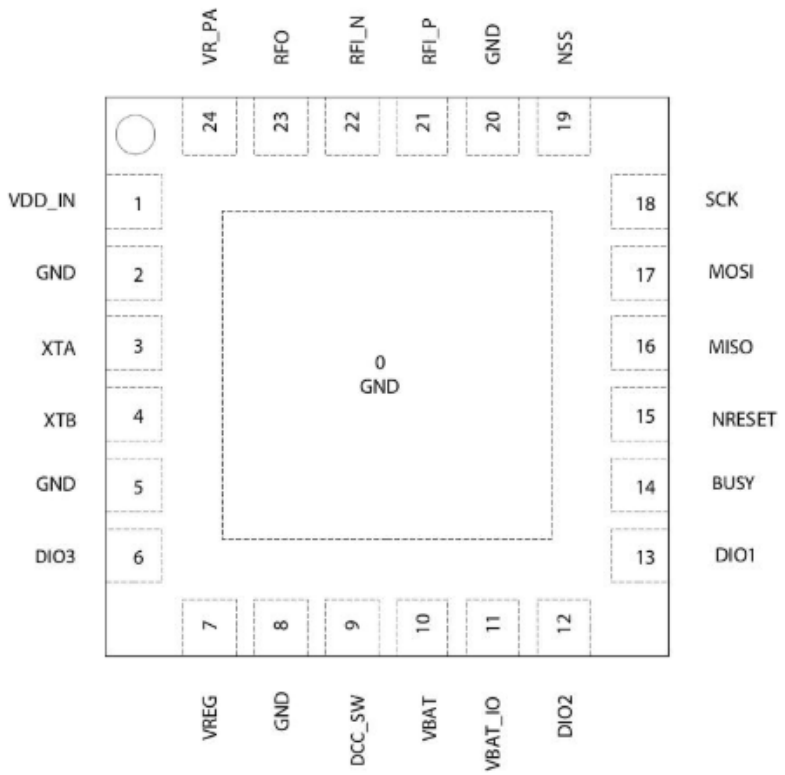
O *transceiver* SX1262, utilizado na comunicação LoRa, opera em modo *half-duplex*. Isso significa que, embora permita a comunicação bidirecional, a transmissão e a recepção de dados não ocorrem simultaneamente. Em um dado momento, o dispositivo pode estar transmitindo ou recebendo, mas não ambos. Essa característica impõe a necessidade de gerenciar a alternância entre os modos de transmissão e recepção, evitando colisões de dados e garantindo a integridade da comunicação. A implementação de mecanismos de controle de fluxo e confirmação de entrega é crucial para mitigar as limitações inerentes ao *half-duplex* e assegurar a confiabilidade da transmissão.

Para fins de prototipagem, foi utilizado uma placa de desenvolvimendo da Heltec Automation no projeto, modelo Heltec ESP32 LoRa V3, a qual possui um chipset SX1262 embarcado.

- Tensão de opração: 3,3 V;
- Corrente stand-by: tipicamente 0,8 mA;
- Corrente *receiver*: tipicamente 8,8 mA;
- Corrente *transmitter*: tipicamente 25,5 mA;
- Frequência de transmissão: 915 MHz;
- Frequência do oscilador de cristal: 32 MHz;
- Taxa de erro de pacote (PER): 1%;
- Dimensões: 4,0 x 4,0 mm (C x L);

A Figura 5 mostra a pinagem do módulo SX1262 que será utilizado no *gateway* e nos nós do protótipo.

Figura 5 – Pinagem SX1262



Fonte: Semtech (2021)

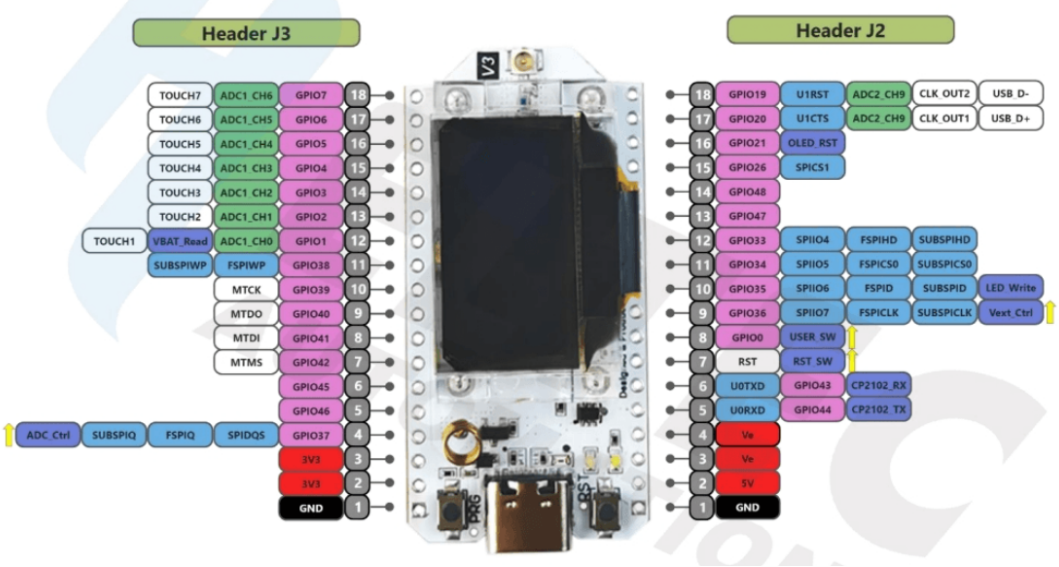
5.1.5 ESP32

O ESP32 é um microcontrolador lançado pela Espressif, possui suporte para conexões Wi-Fi e Bluetooth, sendo assim um excelente dispositivo para projetos de IoT (ESPRESSIF SYSTEMS, 2017). O *gateway* e nó foram desenvolvidos através da integração de uma unidade transceptora LoRa com o microcontrolador ESP32. Para fins de prototipagem o módulo Heltec ESP32 LoRa V3, um *devkit* desse microcontrolador, foi empregado. Por padrão, esse módulo possui algumas características relevantes para o desenvolvimento do protótipo:

- Wi-Fi com faixa de frequência de 2.4 GHz, amplamente utilizada nos dias de hoje;
- Transmissão *wireless* nos padrões: IEEE 802.11b, IEEE 802.11g e IEEE 802.11n (com *bit rate* de 150 Mbps no padrão 802.11n);
- Tensão de alimentação: 5 V;
- Tensão lógica: 3,3 V;
- Corrente de consumo típica: 80 mA;
- Corrente de consumo máxima: 500 mA;
- Dimensões sonda: 50,2 x 25,5 x 9,74 mm (C x L x A);
- Clock: 80 a 240 MHz (ajustável);
- GPIOs: 29;

A Figura 6 mostra a pinagem do Heltec ESP32 LoRa V3.

Figura 6 – Pinagem *DevKit* ESP32 LoRa V3



Fonte: Heltec Automation (2023)

5.1.6 WEB SERVER

O desenvolvimento de um *web server* em Node.js utilizando TypeScript representa uma abordagem mais moderna e eficiente para a criação de sistemas escaláveis. Ao combinar a facilidade de uso e a velocidade de execução do Node.js com a segurança e a tipagem estática fornecidas pelo TypeScript, uma API REST foi implementada de forma mais estruturada e com menos propensão a erros. Além disso, ele dispôs de conexão com o banco de dados, possibilitando uma integração para armazenar e manipular dados de forma eficiente.

A adoção de um *design pattern* para a aplicação *back-end* é importante para estabelecer um padrão de projeto para o desenvolvimento da aplicação. Nesse caso, o *publish-subscribe* (pub/sub) proporciona uma arquitetura flexível e escalável para lidar com eventos e mensagens entre diferentes partes do sistema. Com o pub/sub, os componentes do servidor podem se comunicar de forma assíncrona, permitindo uma maior modularidade e desacoplamento entre os diferentes módulos da aplicação. Isso facilita a manutenção e a evolução do código ao longo do tempo, tornando-o mais adaptável a mudanças e requisitos futuros. Em suma, o uso conjunto de Node.js, TypeScript, API REST, conexão com banco de dados e o *design pattern* pub/sub oferece uma base sólida para o desenvolvimento de uma aplicação *web* escalável.

O uso do ODM (*Object Document Mapping*) Mongoose no desenvolvimento do servidor *web* supre a necessidade de uma maneira eficiente e simplificada de interagir com o banco de dados MongoDB. Com o Mongoose, é possível definir modelos de dados com estruturas claras e intuitivas, mapeando objetos JavaScript para documentos MongoDB de forma transparente. Além disso, o Mongoose fornece uma ampla gama de funcionalidades, como

validação de dados, gatilhos de pré e pós-save e associações entre documentos, facilitando o desenvolvimento de uma aplicação robusta e escalável. Com sua sintaxe expressiva e poderosa, o Mongoose torna o acesso e a manipulação de dados no MongoDB uma tarefa simples e elegante, contribuindo significativamente para a eficiência e a qualidade do *web server* desenvolvido (MONGOOSE, 2024).

Além da abstração que simplifica a interação com o MongoDB, o Mongoose destaca-se pela sua alta configurabilidade através da definição de *schemas*. Esses *schemas* permitem a especificação precisa da estrutura dos documentos, incluindo a definição de tipos de dados para cada campo (como String, Number, Date, Boolean, etc.), a aplicação de validações customizadas (como campos obrigatórios, valores mínimos e máximos, expressões regulares), a definição de valores padrão e a criação de relacionamentos entre diferentes coleções. Essa flexibilidade na configuração dos *schemas* garante a integridade e a consistência dos dados armazenados, permitindo que os desenvolvedores adaptem o modelo de dados às necessidades específicas da aplicação e previnam a inserção de dados inválidos ou inconsistentes. A simplicidade na definição desses schemas e a vasta gama de opções de configuração tornam o Mongoose uma ferramenta poderosa e adaptável para o desenvolvimento de aplicações *web* que utilizam o MongoDB.

A Listagem 1 apresenta um *schema* denominado MoistureSchema que define a estrutura dos documentos que armazenam as leituras de umidade no banco de dados MongoDB. A *interface* IMoisture define os campos que compõem cada documento: sensorId (o identificador do sensor que realizou a leitura, referenciando a coleção "Sensor"), *value* (o valor da umidade) e *timestamp* (o momento da leitura). No *schema*, cada campo é configurado com um tipo de dado específico (*type*: String, *type*: Number, *type*: Date) e a opção "*required: true*" garante que os campos sensorId e *value* sejam obrigatórios. O campo *timestamp* possui um valor padrão ("*default*: Date.now"), registrando automaticamente a data e hora da criação do documento caso nenhum valor seja fornecido. A opção "*timestamps: true*" adiciona automaticamente os campos createdAt e updatedAt, registrando os *timestamps* de criação e atualização do documento. Por fim, a configuração MoistureSchema.set("toJSON", ...) customiza a conversão do documento para JSON, incluindo *virtuals* (campos virtuais) como o id, por exemplo, e excluindo o campo _id da resposta JSON, tornando a resposta da API mais limpa e concisa. Em resumo, este *schema* define uma estrutura simples e configurável para os dados de umidade, garantindo a integridade e a consistência das informações armazenadas no banco de dados.

Listagem 1 – Criação e exportação do *schema* do dado de umidade

```
import { Schema, Document, model } from "mongoose";
interface IMoisture extends Document {
  sensorId: string;
  value: number;
  timestamp: Date;
}
const MoistureSchema = new Schema<IMoisture>(
  {
    sensorId: {
      type: String,
      ref: "Sensor",
      required: true,
    },
    value: {
      type: Number,
      required: true,
    },
    timestamp: {
      type: Date,
      default: Date.now,
    },
  },
  {
    timestamps: true,
  }
);
MoistureSchema.set("toJSON", {
  virtuals: true,
  versionKey: false,
  transform: (_, ret) => {
    delete ret._id;
  },
});
export const Moisture = model("Moisture", MoistureSchema);
```

Fonte: Autoria própria (2025)

5.1.7 GATEWAY

O desenvolvimento de um *gateway* com capacidades Wi-Fi e LoRa representa uma solução altamente versátil e eficiente para integrar dispositivos IoT em uma rede. Ao utilizar Wi-Fi para se conectar a um servidor central por meio de uma API, o *gateway* pode transmitir e receber dados de forma rápida e confiável, permitindo uma comunicação bidirecional entre os dispositivos de IoT e a infraestrutura de back-end. Além disso, ao incorporar tecnologia LoRa, o *gateway* pode estender sua conectividade para dispositivos

remotos de sensoriamento e atuação, mesmo em áreas com cobertura limitada ou em ambientes onde o consumo de energia é uma preocupação.

A combinação de Wi-Fi e LoRa no *gateway* proporciona uma solução robusta para conectar e gerenciar uma ampla gama de dispositivos IoT. Enquanto o Wi-Fi oferece alta velocidade e conectividade de curto alcance, ideal para interação com o servidor central e dispositivos próximos, o LoRa estende a cobertura para dispositivos remotos, possibilitando comunicações de longo alcance com baixo consumo de energia. Isso permite que o *gateway* atue como uma ponte entre o mundo físico dos sensores e atuadores e o mundo digital da computação em nuvem, oferecendo uma plataforma centralizada para coletar, processar e analisar dados em tempo real.

5.2 METODOLOGIA

A metodologia é a descrição dos processos que foram desenvolvidos para implementação do protótipo. Para o projeto, foi desenvolvido a funcionalidade de sensoriamento, a implementação do *gateway*, o desenvolvimento do aplicativo e do *web server*, bem como a integração entre cada um desses processos.

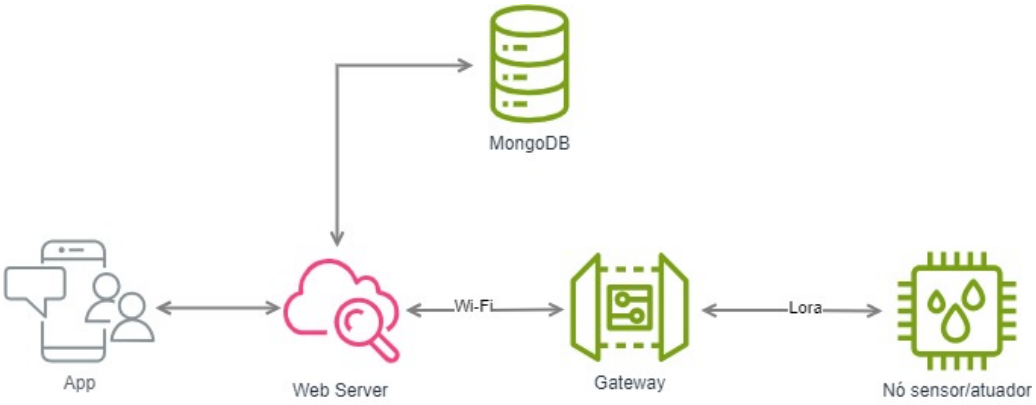
5.2.1 VISÃO GERAL

Para implementação do protótipo foram utilizados os seguintes componentes:

- Heltec ESP32 LoRa V3.
- *Protoboard*;
- Resistores;
- Capacitores;
- LED's;
- Cabos de conexão (*jumpers*);
- Fonte chaveada de alimentação;
- Serviço de hospedagem de aplicações *Cloud*;
- Placa de testes;
- Sensor de umidade S12;
- Solenoide;
- Relé 3V (para acionamento do solenoide);
- Microaspersor;

A Figura 7 exibe o diagrama de blocos que demostra a diposição e como funcionará o sistema.

Figura 7 – Diagrama de blocos do sistema



Fonte: Autoria própria (2024)

5.2.2 PROJETO DO GATEWAY

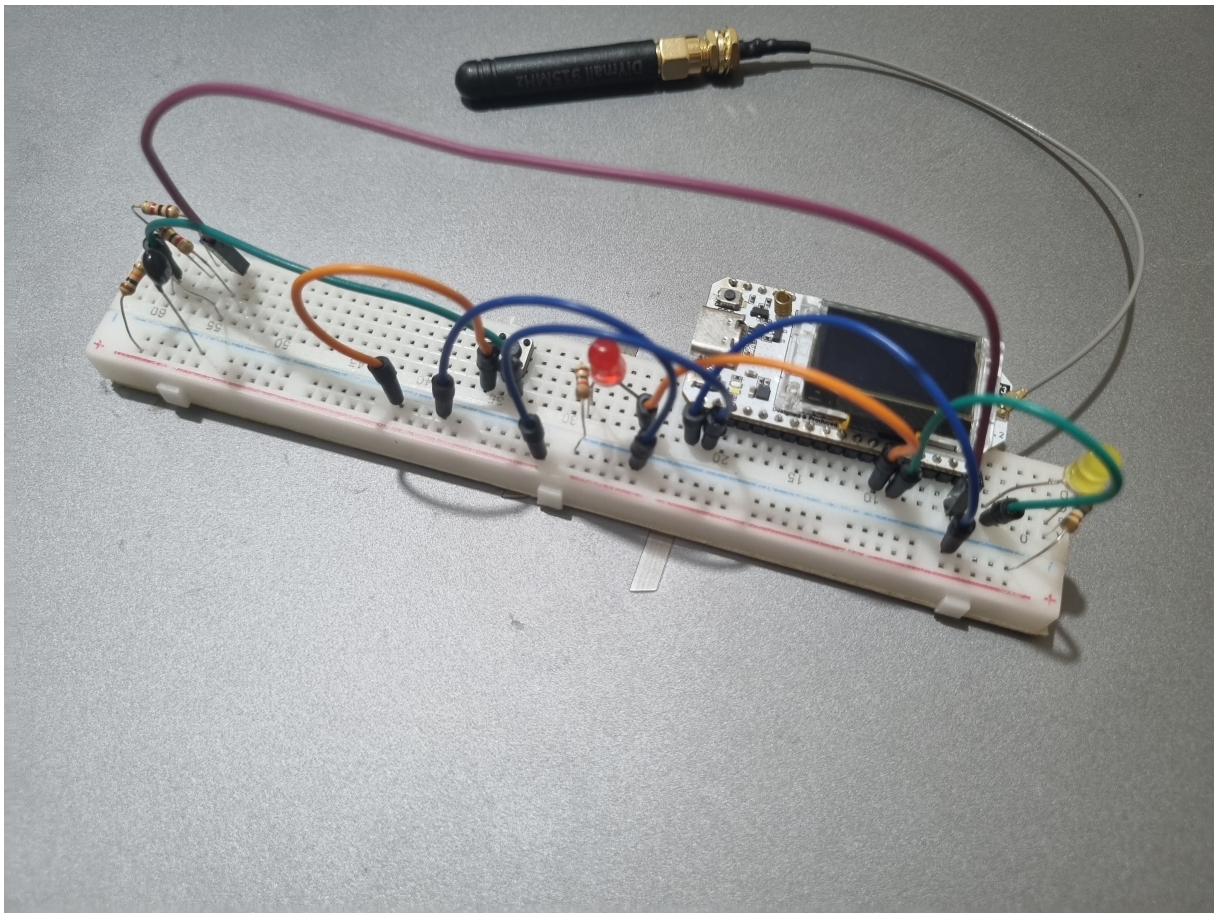
O *gateway* para o sistema de irrigação foi desenvolvido com o Heltec ESP32 LoRa V3, que contém o ESP32 e o transceptor SX1262, utilizando a IDE VSCode com PlatformIO para desenvolvimento de sistemas embarcados. Essa configuração permitiu a comunicação LoRa com sensores de umidade posicionados no campo, viabilizando a coleta de dados a longas distâncias e com baixo consumo energético, características cruciais para aplicações IoT em ambientes agrícolas. O *gateway* atuou como intermediário, recebendo, processando e encaminhando os dados dos sensores, além de transmitir comandos para os atuadores via LoRa

A metodologia de desenvolvimento seguiu uma abordagem iterativa, começando com a implementação básica da comunicação LoRa, seguida pela integração dos sensores e atuadores. Testes com periféricos e o monitor serial foram realizados para analisar a comunicação entre o *gateway* e os nós sensores/atuadores. A partir dos resultados dos testes, ajustes foram feitos para otimizar o desempenho e garantir a precisão dos dados recebidos e transmitidos pelo *gateway*.

Durante o desenvolvimento, algumas dificuldades foram encontradas, principalmente relacionadas ao processamento simultâneo de tarefas. Processos blocantes, como as chamadas HTTP com a biblioteca WiFi.h, causaram o bloqueio de outras tarefas no processador, como o processamento de dados LoRa. Isso ocorreu porque a biblioteca WiFi.h, em suas operações de conexão e transmissão, executava operações síncronas, ou seja, o processador aguardava a conclusão da operação antes de prosseguir para a próxima tarefa. Além disso, a natureza *half-duplex* da comunicação LoRa (onde a transmissão e a recepção não podem ocorrer simultaneamente) apresentou um desafio. Em cenários onde o *gateway*

estava transmitindo comandos para os atuadores, a recepção de dados dos sensores poderia ser perdida, impactando a integridade dos dados coletados. Essas dificuldades foram consideradas durante o desenvolvimento e estratégias foram implementadas para minimizar seus impactos no sistema. Para isso, a lógica de execução do LoRa foi encapsulada em uma classe para gerenciar a instância do rádio que, a cada execução do loop(), verifica se há alguma mensagem sendo transmitida. Caso não haja, imediatamente troca-se o papel do rádio para recebedor. A Figura 8 apresenta a placa de desenvolvimento conectada a sua antena em uma protoboard com LEDs e um botão.

Figura 8 – Heltec ESP32 LoRa V3 atuando como controlador *gateway*



Fonte: Autoria própria (2025)

5.2.2.1 CONFIGURAÇÃO DO GATEWAY PELO USUÁRIO

A configuração do *gateway* para acesso à internet é realizada priorizando a usabilidade do usuário. Quando o *gateway* é iniciado e não possui credenciais de conexão Wi-Fi armazenadas em sua memória não volátil NVS (Non-Volatile Storage), ele entra no modo de operação como servidor. Nesse estado, seu ESP32 cria um ponto de acesso Wi-Fi com o nome "Gateway_Acesso"e disponibiliza uma página de configuração acessível pelo endereço fixo http://192.168.4.1. Ao conectar um dispositivo à rede "Gateway_Acesso", o

usuário pode visualizar todas as redes Wi-Fi disponíveis ao alcance do *gateway*, selecionar uma delas e inserir a senha correspondente. Esse processo permite que as credenciais sejam transmitidas ao *gateway* de forma segura, possibilitando sua conexão à internet. A Figura 9 apresenta a tela de configuração para o usuário disponibilizada no endereço http://192.168.4.1.

Além disso, a configuração pode ser simplificada pelo uso do aplicativo do sistema de irrigação, que oferece uma interface mais amigável para o usuário. Após conectar o dispositivo à rede "Gateway_Acesso"e clicar no botão "Cadastrar Gateway"na tela inicial, selecionar uma rede Wi-Fi disponível e fonecer as credenciais, as informações serão enviadas ao *gateway*. Caso sejam válidas e a conexão seja bem-sucedida, o ESP32 armazena os dados de acesso na memória NVS utilizando a biblioteca Preferences do ESP32. Após esse processo, o dispositivo é reiniciado e, em inicializações futuras, utilizará as credenciais salvas para conectar-se automaticamente à internet, garantindo um funcionamento contínuo e sem necessidade de reconfiguração.

Figura 9 – Página HTML gerada pelo ESP32 para conexão ao Wi-Fi

Fonte: Autoria própria (2025)

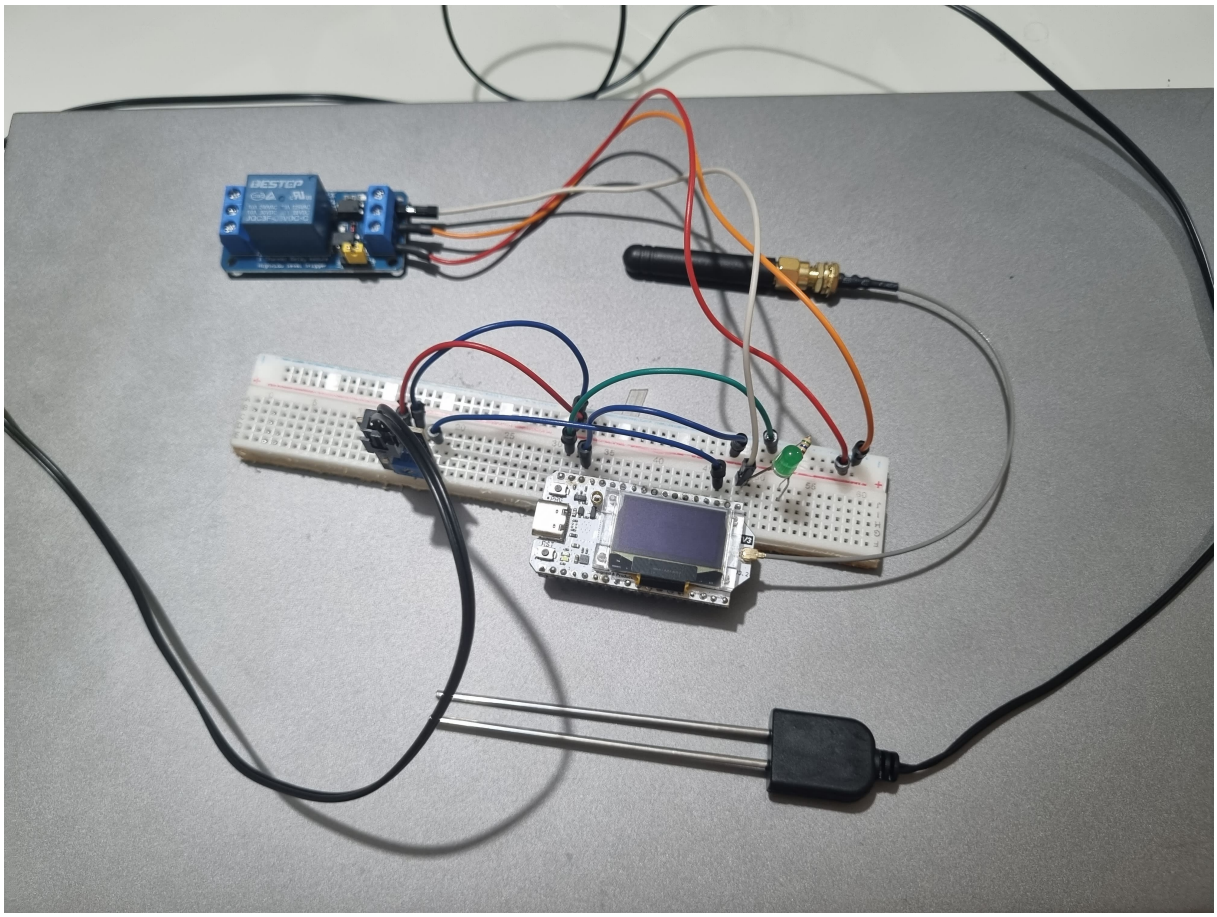
5.2.3 PROJETO DO NÓ DE SENSORIAMENTO E ATUAÇÃO

A metodologia de desenvolvimento do nó sensor/atuador foi centrada na plataforma Heltec ESP32 LoRa V3, utilizando a linguagem C++ e o ambiente de desenvolvimento Arduino IDE, devido à sua vasta documentação, bibliotecas disponíveis e facilidade de programação para microcontroladores. O desenvolvimento seguiu uma abordagem iterativa, iniciando com a configuração do ambiente e a implementação da comunicação LoRa com o *gateway*. Em seguida, foi realizada a integração do sensor de umidade S12, com a implementação da leitura analógica e a conversão para valores digitais. O controle do atuador (relé e solenoide) foi implementado posteriormente, com a definição dos pinos

de controle e a lógica de acionamento baseada nos comandos recebidos via LoRa. Testes foram realizados em cada etapa para garantir o funcionamento correto dos componentes individualmente, antes da integração completa do sistema.

O desenvolvimento do *firmware* do nó sensor/atuador também considerou a gestão eficiente dos recursos do ESP32, principalmente em relação ao processamento simultâneo de tarefas, conforme será apresentado em seção dedicada a esse tópico. A leitura do sensor de umidade, a comunicação LoRa e o controle do atuador foram implementados de forma a minimizar o consumo de energia, prolongando a vida útil da bateria em futuras implementações portáteis. A Figura 10 apresenta a placa de desenvolvimento junto com LED e relé para acionamento do solenoide. A Figura 11 apresenta o solenoide com uma mangueira e um adaptador do tipo engate rápido para permitir a fácil instalação no ambiente pré-existente. A Figura 12 apresenta as ligações elétricas do microcontrolador do nó.

Figura 10 – Heltec ESP32 LoRa V3 atuando como controlador do nó

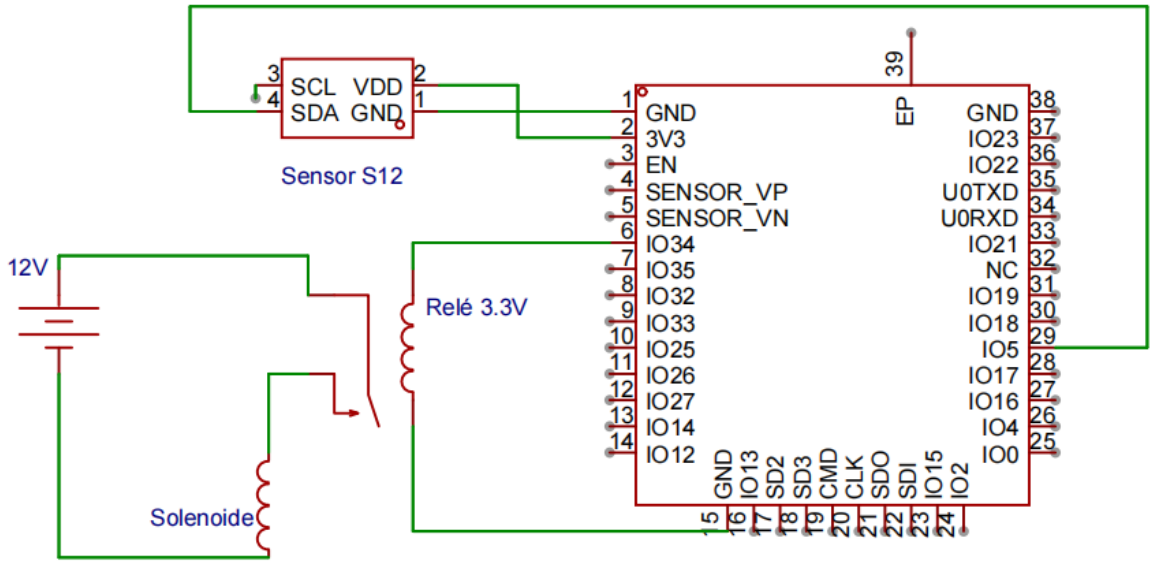


Fonte: Autoria própria (2025)

Figura 11 – Solenoide e adaptador do tipo engate rápido prontos para instalação

Fonte: Autoria própria (2025)

Figura 12 – Ligação elétrica do microcontrolador do nó



Fonte: Autoria própria (2025)

5.2.4 PROJETO DO FIRMWARE

Nessa seção, serão apresentados tópicos relativos ao desenvolvimento do *firmware* embarcado no *gateway* e na unidade sensora/atuadora.

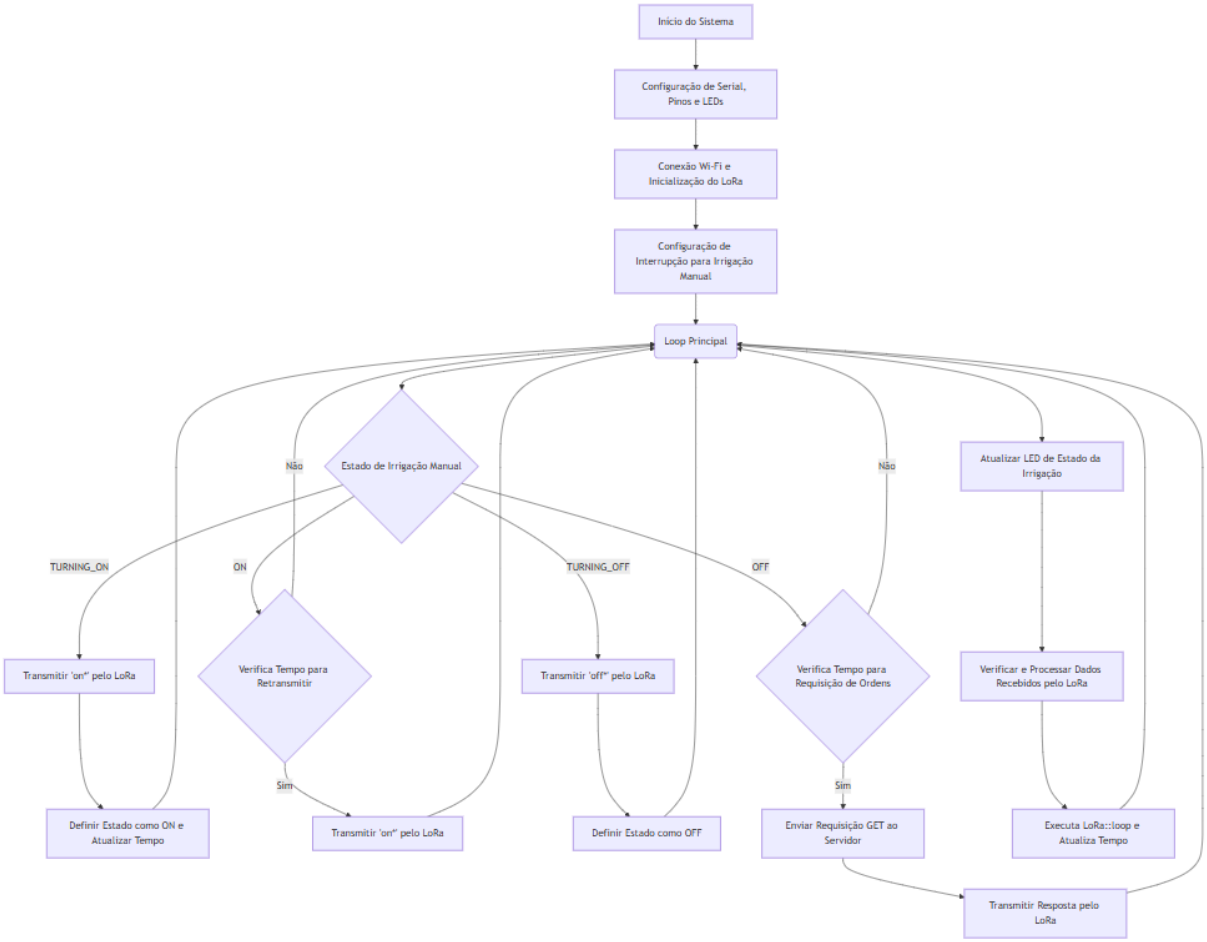
5.2.4.1 FIRMWARE DO GATEWAY

O *firmware* implementado no ESP32 tem como principal objetivo a comunicação bidirecional entre um servidor central e uma rede de dispositivos LoRa. Ele atua como uma ponte, recebendo dados dos dispositivos LoRa e encaminhando-os para o servidor via Wi-Fi, e vice-versa. A arquitetura do *firmware* se baseia em um loop principal que gerencia as comunicações LoRa e Wi-Fi, além de um sistema de interrupção para o acionamento manual da irrigação. A comunicação LoRa é gerenciada pela classe LoRa, que encapsula a biblioteca RadioLib e utiliza o transceptor SX1262. A inicialização do rádio LoRa, configurando os pinos de Chip Select (LORA_CS_PIN), Reset (LORA_RESET_PIN), DIO1 (LORA_DIO1_PIN) e Busy (LORA_BUSY_PIN), ocorre na função bootstrap. A recepção de pacotes LoRa é assíncrona, utilizando uma interrupção acionada pelo pino DIO1, cujo tratamento é realizado pela função interruptionHandler. Esta função seta uma *flag* (receivedPacketFlag) que é verificado no loop principal. Caso a *flag* esteja setado, o pacote é lido usando getPacket() e a *flag* é resetado por resetReceivedFlag(). O envio de mensagens LoRa é feito pela função transmit, que prepara a mensagem para transmissão com prepareToTransmit().

A conexão Wi-Fi com a rede local é gerenciada pela função wifiBootstrapper, presente em WiFiConn.cpp. Esta função recebe o SSID e a senha da rede Wi-Fi como parâmetros e tenta estabelecer a conexão utilizando a biblioteca Wi-Fi do ESP32. Um mecanismo de tentativas com *timeout* é implementado para evitar que o sistema fique travado em caso de falha na conexão. A função exibe mensagens de log informando o status da conexão, incluindo o endereço IP (Internet Protocol) obtido após a conexão bem-sucedida. A comunicação com o servidor é feita através de requisições HTTP GET e POST. Para o registro de eventos e depuração, foi implementada uma biblioteca de logs local (log.h e .cpp). Esta biblioteca utiliza a biblioteca time.h para gerar logs com data e hora local, permitindo o registro preciso dos eventos do sistema. Esses logs poderiam ser coletados por um sistema de visualização de dados e logs em tempo real, facilitando o monitoramento e a análise do comportamento do *gateway*. Uma luz de serviço, conectada ao pino LIGHT_SYSTEM_ON_PIN, indica visualmente que o *gateway* foi iniciado e está operacional, fornecendo um *feedback* imediato sobre o status do dispositivo.

O *firmware* implementa um modo de ação manual para o sistema de irrigação, controlado por uma máquina de estados com os seguintes estados: TURNING_ON, ON, TURNING_OFF e OFF. O acionamento manual é feito através de uma interrupção no pino MANUAL_IRRIGATION_PIN. Quando o botão conectado a este pino é pressio-

Figura 13 – Fluxograma do *firmware* do *gateway*



Fonte: Autoria própria (2024)

nado (transição para LOW), o estado da máquina de estados é alternado entre ON e OFF. Durante a transição para ON (TURNING_ON), o comando "on*"é transmitido via LoRa. Enquanto o estado for ON, o comando “on*” é transmitido periodicamente. A transição para OFF (TURNING_OFF) envia o comando “off*”. Uma luz de indicação do modo de operação manual, conectada ao pino LIGHT_MANUAL_IRRIGATION_ON_PIN, acende quando o sistema está nos estados ON ou TURNING_ON, indicando visualmente que a irrigação manual está ativa. O código fonte do *firmware* está disponível em um repositório público no GitHub[1]. A Tabela 1 apresenta um exemplo alguns comandos possíveis, enviados via LoRa, e definidos para o acionamento ou desligamento da irrigação em nós específicos ou de todos os nós conectados ao sistema.

5.2.4.2 FIRMWARE DO NÓ SENSOR/ATUADOR

O *firmware* implementado no ESP32 do nó sensor/atuador tem como principal objetivo coletar dados de umidade do solo e controlar um sistema de irrigação com base em comandos recebidos de um *gateway* LoRa. A arquitetura do *firmware* é baseada em

[1] https://github.com/mbssilva/irrigation-backend

Tabela 1 – Ordens via LoRa para acionamento e desligamento da irrigação

Comando	Ordem de	Nós impactados
`"on1,2,3,7"`	acionamento	1, 2, 3 e 7
`"off1,8"`	desligamento	1 e 8
`"on*"`	acionamento	Todos
`"off*"`	desligamento	Todos

um loop principal que gerencia a leitura do sensor de umidade, a comunicação LoRa e o controle do atuador de irrigação. A leitura do sensor de umidade, conectado ao pino analógico MOISTURE_READER_ADC, é realizada periodicamente pela função handleReadMoistureAndTransmit, que converte a leitura analógica em um valor percentual e formata uma mensagem contendo o ID do sensor e o nível de umidade em formato CSV (comma-separated values), utilizando a função formatStringsCSV da biblioteca local utils. Esta mensagem é então transmitida para o *gateway* via LoRa.

A comunicação LoRa é gerenciada pela mesma classe LoRa utilizada no *gateway*, encapsulando a biblioteca RadioLib e utilizando o transceptor SX1262. A inicialização do rádio LoRa ocorre na função bootstrap, configurando os pinos necessários. A recepção de comandos do *gateway* é tratada pela função handleReceivedGatewayData, que verifica a presença de pacotes recebidos (getReceivedPacketFlag()) e, em caso positivo, extrai o comando utilizando a função commandSensor da biblioteca utils. Esta função analisa a string recebida, buscando comandos "on" ou "off" direcionados ao ID do nó (ou a todos os nós da rede através da notação "*"). O resultado desta análise define o estado da irrigação.

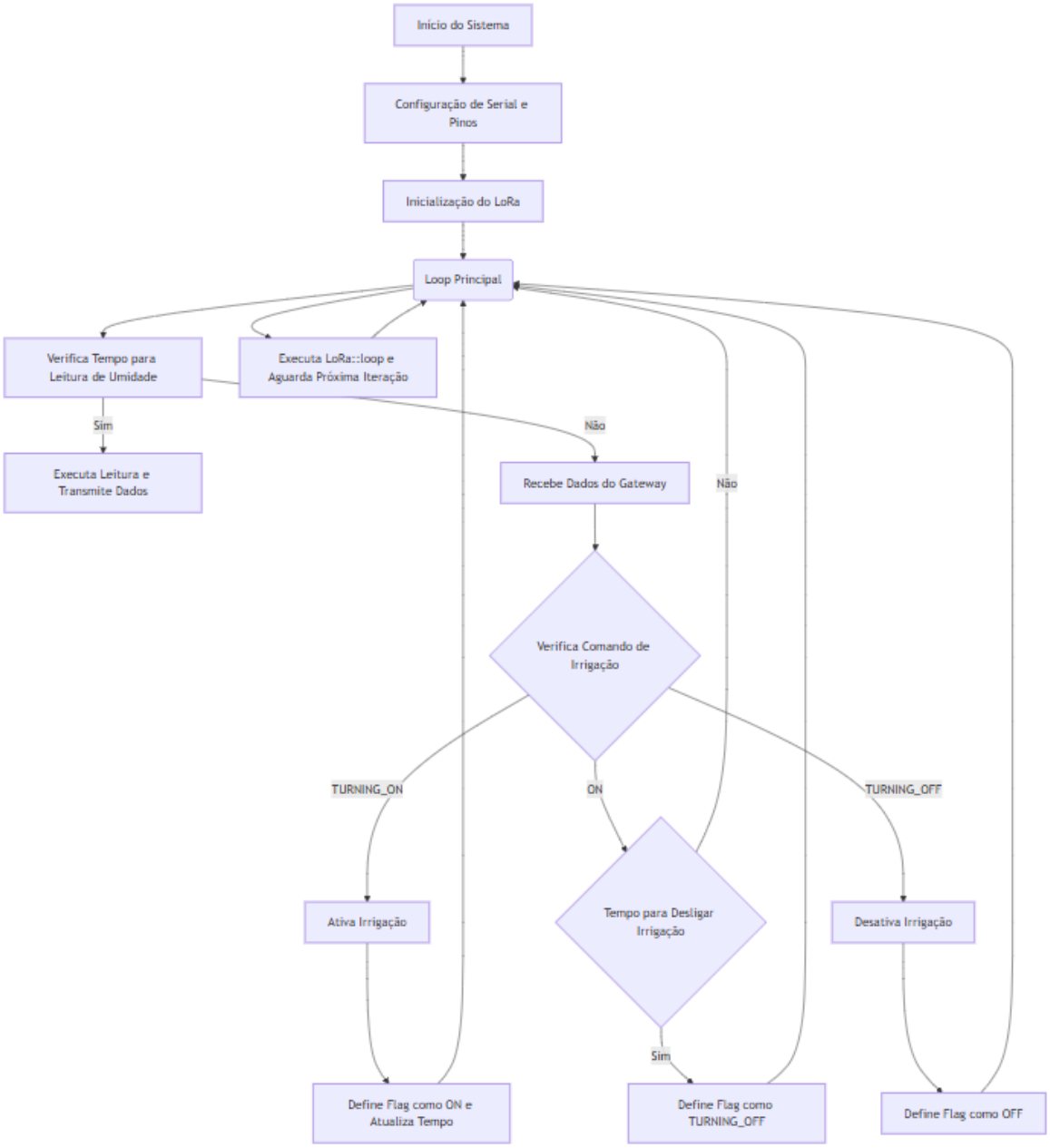
O controle do atuador de irrigação é realizado por uma máquina de estados com os seguintes estados: TURNING_ON, ON, TURNING_OFF e OFF, conectado ao pino digital IRRIGATION_ACTUATOR_PIN. Quando um comando "on*" direcionado ao sensor é recebido, o estado transita para TURNING_ON, o atuador é ligado (HIGH) e o estado muda para ON. O atuador permanece ligado durante um período de tempo ou até que um comando para desligamento seja enviado, quando o estado transita para TURNING_OFF e o atuador é desligado (LOW), retornando ao estado OFF. Este ciclo garante que a irrigação seja desligada após um período determinado, mesmo que novos comandos "off*" não sejam recebidos. Caso um comando "on*" seja recebido, a contagem do tempo é resetada. O código fonte do *firmware* está disponível em um repositório público no GitHub[2]. A Figura 14 apresenta um fluxograma simplificado do funcionamento deste *firmware*.

Para evitar oscilações excessivas no acionamento do sistema de irrigação, foi implementada uma janela de histerese de 30 segundos. Sempre que o nó recebe um comando para ligar a irrigação, o atuador permanece acionado por esse período, mesmo que a umidade necessária seja alcançada ou o horário de desligamento seja atingido durante esse intervalo. Isso serve para evitar que, após o atingimento dos valores configurados elo

[2] https://github.com/mbssilva/firmwares-tcc

usuário, o sistema permaneça ligando e desligando ao redor do limiar de umidade definido pelo usuário. Após o término da janela, a irrigação é desativada e novas requisições de acionamento são processadas normalmente, permitindo que o sistema responda aos comandos do *gateway*.

Figura 14 – Fluxograma do *firmware* do nó



Fonte: Autoria própria (2024)

5.2.5 CARACTERIZAÇÃO DO SENSOR DE UMIDADE S12

A determinação da umidade do solo consiste em quantificar a quantidade de água presente em uma amostra de solo. Define-se a umidade (h) como a razão entre a massa da água (Ma) contida em um certo volume de solo e a massa da parte sólida (Ms) existente nesse mesmo volume, expressa pela fórmula h = Ma / Ms. Existem diversos métodos para realizar essa determinação, cada um com suas particularidades e níveis de precisão. Entre

os métodos mais comuns, destacam-se: o método da estufa[3], o método da frigideira[4], o método "Speedy"[5] e métodos que utilizam um TDR (*Time Domain Reflectometer*)[6] (SANTOS, 2023). Nesse projeto, o sensor de umidade S12 assume o papel de determinar a umidade do solo e, para isso, é preciso caracterizá-lo.

Figura 15 – Amostra de solo colhida

Fonte: Autoria própria (2024)

O sensor de umidade S12 opera medindo a resistividade do solo. Conforme o solo se

[3] normatizado pela NBR 6457/2016, considerado o método padrão e de maior precisão, consiste em secar uma amostra de solo em estufa a uma temperatura entre 105°C e 110°C até atingir peso constante, determinando assim a massa da água evaporada.

[4] rápido e prático, porém pouco preciso, utiliza aquecimento em uma frigideira para evaporar a água.

[5] utiliza carbeto de cálcio, que reage com a água presente no solo, produzindo um gás cuja pressão é medida para estimar a umidade.

[6] equipamento eletrônico que mede a constante dielétrica do solo, que está diretamente relacionada ao teor de umidade.

torna mais úmido, sua resistência elétrica diminui, permitindo maior condutividade da corrente elétrica. O sensor, energizado com 3,3V, converte a variação de resistência em uma tensão analógica de saída que varia entre 3,3 V (operando a vazio) e 0 V. O módulo de interface que acompanha o sensor facilita a leitura dessa tensão, disponibilizando saídas analógicas e digitais para integração com microcontroladores como o ESP32.

O ESP32, utilizado no nó sensor/atuador, possui um conversor A/D integrado com resolução de 12 bits. O ADC converte a tensão analógica proveniente do sensor S12 em um valor digital entre 0 e 4095. Esse valor digital representa a leitura da umidade do solo, sendo 4095 o valor correspondente ao solo seco (maior tensão) e valores menores representando níveis crescentes de umidade (menor tensão). A resolução de 12 bits permite uma leitura precisa das variações de umidade, possibilitando um controle mais refinado do sistema de irrigação.

Para analisar o comportamento dinâmico do sensor S12 em relação aos níveis de umidade foi conduzido um teste prático. Inicialmente, 2,8 kg de solo foram coletados e dispostos em um recipiente com 25 cm de profundidade. A amostra foi mantida por dois dias em local arejado e coberto para garantir maior evaporação da umidade possivelmente presente nela. Em seguida, com o código listado na Listagem 2 aferiu-se o nível analógico do sensor a seco, obtendo-se o valor de 4095 no ADC do ESP32.

Figura 16 – Caracterização do sensor

Fonte: Autoria própria (2024)

Listagem 2 – Código para caracterização do sensor

```
#define BOTAO 48

const int pinoSensorUmidade = 1;
bool stop = false;
int counter = 1;

void setup() {
  Serial.begin(9600);
  pinMode(pinoSensorUmidade, INPUT);
  pinMode(BOTAO, INPUT_PULLUP);
}

void loop() {
  int botaoValue = digitalRead(BOTAO);
  if (botaoValue == LOW) {
    stop = true;
    int read = analogRead(pinoSensorUmidade);
    Serial.println(String(counter++) + " - " + String(read));
    delay(500);
    while (stop) {
      botaoValue = digitalRead(BOTAO);
      delay(500);
      if (botaoValue == HIGH) stop = false;
    }
  }
}
```

Fonte: Autoria própria (2024)

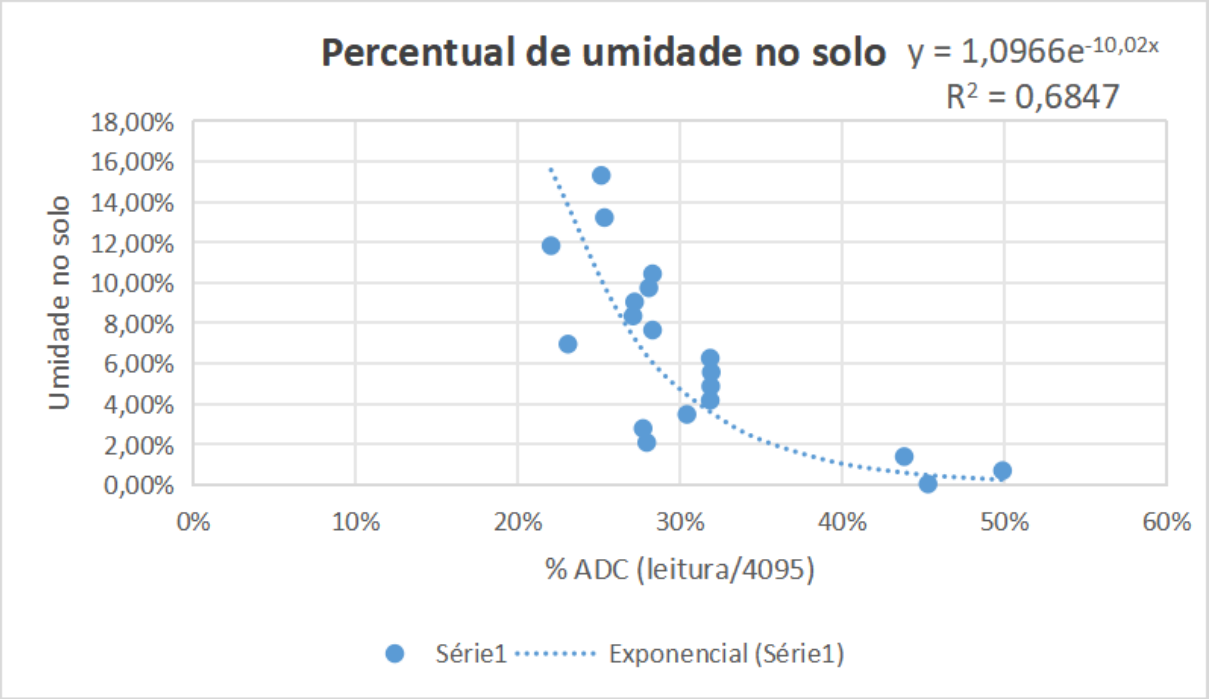
As hastes foram fixadas na amostra de terra, água foi adicionada em incrementos de 20 mL sobre a superfície da terra com o auxílio de uma seringa, e o valor correspondente no ADC foi registrado três vezes (a cada clique de um botão) para cada adição em uma planilha de Excel. Posteriormente, a média das três medições foi calculada. O código de medição utiliza um botão para que, a cada clique, um nova aferição ocorra e seja exibida no monitor. A Figura 15 apresenta a amostra inicial utilizada neste proceso. A Figura 16 apresenta o teste realizado.

Esses passos foram utilizados para quantificar a relação entre a quantidade de água adicionada (mL) por grama de solo e a leitura correspondente do sensor, permitindo a criação de um modelo de regressão para o protótipo. A escolha por esse método, embora não exaustiva para caracterizar precisamente diferentes tipos de solo, se mostrou satisfatória para estabelecer uma correlação entre as leituras do sensor e os níveis de umidade, fornecendo base para o desenvolvimento do protótipo.

Por fim, com o auxílio das ferramentas de linha de tendência do Excel foi possível calcular a regressão e escolher, entre as regressões disponíveis no *software*, a que apresentou

a maior correlação (R^2), no caso, a exponencial (R^2 = 0,6847). Na Figura 17 é possível visualizar a dispersão de 19 dos 24 dados coletados.

Figura 17 – Regressão exponencial para caracterização do sensor



Fonte: Autoria própria (2024)

A Tabela 2 apresenta todas as amostragens realizadas durante o experimento. Observa-se que o valor original de 0 mL foi ajustado para 1 mL na primeira linha, com o objetivo de permitir a criação de linhas de tendência mais diversas, como exponenciais, logarítmicas e polinomiais. Essa modificação visa evitar distorções ou resultados indefinidos nas funções de ajuste que não podem processar valores nulos na variável independente, contribuindo para uma modelagem mais robusta dos dados experimentais. Nota-se que o sensor atinge certa saturação entre 12% e 15% de umidade, apresentando leituras aproximadamente constantes dentro desse intervalo, motivo pelo qual somente as amostras de umidade até 15% foram utilizadas para a análise do sensor. Deve-se destacar a qualidade antioxidante das hastes do sensor, fato que motivou o uso dele para a prototipagem. Porém, como notou-se, a baixa linearidade e rápida saturação são características indesejáveis para esse tipo de aplicação.

Tabela 2 – Valores do ADC e umidade do solo

Índice	Volume (mL)	% ADC	Umidade (%)
1	1	45%	0,03%
2	20	50%	0,70%
3	40	44%	1,39%
4	60	28%	2,09%
5	80	28%	2,78%
6	100	30%	3,48%
7	120	32%	4,18%
8	140	32%	4,87%
9	160	32%	5,57%
10	180	32%	6,27%
11	200	23%	6,96%
12	220	28%	7,66%
13	240	27%	8,35%
14	260	27%	9,05%
15	280	28%	9,75%
16	300	28%	10,44%
17	340	22%	11,83%
18	380	25%	13,23%
19	440	25%	15,32%
20	500	26%	17,40%
21	560	27%	19,49%
22	660	27%	22,97%
23	760	27%	26,45%
24	860	27%	29,93%

Fonte: Autoria própria (2024)

5.2.6 PROJETO DO WEB SERVER

O desenvolvimento do *web server* para o sistema de monitoramento e controle da irrigação foi realizado utilizando Node.js, devido à sua eficiência em operações assíncronas e amplo suporte para servidores *web*. O ambiente de desenvolvimento foi configurado na IDE VSCode, com o projeto baseado no framework Express.js. O Express.js é um *framework web* minimalista e flexível para Node.js que fornece um conjunto de recursos para o desenvolvimento de aplicações *web* e APIs. Ele simplifica tarefas comuns como roteamento, tratamento de requisições HTTP, gestão de *middlewares* e renderização de templates (OPENJS FOUNDATION, 2025). O servidor *web* foi responsável por receber os dados transmitidos pelo *gateway* através de endpoints de uma API REST, processá-los e armazená-los no banco de dados, interpretar as informações armazenadas e, com base nas configurações do usuário, gerar ordens para ligar a irrigação. As rotas da API são apresentadas na Listagem 3.

A arquitetura do servidor foi desenvolvida para suportar a comunicação eficaz entre o servidor e o aplicativo móvel, utilizando o *design pattern* (padrão de projeto) Pub/Sub

Listagem 3 – Rotas configuradas na aplicação com o Express.js

```
1  app.route("/").get(welcome);
2  app.route("/moisture").post(moistureController.save);
3  app.route("/moisture").get(moistureController.index);
4  app.route("/sensor/orders").get(sensorController.index);
5  app.route("/sensor/:id").get(sensorController.show);
6  app.route("/sensor").post(sensorController.save);
7  app.route("/settings").post(settingController.save);
8  app.route("/settings").get(sensorController.show);
```

Fonte: Autoria própria (2025)

para o gerenciamento da coleta dos dados de umidade. Este padrão define uma dependência um-para-muitos entre objetos: uma entidade (o *publisher*) emite um evento e outros objetos (*subscribers*) recebem notificações e reagem a esse evento. No contexto da aplicação, quando o servidor recebe dados de umidade do *gateway*, ele publica um evento com os dados recebidos. Os componentes interessados nesses dados (os *subscribers*) podem então se inscrever neste evento e executar ações específicas, como salvar os dados no banco de dados ou atualizar a interface do usuário. A Figura 18 apresenta o funcionamento deste *design pattern*. A Listagem 4 exibe o código para aquisição de dados de umidade. Ao invés de manter a conexão aberta enquando o servidor realiza a persistência no MongoDB, o evento "HANDLE_REGISTER_MOISTURE" é disparado com o corpo da requisição, seguido do encerramento da chamada sem nenhuma mensagem de resposta, apenas com o código de status 201[7].

Listagem 4 – Função do *Controller* para salvar dados de umidade

```
1  async save(req: Request, res: Response) {
2    try {
3      eventEmitter.publish("HANDLE_REGISTER_MOISTURE", req.body);
4      res.status(201).send();
5    } catch (error) {
6      return res.status(500).send({ message: error.message });
7    }
8  }
```

Fonte: Autoria própria (2025)

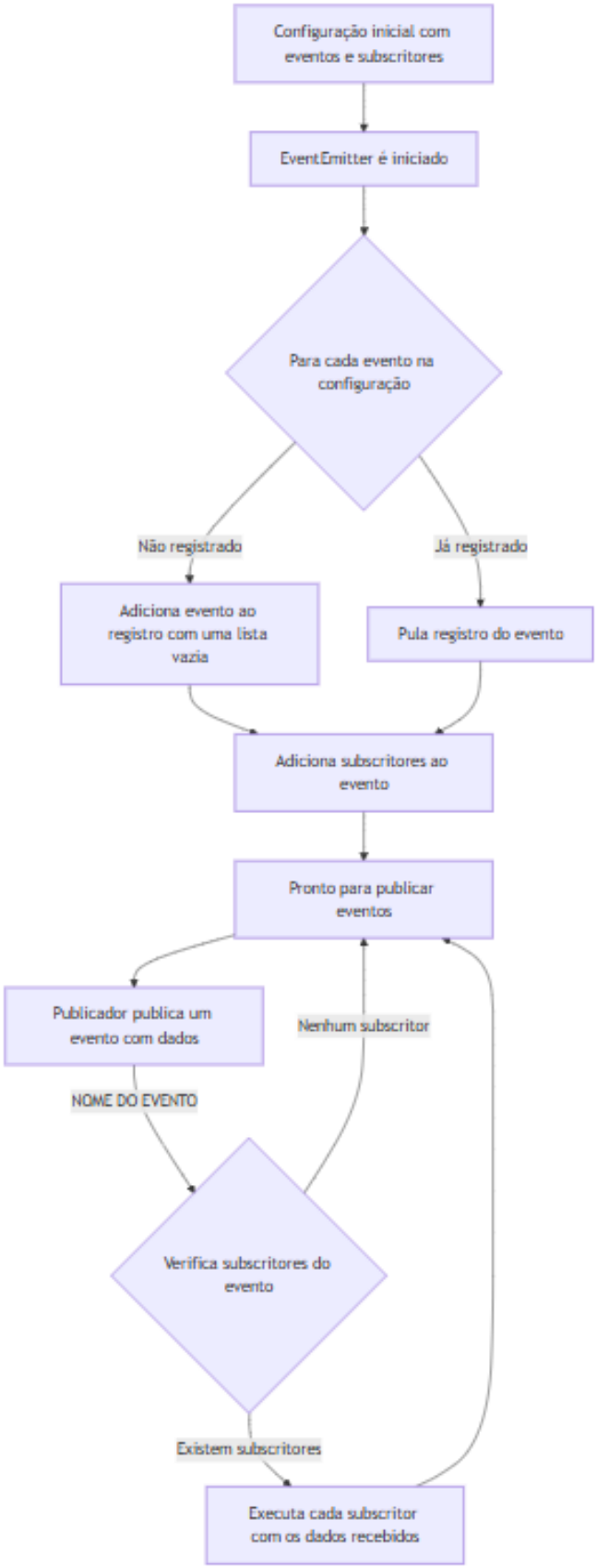
Durante o desenvolvimento, algumas dificuldades foram encontradas. Chamadas frequentes ao *backend* podem ser custosas em termos de recursos e latência, e idealmente deveriam ser substituídas por soluções mais eficientes, como *WebSockets* ou *long polling*, para atualizações em tempo real. O servidor pode apresentar desafios na recuperação de

[7] O código de status de resposta bem-sucedida 201 indica que a solicitação HTTP levou à criação de um recurso (MOZILLA CORPORATION, 2025)

dados do MongoDB em consultas envolvendo um grande número de registros no banco de dados, vários nós cadastrados, relações um-para-muitos (como entre sensores e leituras de umidade) e ordenação em tempo real. Nessa situação, uma melhor performance poderia ser obtida com o uso de cache para dados frequentemente acessados (como as últimas leituras). No desenvolvimento do protótipo, essa abordagem não foi necessária devido à baixa quantidade de registros e à presença de um único nó, o que reduzia significativamente a complexidade das consultas e eliminava a necessidade de otimizações avançadas como o uso de cache. O código fonte do aplicativo está disponível em um repositório público no GitHub[8].

[8] https://github.com/mbssilva/irrigation-backend

Figura 18 – Funcionamento da classe de eventos

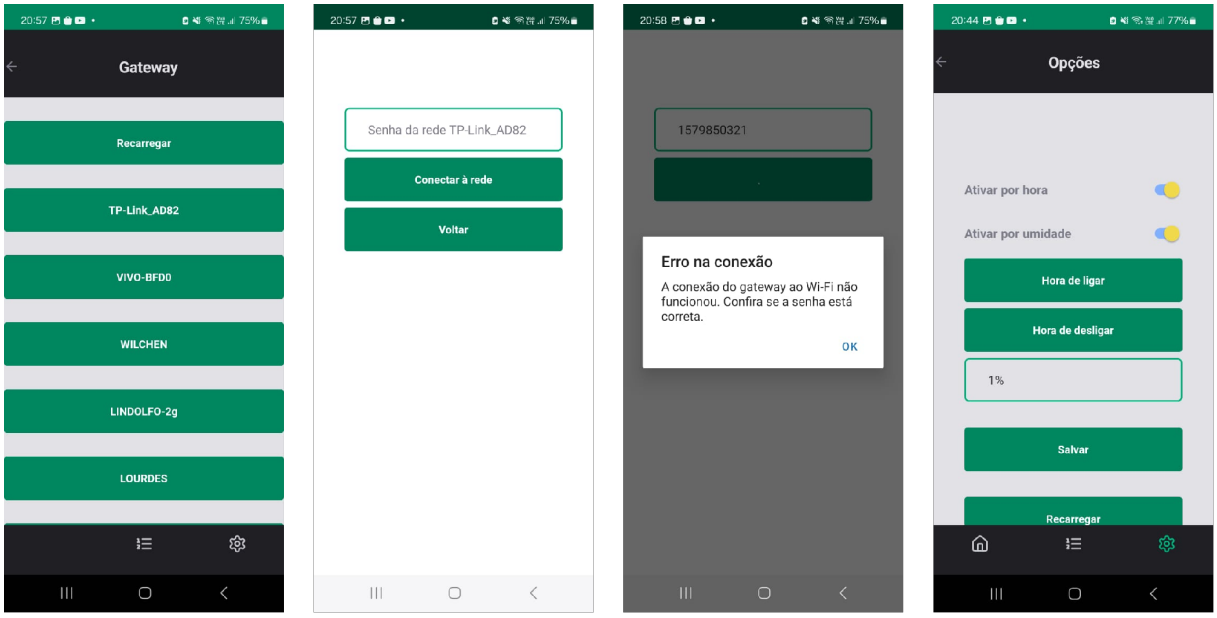


Fonte: Autoria própria (2025)

5.2.7 PROJETO DO APP MOBILE

A criação de uma interface homem-máquina por meio de um aplicativo móvel desenvolvido em React Native oferece uma abordagem flexível e eficaz para interação com sistemas IoT e aplicações similares. Com a capacidade de criar telas personalizadas para apresentar opções de parametrização, exibir dados em tempo real e permitir a comunicação bidirecional com o *backend* através de chamadas HTTP, o aplicativo proporciona uma experiência intuitiva e acessível para os usuários. Além disso, a compatibilidade do React Native com múltiplas plataformas móveis garante que a interface seja facilmente acessível em uma variedade de dispositivos, ampliando ainda mais sua utilidade e alcance. A combinação de uma interface amigável e adaptável com a capacidade de comunicação entre o aplicativo e o *backend* permite que os usuários controlem dispositivos, ajustem configurações e monitorem dados de forma rápida e conveniente, independentemente de sua localização.

Figura 19 – Telas do aplicativo: configuração do *gateway* e da irrigação

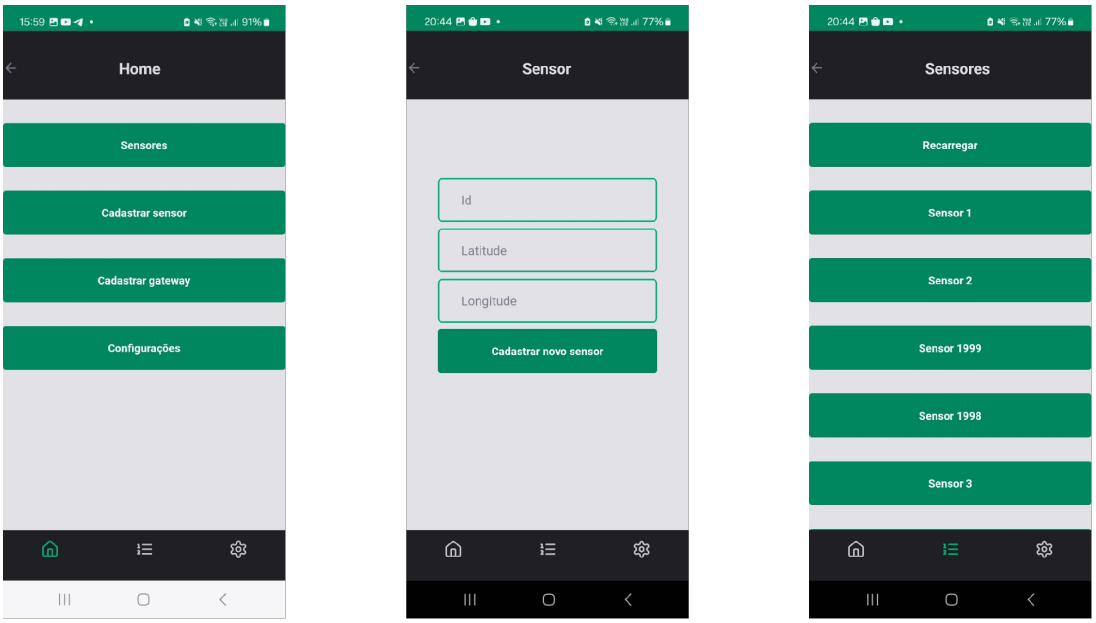


Fonte: Autoria própria (2025)

O aplicativo se comunica com um servidor *web* em Node.js, realizando chamadas HTTP GET para buscar atualizações dos dados nas telas conforme elas forem renderizadas. O desenvolvimento inicial envolveu a configuração do ambiente com o React Native CLI e a criação de um novo projeto. Telas básicas foram implementadas para a visualização dos dados de umidade do solo e o controle dos solenoides de irrigação a partir da edição de horário e nível de umidade, utilizando componentes React Native para exibir as informações recebidas do servidor. Embora uma arquitetura baseada em *WebSockets* seja mais adequada para atualizações em tempo real, as chamadas HTTP periódicas atendem às necessidades do protótipo, permitindo a sincronização dos dados entre o aplicativo e o servidor. O desenvolvimento seguiu uma abordagem iterativa, focando inicialmente na

implementação da comunicação com o servidor e a apresentação dos dados no aplicativo. Em seguida, a integração do *app* com o servidor foi implementada para permitir a configuração dos parâmetros. Pela natureza do protótipo, a segurança não foi implementada nesta fase.

Figura 20 – Telas do aplicativo: cadastramento e listagem de nós



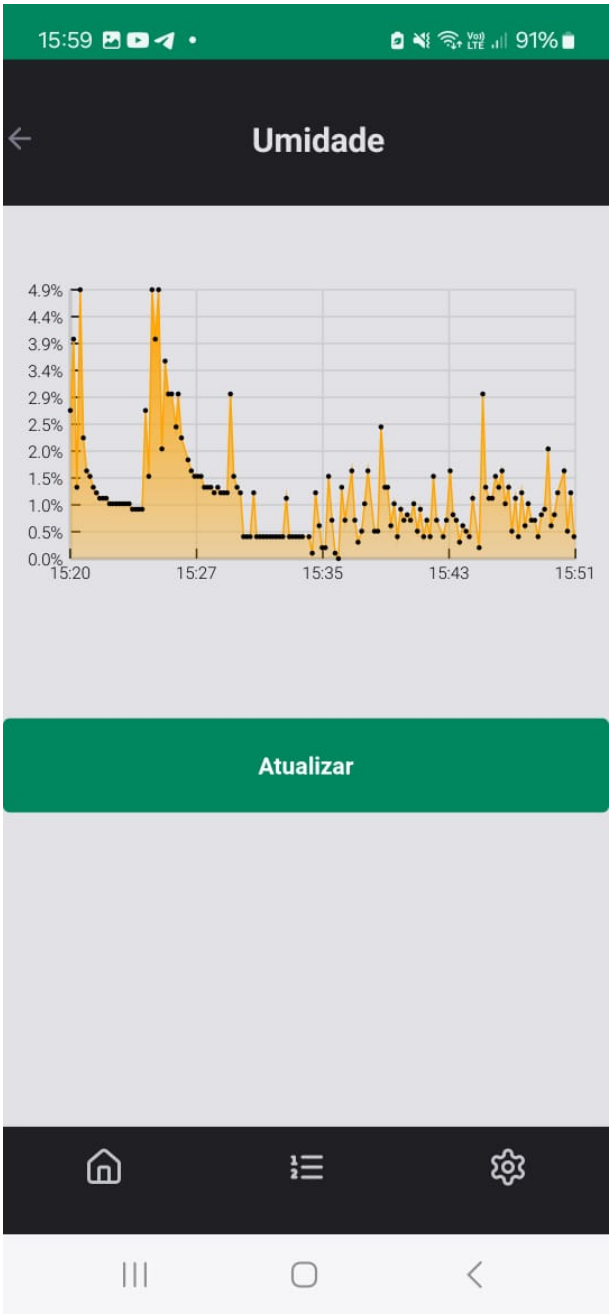
Fonte: Autoria própria (2025)

A navegação entre as telas do aplicativo foi implementada com a biblioteca React Navigation. O React Navigation é uma biblioteca de roteamento e navegação para aplicativos React Native. Ela oferece uma maneira fácil de definir a estrutura de navegação do aplicativo, permitindo a criação de diferentes tipos de navegadores, como navegadores de pilha (*stack navigators*), navegadores de abas (*tab navigators*) e navegadores de gaveta (*drawer navigators*). Cada navegador gerencia um conjunto de telas e define como a transição entre elas ocorre (REACT NAVIGATION, 2025). Isso facilita a criação de interfaces de usuário com múltiplas telas e fluxos de navegação. A Figura 20 mostra algumas telas de conexão do *gateway* as redes Wi-Fi disponíveis. Também, foi implementado uma seleção para parametrização da hora de acionamento e nível de umidade, baseado nos conhecimentos prévios do usuário sobre a necessidade hídrica do plantio, tipo de solo e outras particularidades. A Figura 20 exibe o cadastro e a listagem de sensores no sistema e uma lista de botões que, ao serem clicados, redirecionam para a tela de visualização do histórico de umidade daquele nó, conforme pode ser visto na Figura 21.

O código fonte do aplicativo está disponível em um repositório público no GitHub[9], e permite a reprodução, o estudo e a adaptação do projeto.

[9] https://github.com/mbssilva/irrigation-mobile

Figura 21 – Telas do aplicativo: exibição do histórico de umidade



Fonte: Autoria própria (2025)

5.2.8 INTEGRAÇÃO DO SISTEMA

O sistema de irrigação é composto por várias partes integradas e baseado no servidor *web* como o elemento central conectando todas as entidades. A primeira parte do sistema incluiu os sensores de umidade do solo e os circuitos de comando que controlaram as válvulas solenoides. O *gateway* foi programado para receber dados dos sensores via comunicação LoRa e processá-los. Simultaneamente, ele se conecta à rede Wi-Fi para enviar esses dados a um servidor *web* desenvolvido em Node.js. O servidor *web* armazenou os dados no banco de dados MongoDB e disponibilizou uma API para acesso remoto. Assim,

o *gateway* funcionou como um intermediário, coletando dados do campo e transmitindo-os para o servidor, permitindo o monitoramento contínuo das condições do solo.

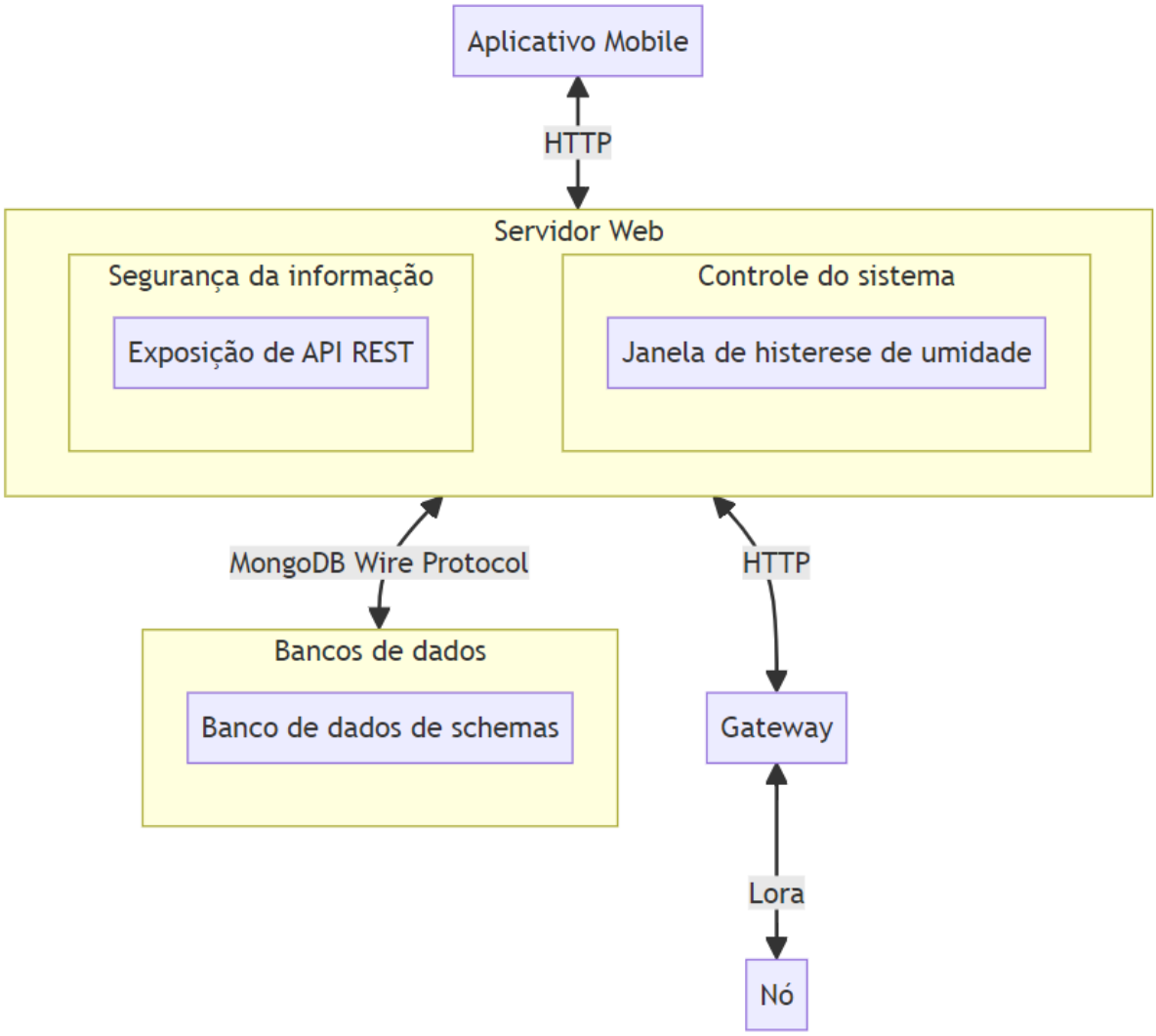
A segunda parte do sistema envolveu o aplicativo móvel desenvolvido em React Native, que se comunicou com o servidor *web* para exibir os dados de umidade do solo e permitir o controle remoto das válvulas solenoides. O aplicativo enviou comandos ao *web server*, que, por sua vez, retransmitiu esses comandos ao ESP32. Quando o ESP32 recebeu um comando, ele ajustou as válvulas solenoides usando um pino de saída lógico para controlar o fluxo de água de acordo com as necessidades das plantas. Esse processo ocorreu de maneira contínua, permitindo que o usuário gerenciasse a irrigação de suas culturas remotamente. A integração de todas essas partes resultou no protótipo do sistema de irrigação automatizada, capaz de monitorar e controlar a irrigação de forma remota.

Por fim, o servidor transmite o comando para o *gateway*. O *gateway*, por sua vez, retransmite esse comando via LoRa para o nó sensor/atuador no qual, ao receber o comando de ativação, o ESP32 presente no nó energiza um pino de saída digital, que comuta um relé de 10A e 3,3V. Devido à diferença de tensão entre os pinos GPIO do ESP32 (3,3V) e a tensão de operação do solenoide (12V), a utilização do relé se faz necessária para isolar os circuitos e garantir o acionamento adequado. Uma alternativa ao relé é um driver baseado no chaveamento de um transistor. O fechamento dos contatos do relé estabelece a conexão com uma fonte de alimentação externa de 12V, energizando o solenoide e, consequentemente, abrindo a válvula de irrigação. A duração da irrigação, configurada previamente no *firmware*, é monitorada pelo ESP32 presente no nó. Decorrido o tempo predefinido, o ESP32 do nó desativa o pino de saída, interrompendo a corrente no circuito do relé e, por consequência, o fluxo de energia para o solenoide, cessando a irrigação. A utilização de uma fonte de alimentação externa de 12V garante a operação do solenoide dentro de suas especificações, enquanto o relé e o ESP32 do nó asseguram o controle do sistema de irrigação. Para os fins de prototipagem desejados, essa solução atendeu ao teste do sistema.

Simultaneamente ao controle do acionamento da irrigação, o nó sensor/atuador executa a leitura contínua da umidade do solo por meio do sensor S12. Este, ao detectar variações na umidade do solo, modula sua saída analógica, que é então convertida para um valor digital pelo conversor analógico-digital (ADC) integrado ao ESP32 presente no nó. Este valor digital, representando o nível de umidade, é então encapsulado em um pacote de dados e transmitido sem fio, utilizando o protocolo LoRa e o transceptor SX1262, para o *gateway*. O *gateway*, também baseado em ESP32, recebe o pacote LoRa e, após verificar sua integridade, o encaminha para o servidor *web* através de uma conexão Wi-Fi. No servidor, implementado em Node.js e utilizando o framework Express.js, os dados recebidos são processados e convertidos em níveis de umidade baseados na função que caracteriza o sensor, e armazenados no banco de dados MongoDB, gerenciado pelo Mongoose. A partir do momento em que os dados são persistidos no banco de dados, eles se tornam acessíveis

através da API REST disponibilizada pelo servidor. O aplicativo móvel, desenvolvido em React Native, realiza requisições à API para obter os dados de umidade e os exibe na interface para o usuário, permitindo o monitoramento em tempo real das condições do solo e auxiliando na tomada de decisões sobre o manejo da irrigação. Este fluxo contínuo de coleta, transmissão, processamento e visualização dos dados garante o funcionamento integrado do sistema de irrigação automatizada.

Figura 22 – Integração do sistema



Fonte: Autoria própria (2024)

6 RESULTADOS

O desenvolvimento do protótipo de sistema de irrigação automatizada teve como objetivo principal a criação de uma solução estável e eficiente para o controle da irrigação com base nas condições de umidade do solo. As expectativas iniciais incluíam a integração confiável entre o *gateway* e os nós de sensoriamento e atuação, a comunicação via LoRa, a execução precisa dos comandos para as válvulas solenoides e o processamento e armazenamento dos dados pelo servidor *web* em Node.js, permitindo a visualização da variação de umidade. Além disso, a interação entre o aplicativo móvel em React Native e o servidor *web* deveria ser estável e rápida, proporcionando uma experiência de usuário intuitiva e operando em dispositivos Android e iOS, permitindo o monitoramento em tempo real e o envio de comandos sem atrasos significativos.

Durante a implementação, algumas dificuldades foram encontradas. A comunicação LoRa, apesar de ter se mostrado eficaz em distâncias consideráveis, apresentou instabilidades pontuais em ambientes com muitas interferências, exigindo ajustes na configuração dos parâmetros de transmissão. A integração inicial entre o ESP32 do *gateway* e o servidor *web* demandou um esforço considerável para garantir a sincronia e o correto processamento dos dados, principalmente no tratamento de possíveis perdas de pacotes. O desenvolvimento do aplicativo móvel também apresentou desafios, principalmente na adaptação da interface para diferentes tamanhos de tela e versões de sistemas operacionais, buscando garantir a usabilidade em diversos dispositivos.

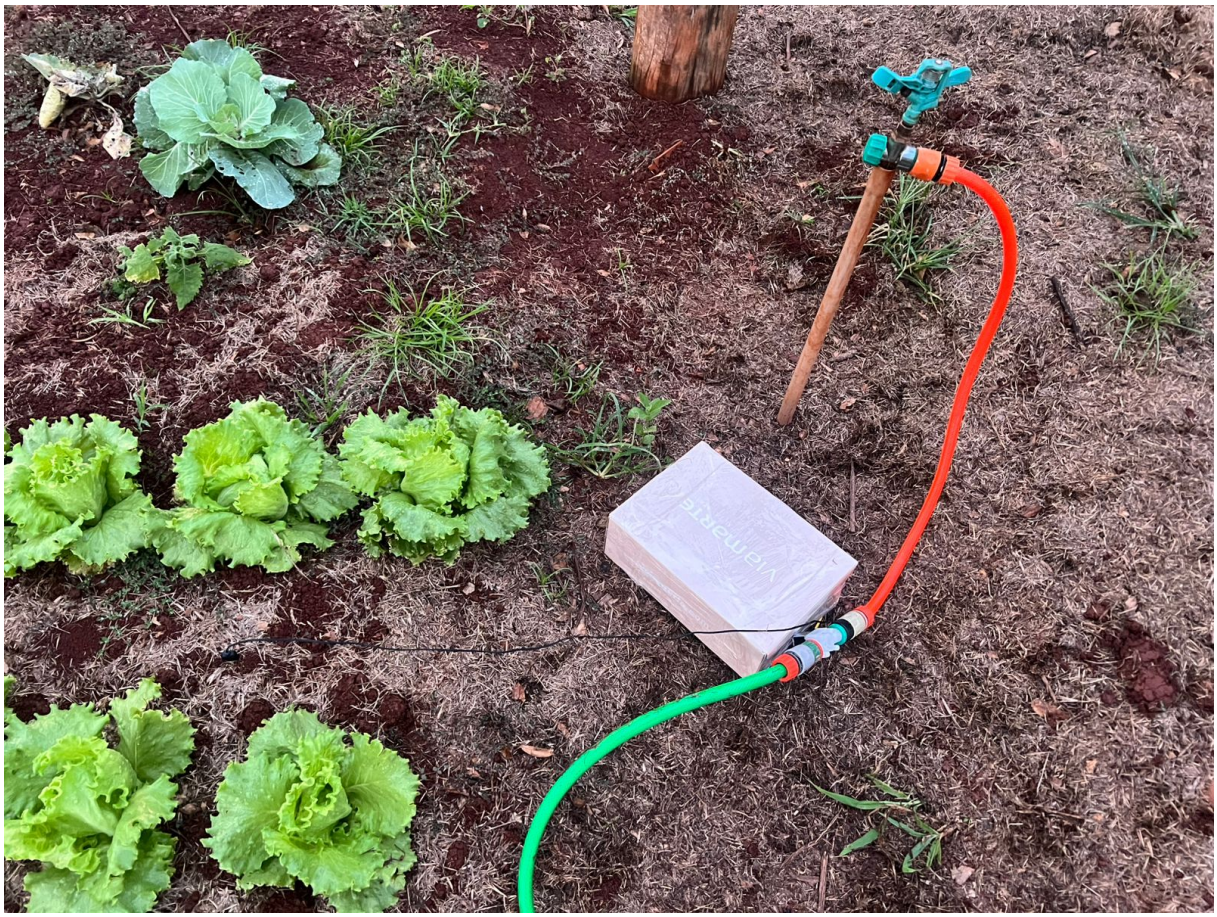
Apesar dos desafios encontrados, o protótipo atingiu os objetivos propostos. A integração entre o *gateway*, o nó sensor/atuador e o servidor *web* se mostrou funcional, com a comunicação LoRa operando de forma satisfatória na maioria dos cenários testados. Os comandos para as válvulas solenoides foram executados com precisão, controlando o fluxo de água conforme os parâmetros definidos, especialmente no modo de operação por horário (no qual o usuário determina o horário inicial e final da irrigação). O servidor *web* em Node.js processou e armazenou os dados de umidade de forma satisfatória, disponibilizando-os para visualização no aplicativo móvel. A interação entre o aplicativo e o servidor *web* demonstrou estabilidade e rapidez, proporcionando uma experiência de usuário satisfatória, com o monitoramento em tempo real e o envio de comandos funcionando conforme o esperado.

Em conclusão, o protótipo demonstrou a viabilidade da implementação prática do sistema de irrigação automatizada, cumprindo as necessidades esperadas e facilitando o manejo da irrigação agrícola. O sistema fornece subsídios para futuros estudos na área de automação agrícola e IoT, e sua disponibilização sob licença aberta proporciona aos projetistas insumos para projetos acadêmicos ou desenvolvimento de produtos, permitindo seu uso para fins comerciais, contribuindo para a inovação no setor.

6.1 TESTE DE ACIONAMENTO

Esta seção apresenta alguns testes de integração do sistema realizado e os resultados. Inicialmente, o *gateway* foi instalado em um local abrigado e seguro, e configurado para acessar a rede local de internet. Em seguida, dentro de uma caixa, foi inserido o nó de sensoriamento e atuação, uma bateria de 12V (para acionamento do solenoide), um carregador portátil de 5V (para alimentação do nó), o sensor e o solenoide. A caixa foi então revestida com plástico filme e instalada entre o aspersor e a linha d'água, conforme mostrado na Figura 23.

Figura 23 – Montagem do nó em campo



Fonte: Autoria própria (2025)

Primeiro, com os modos de acionamento por umidade e por horário foram desligados, e testou-se o acionamento manual, o qual apresentou excelente precisão, ligando e desligando conforme o botão de acionamento no *gateway* era pressionado. Em seguida, foi configurado através do aplicativo de celular o acionamento por horário, com uma janela de 1 minuto e hora inicial igual à hora do teste. O sistema foi capaz de acionar a irrigação e mantê-la ligada conforme o esperado. Mesmo após o atingimento da hora de desligamento, o nó manteve o aspersor ligado por mais alguns segundos, novamente, conforme o esperado, dado a janela de histerese implementada. Por fim, o gráfico de umidade foi ana-

lisado e uma umidade superior àquela medida no momento foi definida, juntamente com o modo de acionamento por umidade. Novamente, o sistema foi acionado pelo *gateway* em virtude da ordem recebida do servidor, pois a umidade estava abaixo da definida. Ao final dos testes, uma captura da tela de exibição da umidade foi realizada. Durante o teste, o sensor parou de funcionar, devido a desconexão de um pino na protoboard que estava dentro da caixa e, por isso, não houve coleta de umidade após as 19h44, conforme é exibido na Figura 24.

Figura 24 – Dados de umidade após o teste

Fonte: Autoria própria (2025)

6.2 PRINCIPAIS DIFICULDADES ENCONTRADAS

O desenvolvimento do protótipo de irrigação automatizada apresentou diversos desafios que exigiram soluções específicas e adaptações no projeto. Uma das principais dificuldades residiu na integração entre o ESP32 e o servidor *web*. A comunicação entre esses dois componentes, essencial para o fluxo de dados do sistema, demonstrou problemas de sincronia e processamento, principalmente no tratamento de perdas de pacotes. A natureza assíncrona da comunicação, combinada com a necessidade de garantir a integridade dos dados transmitidos, demandou a implementação de mecanismos de confirmação de recebimento. Essa complexidade aumentou significativamente o tempo de desenvolvimento e exigiu um esforço considerável de depuração para assegurar a confiabilidade da troca de informações entre o hardware e o servidor. Além disso, a escolha inicial da biblioteca WiFi.h para o *gateway* se mostrou um gargalo, pois suas requisições blocantes comprometiam severamente o desempenho do sistema, introduzindo latência excessiva e até mesmo a perda de dados. A substituição por uma biblioteca alternativa, com suporte a operações não bloqueantes e gerenciamento eficiente de conexões, foi crucial para otimizar a comunicação e garantir a responsividade do *gateway*, permitindo o processamento simultâneo de múltiplas requisições.

Outro obstáculo significativo foi a natureza *half-duplex* da comunicação LoRa. Em sistemas *half-duplex*, a transmissão e a recepção de dados não podem ocorrer simultaneamente, o que significa que, enquanto o rádio LoRa está transmitindo dados do sensor para o *gateway*, ele não pode receber comandos ou confirmações. Essa limitação inerente ao protocolo resultou na perda de alguns dados, principalmente em cenários com alta frequência de transmissões ou quando o *gateway* precisava enviar comandos urgentes para os nós sensores. A análise do comportamento do sensor de umidade S12 de umidade também revelou uma limitação importante relacionada às características do equipamento. A baixa linearidade, que dificulta a conversão direta das leituras analógicas em valores precisos de umidade, e a rápida saturação, que o torna menos sensível a variações de umidade em níveis mais altos, representaram desafios para a obtenção de medições confiáveis.

Por fim, a escolha da API REST para a comunicação entre o servidor *web* e o aplicativo móvel apresentou limitações de performance, especialmente considerando o potencial aumento no volume de dados gerados pelo sistema em uma implantação em larga escala. A arquitetura REST, baseada em requisições e respostas HTTP, pode gerar uma sobrecarga no servidor quando lida com um grande número de requisições simultâneas ou com a necessidade de atualizações em tempo real. Nesse contexto, a utilização de tecnologias como WebSockets ou *long polling* se mostraria mais adequada, permitindo uma comunicação bidirecional em tempo real e reduzindo a latência na troca de informações entre o aplicativo e o servidor. A consideração dessas alternativas para futuras iterações do projeto se mostra crucial para garantir a escalabilidade e o desempenho do sistema em cenários de alta demanda.

7 CONCLUSÃO

O presente projeto propôs o desenvolvimento de um sistema de irrigação automatizada, utilizando a plataforma ESP32 e comunicação LoRa, com o objetivo de otimizar o uso da água na agricultura e facilitar o monitoramento remoto das condições do solo. Através deste trabalho, demonstrou-se a viabilidade da integração de tecnologias IoT para o controle preciso da irrigação, com potencial para reduzir o desperdício de água e aumentar a eficiência das práticas agrícolas. A escolha do ESP32, com seus recursos de processamento e conectividade, combinada com a tecnologia LoRa, que oferece longo alcance e baixo consumo de energia, mostrou-se adequada para as necessidades do projeto, permitindo a comunicação entre os sensores de umidade do solo e o *gateway*, mesmo em áreas extensas. As dificuldades encontradas, como a complexa integração entre o ESP32 e o servidor web, a natureza *half-duplex* da comunicação LoRa e a necessidade de caracterizar o sensor para diferentes tipos de solo, apontam para a importância de futuras iterações e aprimoramentos no sistema. No entanto, o protótipo desenvolvido representa um passo importante rumo a soluções mais eficientes e sustentáveis para a irrigação, abrindo caminho para o desenvolvimento de sistemas mais robustos, escaláveis e adaptáveis às diversas necessidades dos agricultores.

7.1 TRABALHOS FUTUROS

Como todo projeto de engenharia, há espaço para aprimoramentos e expansões que podem aumentar ainda mais a eficiência, confiabilidade e aplicabilidade do sistema. Os trabalhos futuros propostos visam refinar o protótipo atual, abordando aspectos como aprimoramentos de hardware, otimização de software, escalabilidade e usabilidade.

Uma das principais áreas de desenvolvimento futuro concentra-se na melhoria da robustez e portabilidade do sistema. A implementação de um circuito de alimentação autônomo, com bateria recarregável e placa solar de baixa potência, permitirá a operação do sistema em locais remotos sem acesso à rede elétrica, ampliando suas possibilidades de aplicação. Além disso, o desenvolvimento de envólucros resistentes à umidade para os nós sensores garantirá a proteção dos componentes eletrônicos contra as intempéries, aumentando a durabilidade e a confiabilidade do sistema em ambientes externos. A criação de uma placa de circuito impresso compacta otimizará o design do hardware, reduzindo o tamanho dos módulos e facilitando a montagem e a integração dos componentes.

No âmbito do software e da comunicação, trabalhos futuros incluem aprimoramentos significativos. A utilização de uma biblioteca Wi-Fi não bloqueante e com melhor gerenciamento de conexões para o *gateway* otimizará a comunicação com o servidor, evitando gargalos e melhorando o desempenho geral do sistema. A caracterização do sensor de

umidade para diferentes tipos de solo, com a opção de seleção para o usuário, garantirá leituras mais precisas e adaptadas às diversas condições de plantio. A implementação de tasks personalizadas no ESP32, explorando seus recursos dual-core para lidar com as chamadas long polling, isolará esse processamento e evitará impactos negativos no desempenho de outras tarefas críticas, como a comunicação LoRa e o processamento dos dados dos sensores.

A implementação de um sistema de *logging*, tanto no ESP32 quanto no servidor *web*, permitirá o registro de eventos importantes, erros e dados de desempenho, facilitando a depuração, a identificação de problemas e o monitoramento contínuo em produção por parte do admnistrador do sistema, em especial, pelo fato da utilização do padrão de projeto Pub/Sub, que cria assincronismos na aplicação, ou seja, as tarefas deixam de ser executadas em uma ordem pré-determinada e ocorrerem conforme são finalizadas e com a publicação de eventos, o que gera paralelismo na aplicação. Por isso, capturar as mensagens de erro em um visualizador possibilitaria o *tracing* ou *error tracking*. Além disso, testes de escalabilidade com um número maior de nós sensores permitirão analisar o desempenho do *gateway* em cenários de alta demanda, identificando possíveis gargalos e endereçando melhorias para garantir a escalabilidade do sistema para aplicações em larga escala. Essas melhorias, em conjunto, visam consolidar um sistema de irrigação automatizada mais completo, eficiente e adaptável às diversas necessidades dos usuários.

No âmbito de sensoriamento, uma característica de grande interesse em sistemas de irrigação é o volume de água aplicado. Para possibilitar o sensoriamento desta variável, um sensor de vazão poderia ser adicionado ao nó. Assim, dados como a vazão instantânea e a vazão média poderiam ser exibidos ao usuário, bem como, o volume total irrigado, que pode ser calculado através da integração discreta da vazão.

Por fim, uma direção promissora para trabalhos futuros consiste na substituição da válvula solenoide normalmente fechada por um solenoide servo-operado. As válvulas solenoides, embora eficazes, consomem energia continuamente enquanto energizadas para manter a válvula aberta. A adoção de um servo mecanismo, que consome energia apenas durante a mudança de posição (abertura ou fechamento), representaria uma economia significativa de energia, viabilizando a operação autônoma do nó sensor/atuador com uma fonte de energia portátil, como uma bateria e um painel solar de baixa potência, eliminando a necessidade de uma fonte de alimentação externa. No entanto, a implementação de servos motores apresenta um desafio financeiro, dado o custo geralmente mais elevado desses dispositivos em comparação com as válvulas solenoides normais. Diferentemente de uma válvula normalmente fechada, que permanece fechada sem energia e abre quando energizada, um servo motor controla a posição da válvula de forma precisa através de um sinal de controle, permitindo inclusive o controle gradual da abertura. A Figura 25 exibe um exemplo de válvula servo-operada encontrada no mercado, fabricada e comercializada pela Danfoss, para aplicações de refrigeração (DANFOSS, 2025).

Figura 25 – Válvulas solenoides servo-operadas para refrigeração

Fonte: Danfoss (2025)

REFERÊNCIAS

DANFOSS. **Válvulas solenoides EVR, diretas ou servo-operadas**. [S.l.], 2025. Disponível em: <https://www.danfoss.com/pt-br/products/dcs/valves/solenoid-valves/solenoid-valves-for-hvac-r/evr-solenoid-valves/>. Acesso em: 20 de janeiro de 2025.

EDITORA MELHORAMENTOS LTDA. **Michaelis Dicionário Brasileiro da Língua Portuguesa**. 2024. Disponível em: <https://michaelis.uol.com.br/moderno-portugues>. Acesso em: 02 de junho de 2024.

ELMASRI, R.; NAVATHE, S. B. **Fundamentals of database systems**. 4. ed. [S.l.]: Pearson Education, Inc, 2004. 1030 p.

ESPRESSIF SYSTEMS. **ESP32 Datasheet V1.4**. Xangai, 2017. 56 p.

GERMANO, S. B. **Protótipo de Solução Iot Para uma Estação Meteorológica Aplicando Tecnologia Lora no Ambiente Agro**. [S.l.], 2022.

LIMA, K. R. **Desenvolvimento de Protótipo para Automação de Sistema de Irrigação Utilizando IoT**. [S.l.], 2019.

META. **Reatc Native documentation**. [S.l.], 2024. Disponível em: <https://reactnative.dev/docs/intro-react-native-components>. Acesso em: 26 de maio de 2024.

MICROSOFT. **TypeScript handbook**. [S.l.], 2024. Disponível em: <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes-oop.html>. Acesso em: 27 de maio de 2024.

MONGODB, INC. **MongoDB Architecture Guide**: The foundational concepts that underpin the architecture of mongodb. [S.l.], 2021. 18 p.

MONGOOSE. **Mongoose documentation**. [S.l.], 2024. Disponível em: <https://mongoosejs.com/docs/index.html>. Acesso em: 27 de maio de 2024.

MOZILLA CORPORATION. **201 Created**. [S.l.], 2025. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/201>. Acesso em: 16 de dezembro de 2024.

NEVES, W. d. S.; RODRIGUES, E. C. **Circular técnica n. 253**. Belo Horizonte, 2017. 3 p.

NICOLOSI, D. E. C. **Microcontrolador 8051 detalhado**. 6. ed. [S.l.]: Editora Erica, 2004. 227 p.

OPENJS FOUNDATION. **Node.js documentation**. [S.l.], 2024. Disponível em: <https://nodejs.org/en/about>. Acesso em: 25 de maio de 2024.

OPENJS FOUNDATION. **Express**. [S.l.], 2025. Disponível em: <https://expressjs.com/>. Acesso em: 15 de dezembro de 2024.

PAOLINELLI, A.; DOURADO, D. N.; MANTOVANI, E. C. **Diferentes abordagens sobre agricultura irrigada no Brasil: Técnica e Cultura**. 4. ed. Piracicaba, 2021. 598 p.

PARKER HANNIFIN. **Válvulas Solenóide**: Catálogo 4201-2 br. Jacareí, 2002. 80 p.

PEREIRA, S. d. L. **Linguagem C++**. São Paulo, 1999. 227 p.

REACT NAVIGATION. **Getting started**. [S.l.], 2025. Disponível em: <https://reactnavigation.org/docs/getting-started/>. Acesso em: 10 de janeiro de 2025.

SANTOS, J. **Determinação da umidaade do solo**. [S.l.], 2023. 4 p.

SCHILDT, H. **C completo e total**. 3. ed. São Paulo: Pearson Education do Brasil, 1996. 816 p.

SEMTECH CORPORATION. **SX1261/2 Datasheet**. Camarillo, 2021. 114 p.

SEMTECH CORPORATION. **LoRa PHY**. [S.l.], 2024. Disponível em: <https://www.semtech.com/lora/what-is-lora>. Acesso em: 29 de junho de 2024.

TESTEZLAF, R. **Irrigação: Métodos, sistemas e aplicações**. Campinas, 2012. 209 p.

TOCCI GREGORY L. MOSS, N. S. W. R. J. **Sistemas digitais: princípios e aplicações**. 3. ed. São Paulo: Pearson Education do Brasil, 2011. 817 p.