Alisson Gabriel Lucas da Silva

Desenvolvimento de um sistema de monitoramento de variáveis pluviométricas utilizando o microcontrolador ESP32

Natal – RN

Agosto de 2024

Alisson Gabriel Lucas da Silva

Desenvolvimento de um sistema de monitoramento de variáveis pluviométricas utilizando o microcontrolador ESP32

Trabalho de Conclusão de Curso de Engenharia Mecatrônica da Universidade Federal do Rio Grande do Norte, apresentado como requisito parcial para a obtenção do grau de Bacharel em Engenharia Mecatrônica

Orientador: Dr. Heitor Medeiros Florencio

Universidade Federal do Rio Grande do Norte – UFRN

Departamento de Engenharia de Computação e Automação – DCA

Curso de Engenharia Mecatrônica

Natal – RN

Agosto de 2024

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Silva, Alisson Gabriel Lucas da.
Desenvolvimento de um sistema de monitoramento de variáveis pluviométricas utilizando o microcontrolador ESP32 / Alisson Gabriel Lucas da Silva. - 2024.
91 f.: il.

Trabalho de Conclusão de Curso - TCC (graduação) - Universidade Federal do Rio Grande do Norte, Centro de Tecnologia, Curso de Engenharia Mecatrônica, Natal, RN, 2024.
Orientação: Prof. Dr. Heitor Medeiros Florencio.

1. Internet das Coisas - TCC. 2. Sistema de Monitoramento - TCC. 3. Microcontrolador ESP32 - TCC. I. Florencio, Heitor Medeiros. II. Título.

RN/UF/BCZM CDU 004.738.5:551.577

Elaborado por Fernanda de Medeiros Ferreira Aquino - CRB-15/301

Alisson Gabriel Lucas da Silva

Desenvolvimento de um sistema de monitoramento de variáveis pluviométricas utilizando o microcontrolador ESP32

Trabalho de Conclusão de Curso de Engenharia Mecatrônica da Universidade Federal do Rio Grande do Norte, apresentado como requisito parcial para a obtenção do grau de Bacharel em Engenharia Mecatrônica

Orientador: Dr. Heitor Medeiros Florencio

Banca Examinadora:

Prof. Dr. Luiz Eduardo Cunha Leite - Avaliador Interno
ECT/UFRN

Prof. Msc. Camilo Gustavo Araújo Alves - Avaliador Interno
MEC/UFRN

Wisley Denley Moura de Lima - Avaliador Externo
Dois A Engenharia e Tecnologia LTDA

Prof. Dr. Heitor Medeiros Florencio - Orientador
IMD/UFRN

Natal – RN
Agosto de 2024

Dedico o trabalho a todos que me apoiaram nos momentos mais difíceis, fizeram-me acreditar e agradeço o sentimento de encorajamento, pois as palavras e atitudes foram essenciais para a realização deste trabalho. Sem a compreensão e dedicação, este projeto não teria sido elaborado com êxito.

AGRADECIMENTOS

Nossas conquistas só são possíveis graças ao trabalho de várias pessoas, diretamente ou indiretamente, e é por esse motivo que eu agradeço primeiramente a Deus por ter possibilitado esse acontecimento.

Agradeço a minha companheira de vida, Maria Elita, por ter me apoiado e motivado em todos os momentos e em todos os aspectos da vida. Também agradeço aos meus familiares, especialmente minha mãe, Vera Lucia, e meu pai, José Araújo, que me apoiaram em todos os momentos da minha jornada acadêmica e vida.

Agradeço ao meu orientador, Heitor Medeiros, pela colaboração, organização nas reuniões, apoio, ensinamentos sobre pesquisa e elaboração do trabalho, paciência e conselhos em toda produção.

Aos meus professores, que foram tantos, dentro e fora de sala, ótimos profissionais e educadores.

Aos meus chefes, Wisley Moura e Camilo Alves, que me proporcionaram espaço, apoio e paciência para implementar a solução do trabalho nos projetos da empresa.

Aos meu companheiros de trabalho pela força de cada dia para a conclusão do trabalho. Agradecimento especial a Andryw Afonso e Edilson Melo pela grande ajuda.

Meus amigos e colegas de vida por sempre me incentivarem no meu trabalho e carreira.

Em suma, gostaria de deixar os agradecimentos a todos que fizeram parte da meu crescimento pessoal e acadêmico. Muito obrigado por tudo!

RESUMO

Este trabalho apresenta o desenvolvimento de um dispositivo IoT (*Internet of Things*) para detecção de chuva, obtendo variáveis pluviométricas utilizando um microcontrolador da série ESP32. A criação do dispositivo foi motivada pela necessidade de monitorar a intensidade e frequência das chuvas em uma obra de construção civil, fornecendo dados para otimizar a gestão da obra. A metodologia do trabalho inclui o projeto eletrônico e mecânico do dispositivo, além da arquitetura do sistema, funcionalidades e protocolos de comunicação utilizados. São discutidos os sensores aplicados ao projeto, suas estruturas, compatibilidades com o microcontrolador ESP32, protocolos de comunicação e seus resultados, obtidos através de testes em laboratório e em campo. Os resultados são apresentados através de um painel de gestão visual desenvolvido com a ferramenta Power BI.

Palavras-chaves: ESP32; Internet da Coisas; Detecção de chuva; Sistema de monitoramento; Pluviômetro.

ABSTRACT

This paper presents the development of an IoT (Internet of Things) device for rain detection, acquiring pluviometric variables using an ESP32 series microcontroller. The creation of the device was motivated by the need to monitor the intensity and frequency of rain at a construction site, providing data to optimize site management. The methodology includes the electronic and mechanical design of the device, as well as the system architecture, functionalities, and communication protocols used. The sensors applied to the project, their structures, compatibilities with the ESP32 microcontroller, communication protocols, and their results obtained through laboratory and field tests are discussed. The results are presented through a visual management dashboard developed with Power BI.

Keywords: ESP32; Internet of Things; Rain detection, Monitoring System; Rain Gauge.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

IoT	*Internet das Coisas*
GPS	*Global Positioning System*
RTC	*Real Time Clock*
mm	Milímetros
cm	Centímetros
m^3	Metros cúbicos
cm^3	Centímetros cúbicos
ml	Mililitros
Wi-Fi	*Wireless Fidelity*
API	*Application Programming Interface*
SoC	*System on a Chip*

SUMÁRIO

1 INTRODUÇÃO

A construção civil é um área muito ampla, com variadas grandezas de infraestrutura, isto é, abrange de uma obra de um pequeno prédio até obras com centenas de quilômetros de extensão como algumas obras do setor de geração de energia como usinas eólicas e fotovoltaicas. Todos os tipos de obra necessitam de planejamento, no entanto, as mais extensas necessitam de uma maior demanda devido à quantidade de recursos envolvidos, como máquinas pesadas, veículos leves, operadores qualificados e gestão do tempo. Essa combinação é fundamental, gerir os recursos para que se possa otimizar o tempo de cada atividade, agilizando o término da obra, diminuindo custos e aumentando o lucro (M&T, 2024).

Em obras de grande porte, onde a demanda por estratégias eficazes é maior, vários fatores podem interromper o andamento da execução do projeto, sejam eles de natureza natural ou operacional. A paralisação das atividades em frentes de trabalho é crítica, pois gera tempo ocioso para máquinas e trabalhadores, comprometendo o cumprimento das metas e reduzindo a margem de lucro, podendo até resultar em multas por atraso. Obras extensas, como a construção de um complexo de energia eólica, frequentemente possuem várias frentes de trabalho distantes entre si. Esses projetos, que podem se estender por centenas de quilômetros e geralmente ocorrem em locais remotos, enfrentam desafios adicionais devido à dificuldade de acesso à internet e sinal de telecomunicação.

Dentre os problemas que podem ocorrer em uma obra, as intempéries são os mais complicados, pois não podem ser controlados. Em zonas urbanas, as chuvas geralmente não possuem a mesma intensidade ou frequência em comparação às regiões de altitudes mais elevadas, como as serras, onde a maioria dos parques eólicos é construída (ESCOLA, 2024). Durante as chuvas, o solo molhado diminui o atrito, aumentando o risco de caminhões carregados ou máquinas pesadas deslizarem, o que pode resultar em acidentes. Além disso, a queda de temperatura pode causar enfermidades nos trabalhadores que estão expostos ao ambiente.

Em situações em que, por razões de segurança, os veículos em uma obra precisam interromper suas atividades devido ao volume elevado de chuva, o tempo de paralisação muitas vezes é descontado do contrato. Por exemplo, se um trator de esteira ficar parado por 63 minutos, a obra pode, conforme estipulado em contrato, não remunerar esse tempo de inatividade causado pela chuva. No entanto, o registro do período de chuva e a determinação do evento climático são atualmente realizados de forma manual. Isso significa que um operador precisa lembrar-se de que o trabalho foi interrompido e saber exatamente o período em que as máquinas ficaram paradas e quais máquinas estavam

no local. Embora seja possível que exista uma estação meteorológica na obra, devido à grande extensão da área de trabalho, a chuva pode afetar ou não as diferentes frentes de serviço, que podem estar muito distantes umas das outras.

O dispositivo descrito neste trabalho tem como função possibilitar a entrega de informações importantes de forma fácil, rápida e de baixo custo. Com os dados disponíveis, a gerência de uma obra, por exemplo, poderá analisar o histórico de chuvas, avaliar o volume médio de precipitação em cada localização e criar um mapa de volumes. Além disso, poderá identificar os dias de chuva, quantificar a quantidade de água que causou a interrupção das atividades e verificar se as condições climáticas para um determinado período poderá impactar o trabalho. A capacidade de realizar todas essas análises agrega valor ao serviço e torna a obra mais eficiente na gestão de suas operações.

Nessa busca por dados importantes de forma fácil e ágil, os dispositivos de Internet das Coisas (IoT) se destacam e estão ganhando cada vez mais espaço no cenário nacional e internacional. Apesar de existirem há apenas duas décadas, já estão presentes em diversas situações do cotidiano (FILHO, 2016). É notório o crescimento no número de empresas que oferecem soluções IoT para monitoramento e controle de dados em indústrias, construção civil, logística de estoque, entre outros setores. A indústria de microcontroladores também tem se expandido, impulsionada pelo custo acessível dos dispositivos e por suas crescentes capacidades de processamento e conectividade, que aumentam a cada novo lançamento.

1.1 Objetivo

O dispositivo desenvolvido tem a função de coletar as informações sobre as condições climáticas em um ponto específico, permitir uma análise precisa dos cenários de chuva que estão ocorrendo na obra. É possível que esteja chovendo em um local enquanto que, a apenas 20 metros de distância, não esteja chovendo. O equipamento de Internet das Coisas produzido tem o objetivo de fornecer os dados de precipitação, juntamente com um painel de visualização para facilitar a tomada de decisões da gestão com base nos dados apresentados.

O dispositivo em questão utiliza recursos baratos e comuns no mercado de automação, deixando a manutenção barata e rápida. Segue a lista de materiais:

- Microcontrolador ESP32
- Sensor de Chuva
- Sensor de Efeito Hall
- Módulo de *Real Time Clock*

- Módulo GPS NEO6M

Tendo como meta capturar a situação do clima, isto é, indicar se está chovendo ou não, determinar o volume de precipitação em milímetros/horas, horário do início e do final da chuva, a duração e a localização.

2 REVISÃO BIBLIOGRÁFICA

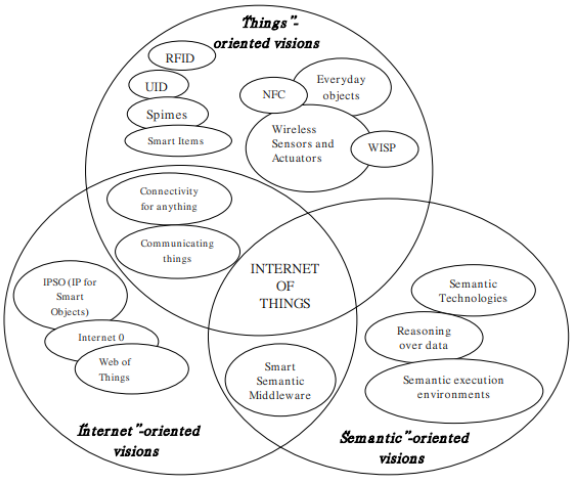
2.1 *Internet of Things*

O termo Internet das Coisas (*IoT*, em inglês *Internet of Things*) ganhou maior relevância em meados dos anos 2000 e refere-se à conexão da Internet com objetos físicos do cotidiano, interligando o mundo virtual ao mundo real. Isso permite que itens físicos sejam controlados remotamente de qualquer lugar do mundo por meio da Internet (MATTERN; FLOERKEMEIER, 2010).

As aplicações para os dispositivos *IoT* são inúmeras, influenciando praticamente todos os aspectos do cotidiano em diversos setores, como indústria, medicina, telecomunicações, construção civil, agricultura, entre outros (ATZORI; IERA; MORABITO, 2010). A versatilidade e praticidade proporcionadas pelos dispositivos *IoT* têm contribuído para o rápido crescimento de sua popularidade. A previsão é que, a partir de 2025, o número de equipamentos conectados ultrapasse 30 bilhões (KATIE, 2024).

O paradigma da Internet das Coisas é o resultado da união de três visões: a visão orientada às coisas, a visão orientada à Internet e a visão orientada à semântica (ATZORI; IERA; MORABITO, 2010). Esta última trata da organização de todos os dados, que devem ter um fluxo livre e espaço adequado para armazenamento. Além disso, os próprios dispositivos *IoT* devem ser endereçados, pois, na Internet do futuro, haverá desafios devido ao alto volume de equipamentos *IoT* que estarão conectados e enviando dados. Na Figura 1, é mostrada uma ilustração da união dos grupos das visões.

Figura 1 – União dos parâmetros *IoT*



Fonte: (ATZORI; IERA; MORABITO, 2010)

A Internet das Coisas não é nomeada como uma tecnologia específica, mas é considerada um conceito. Isso se deve ao fato de que não resulta da evolução de uma tecnologia única, mas sim do agrupamento de diversas tecnologias que foram desenvolvidos até alcançar a capacidade de comunicação com a rede de Internet. Essa integração permite todas as interações necessárias para que um sistema possa ser classificado como uma solução *IoT* (FILHO, 2016).

2.2 Microcontrolador

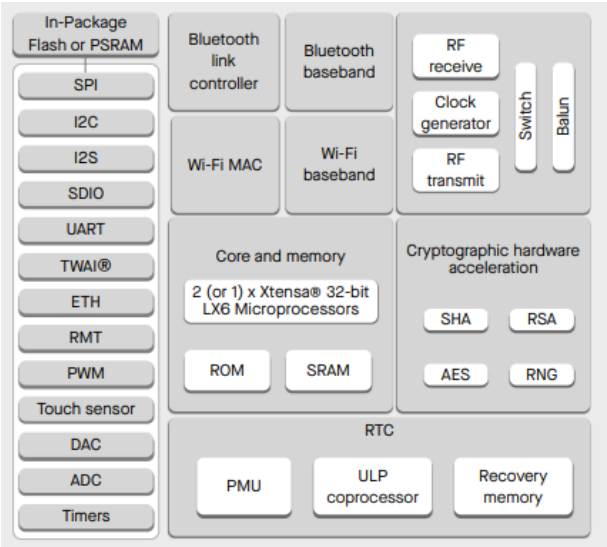
Todo sistema autônomo precisa operar com base na análise dos dados obtidos a partir de seus sensores. Isso significa que a escolha da atividade a ser executada depende da detecção de uma situação específica através de um sinal de um sensor. Para isso, é necessário que haja uma unidade de controle para determinar a origem e o destino das informações, além de uma unidade de processamento para tratar a informação recebida e entregar o dado processado à unidade de controle, que executará a resposta do sistema. Um microcontrolador (µC) possui tanto a unidade de controle quanto a de processamento, além de recursos adicionais de armazenamento e comunicação que evoluíram significativamente nos últimos anos (MAIER; SHARP; VAGAPOV, 2017). Essa evolução aumentou as possibilidades de suas aplicações, tornando-os fundamentais para os sistemas embarcados (SAMIULLAH; IRFAN; RAFIQUE, 2023).

2.2.1 µC ESP32

O microcontrolador da Espressif Systems, o ESP32, é um dispositivo de baixo valor de compra e baixo consumo de energia, com um modo de uso chamado de *Deep Sleep* ou Sono profundo, onde o consumo é extremamente baixo, apenas esperando algum sinal dos sensores conectados para atuação (SYSTEMS, 2024b).É utilizado em várias aplicações como, por exemplo, trabalhos remotos sem conexão e aplicações *IoT*.

A arquitetura do ESP32 possui um microprocessador Xtensa® 32-bit LX6 da empresa Tensilica que foi comprada pela Cadence Design Systems (ROSARIO, 2023). O diagrama do bloco funcional do *System-on-a-Chip* (SoC, sigla em inglês) é ilustrada na Figura 2.

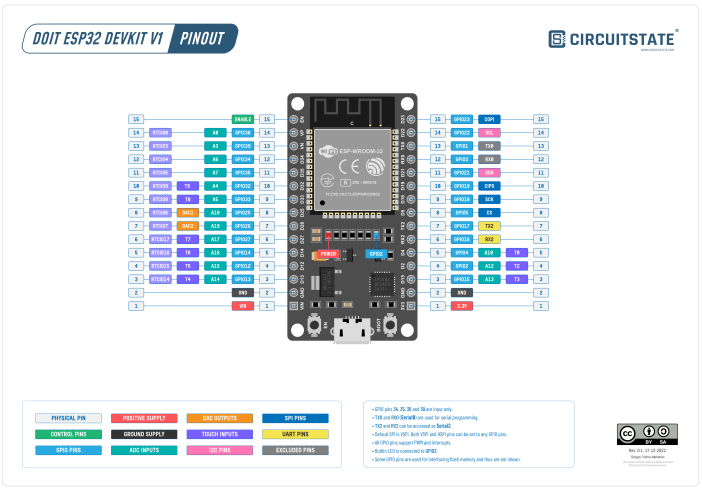
Figura 2 – Diagrama do bloco funcional do ESP32



Fonte: (SYSTEMS, 2024a)

A placa DOIT ESP32 DevKit V1 possui o chip do microcontrolador e 30 pinos para conexões, dos quais 26 são *General Purpose Input/Output* (GPIO). Esses pinos podem ser configurados tanto como entradas quanto saídas e podem operar de acordo com os protocolos de comunicação ou simplesmente como portas digitais. Os outros pinos são destinados à alimentação (5 V e 3,3 V) e há também 2 portas de aterramento (*GROUND*) (CIRCUITSTATE, 2024). Na Figura 3 mostra o detalhe dos pinos da placa.

Figura 3 – Pinagem do DOIT ESP32 DevKit V1



Fonte: (CIRCUITSTATE, 2024)

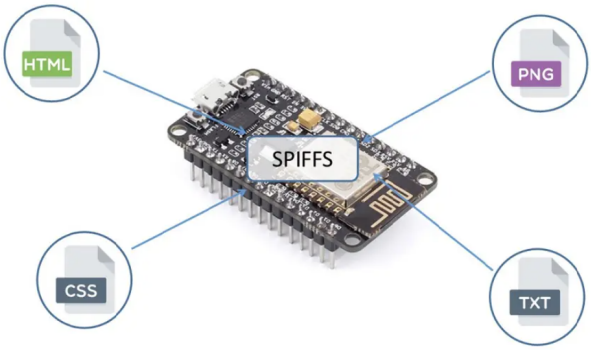
O microcontrolador da Espressif conta com um sistema de *pipeline* de 7 estágios e suporta instruções de 16/24 bits, configurando-se como uma solução que oferece alta densidade de código. Isso significa que, enquanto um estágio do *pipeline* busca uma

instrução, outro estágio decodifica uma instrução previamente recolhida para execução, acelerando o processamento das instruções e a resposta do microcontrolador (SAMIULLAH; IRFAN; RAFIQUE, 2023). Além disso, o sistema do ESP32 inclui um multiplicador e um divisor de 32 bits, um endereço MAC de 40 bits, e um sistema integrado com antenas, balanceamento de sinal de RF, amplificadores, filtros e módulos de gerenciamento de energia. Ele conta com 520 KB de SRAM, 448 KB de ROM, 16 KB de SRAM no RTC interno, 4 MB de Flash, uma frequência máxima de clock de 240 MHz e módulos de comunicação Wi-Fi e Bluetooth (SYSTEMS, 2024a).

Na Figura 2, nota-se que os recursos das portas de entrada e saída estão destacados, como, por exemplo, os conversores analógico-digital e digital-analógico, que são amplamente utilizados e possuem portas específicas para conexão. Entre esses recursos, estão listados os possíveis protocolos de comunicação que já possuem gerenciamento interno, como *Serial Peripheral Interface* (SPI), *Inter-Integrated Circuit* (I2C) e *Universal Asynchronous Receiver Transmitter* (UART). Esses protocolos são os mais populares entre os sensores e podem ser associados a qualquer pino, uma vez que o ESP32 possui um recurso de multiplexador para as portas (SANTOS, 2024).

Outro recurso do ESP32 é a partição da sua memória Flash de 4 MB. Por padrão, o sistema reserva aproximadamente 1,3 MB dessa memória para armazenamento não volátil, o que significa que, se o microcontrolador for desligado por qualquer motivo, os dados salvos nessa partição permanecerão intactos quando o sistema voltar a funcionar. Essa partição é de fácil acesso e é chamada de SPIFFS, permitindo várias operações, como ler, escrever, adicionar, deletar, abrir e fechar arquivos (PIMENTEL, 2024).

Figura 4 – SPIFFS - Variedade de tipos de arquivos possíveis



Fonte: (SYSTEMS, 2024a)

2.3 Sensor de chuva

O sensor de chuva foi selecionado por ser de baixo custo e de fácil instalação. O sensor mostrado na Figura 5 utiliza o chip comparador LM393 da Texas Instruments®.

O LM393 possui um funcionamento simples, recebendo o sinal da placa exposta à chuva, comparando-o com um sinal de referência, e enviando a saída de forma analógica. Além disso, o chip da Texas Instruments possui uma saída digital que resulta em um valor lógico alto ou baixo, dependendo do sinal recebido da placa de chuva pelo comparador, que pode ou não estar com água (Texas Instruments, 2020). No módulo, há um potenciômetro que ajusta a sensibilidade do alerta de chuva, permitindo calibrar o sinal de saída do módulo, aumentando ou diminuindo a sensibilidade.

Figura 5 – Sensor de chuva

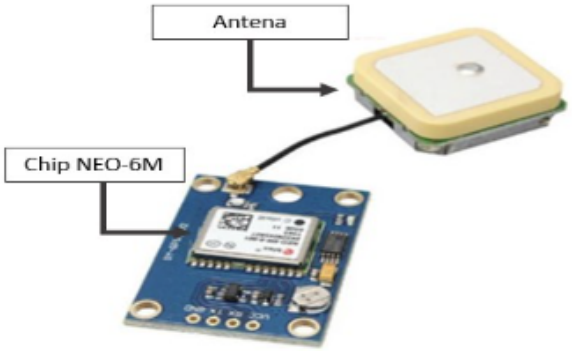
Fonte: Adaptada de (ROBOCORE, 2024)

No módulo do sensor, há 4 pinos que são conectados ao microcontrolador: a fonte de alimentação (3,3 V), o pino de aterramento (*ground*), uma saída digital (D0) e outra analógica (A0). Considerando que o ESP32 possui portas de leitura de sinais analógicos com conversores analógico-digitais (ADC), a conexão com a porta A0 do módulo do sensor de chuva é necessária para que o microcontrolador receba as informações analógicas e as converta em digitais. Sabendo que o ADC do ESP32 é de 12 bits, pode-se obter valores de 0 a 4095 como resultado do nível de água no sensor de chuva.

2.4 Módulo GPS NEO-6M

O módulo GPS NEO-6M é amplamente utilizado por seu baixo custo e boa precisão, sendo baseado no chip NEO-6 da ublox®. Seu tamanho reduzido, opções de memória e baixo custo fazem com que seja ideal para projetos com restrições de espaço e orçamento. Pode ser conectado a uma fonte de 2,7 V a 3,6 V, e seu protocolo de comunicação é o *Universal Asynchronous Receiver Transmitter* (UART) (u-blox, 2011).

Figura 6 – Módulo GPS NEO-6M



Fonte: (STA Eletrônica, 2024)

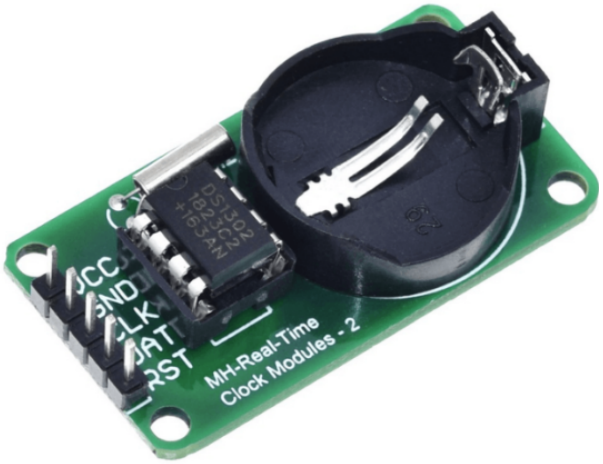
A comunicação UART é suportada pelo ESP32, como mencionado na Seção 2.2 e para a conexão é necessário que o pino *Receiver* (RX) do GPS esteja conectado ao pino *Transmitter* (TX) do microcontrolador. A mesma relação se aplica ao RX do microcontrolador, que deve ser conectado ao TX do módulo GPS. Com essa configuração, os dois dispositivos podem se comunicar de forma *full-duplex*, ou seja, ambos podem enviar e receber informações ao mesmo tempo (SOLUTIONS, 2024).

A comunicação ocorre de forma assíncrona, ou seja, não há sincronização com o relógio interno dos dispositivos. No entanto, para indicar que uma sequência de bits será enviada, é utilizado um bit de *start* para o reconhecimento do envio e, consequentemente, o recebimento dos dados (SOLUTIONS, 2024). Durante o envio dos dados, os bits são primeiramente carregados em paralelo na unidade responsável pela transmissão, e então são adicionados o bit de *start*, o bit de paridade e o bit de *stop*, completando o pacote. O bit de paridade é inserido para checar a integridade da mensagem recebida, enquanto o bit de *stop* sinaliza o fim do envio (DEVICES, 2024).

2.5 Módulo RTC

O módulo RTC (Relógio de Tempo Real) é capaz de contar os segundos, somando-os ao horário anterior e incrementando a cada interação, registrando assim o avanço do tempo. Esse relógio possui uma bateria reserva, o que permite que o módulo continue contando o tempo e ajustando seu calendário interno, mesmo em caso de falha de energia. Na Figura 7, é possível ver a pinagem e o módulo RTC DS1302, que será utilizado no dispositivo.

Figura 7 – Módulo DS1302



Fonte: (DIGITAL, 2024)

Pode-se notar que o módulo possui 5 pinos e um cristal oscilador, cuja frequência é de 32,768 kHz (SEMICONDUCTOR, 1996). Além disso, possui um espaço reservado para a inserção de uma bateria de 3 V, comumente chamada de CR2032. Os pinos do módulo são:

- Alimentação de energia de 2,0 V a 5,5 V (VCC)
- Aterramento (*ground*)
- CLK - Alinhamento do clock do microcontrolador com o do DS1302 para sincronização do envio de dados
- DAT - Canal de envio e recebimento dos dados
- RST - Canal que deve ser setado para o nível lógico alto para leitura ou escrita.

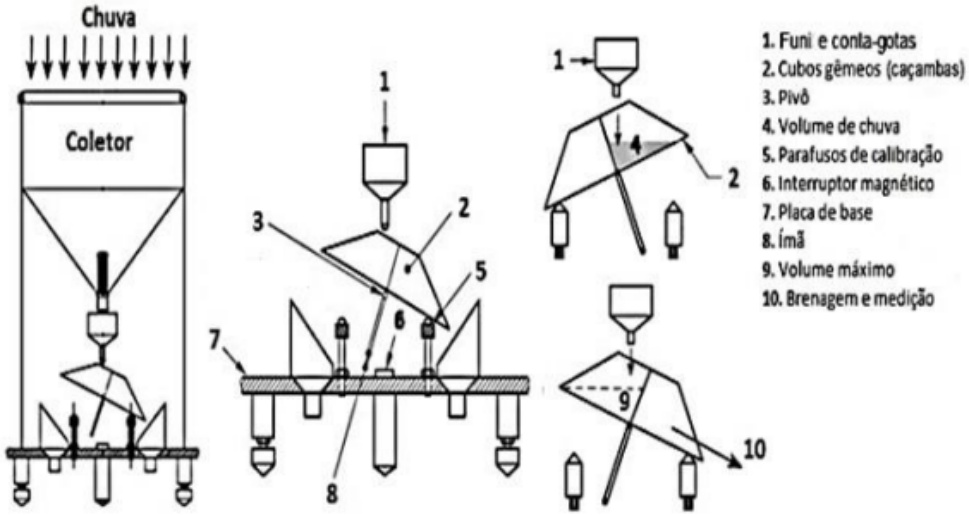
Como o envio e recebimento de dados é feito por um único pino, a comunicação é half-duplex. Tanto o microcontrolador quanto o DS1302 podem enviar dados, mas apenas um de cada vez.

O byte de comando é utilizado sempre que se inicia uma leitura ou escrita nos registradores do DS1302. O bit mais significativo (MSB - bit 7) corresponde ao nível lógico alto para iniciar a comunicação. O bit 6 configura se a troca de informações será com a RAM (nível lógico alto - 3,3 V) ou com o relógio (nível lógico baixo - 0,0 V). Os bits de numeração 5 a 1 determinam em quais registradores a informação será lida ou escrita, e o bit menos significativo (LSB) indica, se estiver em nível alto, que é uma leitura, ou se estiver em nível baixo, que é uma escrita (SEMICONDUCTOR, 1996).

2.6 Pluviômetro

O pluviômetro é um equipamento que mede a quantidade de água precipitada, e a medição é feita em milímetros. Os pluviômetros podem ser classificados como automáticos ou convencionais (MENEZES, 2024). O equipamento automático coleta a água da chuva e envia a quantidade de precipitação em relação ao tempo. Dentro dos automáticos, existem três tipos: o pluviômetro de pesagem, que pesa a quantidade de água precipitada; o equipamento com flutuador, onde uma haste presa a um reservatório analisa a altura da coluna de líquido; e o dispositivo com sistema de básculas, onde um sistema semelhante a uma gangorra coleta a água. Quando o líquido dentro da báscula exerce força suficiente para movimentar o sistemá de báscula, ele vira para o lado oposto, esvaziando-se e começando a coleta na outra báscula. Nas Figuras 8 e 9, são mostrados os equipamentos com o sistema de básculas e o convencional.

Figura 8 – Pluviômetro automático



Fonte: (DIGITAL, 2024)

Figura 9 – Pluviômetro convencional

Fonte: (SPLabor, 2024)

Quando as básculas mudam de posição, o sistema passa por um sensor que detecta a entrada de uma determinada quantidade de água e calcula o acréscimo do líquido capturado.

O sensor responsável por captar a mudança no sistema de básculas é o de efeito Hall, que consegue detectar um campo magnético e informar ao microcontrolador através de um pino digital.

2.7 Banco da dados

O banco de dados é uma forma organizada de armazenamento de informações, normalmente inseridas em sistemas de computadores, que podem ser *data centers* ou grandes servidores para aplicações mais locais. Pode-se pensar que um banco de dados é algo simples, mas é necessária a existência de um sistema de gerenciamento de banco de dados e alguns aplicativos relacionados, tornando-o uma estrutura mais complexa (Oracle, 2024).

Normalmente, os bancos de dados são organizados em várias linhas e colunas, formando tabelas, o que facilita a leitura e a escrita dos dados (Oracle, 2024). A linguagem comumente utilizada para consultar bancos de dados é a Structured Query Language (SQL). No entanto, o SQL não é usado apenas para realizar consultas; seu uso é abrangente, incluindo a modificação do agrupamento de dados, a inserção de dados e a inclusão de níveis de segurança nas tabelas (GARCIA-MOLINA; ULLMAN; WIDOM, 2024).

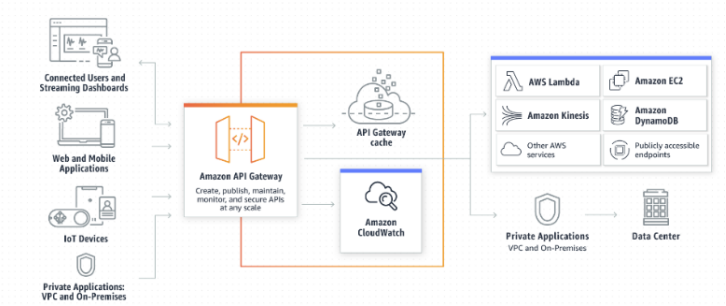
Os primeiros bancos de dados surgiram nos anos 1960, com estruturas mais simplificadas e dados que não se relacionavam entre as tabelas. Na década de 1980, surgiram os bancos de dados relacionais e, logo depois, na década de 1990, foram desenvolvidos os

bancos de dados orientados a objetos. Com o tempo, também surgiram os bancos de dados não estruturados (Oracle, 2024).

2.8 *Application Programming Interface* - API

A *Application Programming Interface* (API) é uma ferramenta para uso entre plataformas que executa, através de protocolos e definições, transferências de informações. Funciona basicamente para acessar informações entre plataformas e pode ser usada para ler e enviar dados em um banco de dados, como mostrado na Figura 10, que ilustra as conexões possíveis com uma API da Amazon©(Amazon Web Services, 2024).

Figura 10 – Diagrama da API da AWS



Fonte: (Amazon Web Services, 2024)

Nas comunicações com APIs, sempre haverá um cliente e um servidor. O servidor é responsável por responder às requisições feitas através da interface, enquanto o cliente é o sistema que realiza a requisição. Existem diferentes maneiras de comunicação. A API que utiliza o *Simple Object Access Protocol* ou API *SOAP* que realiza a troca de mensagens por arquivos do tipo XML. Outra opção é a API *RPC*, que usa o *Remote Procedure Calls*, onde o cliente faz uma chamada e a resposta é enviada de volta ao cliente. A API WebSocket, por sua vez, utiliza serviços da web que empregam a estrutura JSON para comunicação. Finalmente, a API REST é a mais popular e flexível; neste modelo, o cliente envia dados e o servidor executa as rotinas necessárias para retornar a resposta à requisição do cliente (Amazon Web Services, 2024).

3 METODOLOGIA

No desenvolvimento de um projeto de operação autônomo, é desejável que o sistema funcione sem nenhuma intervenção. Para alcançar esse objetivo, é preciso observar vários parâmetros, que vão desde o estudo dos sensores, módulos e atuadores a serem utilizados, até a escrita do *firmware* do controlador ou microcontrolador. Primeiramente, foi escolhido o microcontrolador. Logo após, então para a determinação da linguagem do *firmware* do dispositivo, cuja linguagem escohida foi a C++, Nesta seção, estão descritos os softwares, API, escolha de hardware, uso de memória interna como *buffer* de dados e uso do protocolo HTTP.

3.1 Sistema do RainDetector

Na definição do projeto, foi necessário determinar quais recursos o sistema deveria possuir para atender às funcionalidades que são requisitos. Primeiramente, deve-se considerar que construções extensas podem ter dispositivos localizados em áreas distantes do canteiro central da obra. Nesse contexto, a passagem de cabos pode ser ineficiente e inviável devido a problemas como queda de sinal, ruídos e danos estruturais, como cortes e esmagamentos. Além disso, por questões de boas práticas e agilidade, atualmente, as informações não são mais armazenadas localmente. Em vez disso, os dados são guardados em bancos de dados na nuvem. Portanto, o microcontrolador que gerenciará as informações precisa ter conectividade com a rede Wi-Fi e uma boa capacidade de armazenamento interno não volátil. Assim, mesmo que a conexão Wi-Fi ou a fonte de energia seja interrompida, as informações permanecerão armazenadas sem perda.

O sistema foi projetado para captar o horário inicial e final da chuva, sua intensidade, quantos milímetros foram precipitados e localização de onde a chuva foi detectada.

Os requisitos do sistema é listado a seguir:

- Conexão com a rede Wi-Fi
- Conexão com banco de dados virtual
- Robustez estrutural devido à água e outras impurezas
- Localização no momento da detecção da chuva e no final dela.
- Quantidade de chuva em mililitros
- Horário de inicial da chuva

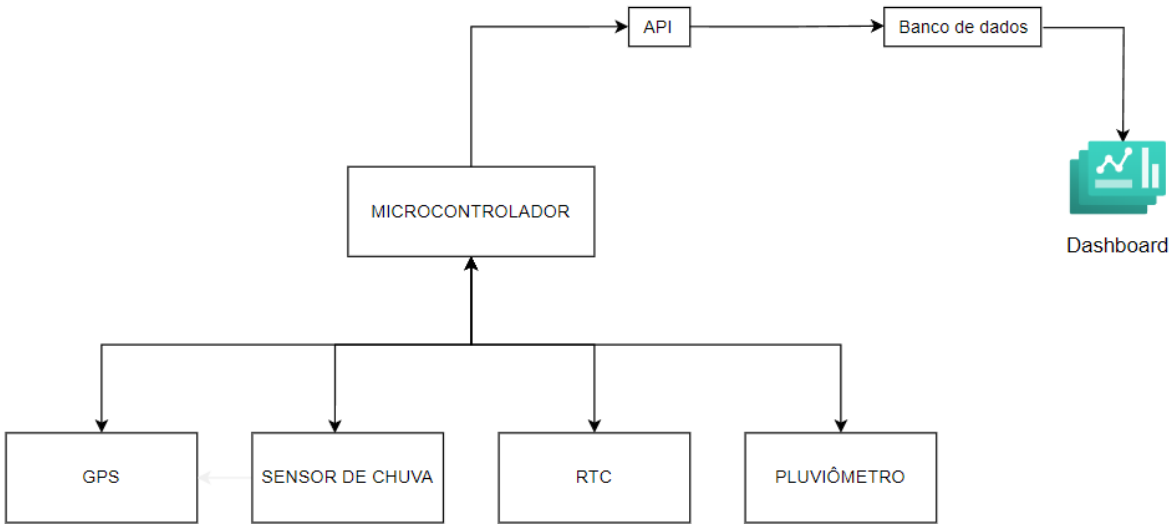
- Horário final da chuva
- Intensidade da chuva

3.2 Design do sistema

Na visão geral, o sistema coleta informações do ambiente em uma localização específica, processa esses dados e os envia para um banco de dados. O dispositivo utiliza uma API para alocar as informações no banco de dados. Quando há conexão com a rede Wi-Fi e essa rede está conectada à Internet, os valores dos parâmetros capturados pelos sensores são enviados em tempo real para armazenamento no banco. Se a conexão com a Internet estiver ausente, os dados serão armazenados na memória interna não volátil do microcontrolador até que a conexão seja restabelecida. Após a reconexão, os dados serão enviados e a memória interna será liberada.

A arquitetura do sistema está descrito na Figura 11 a seguir.

Figura 11 – Desenho da arquitetura do sistema



Fonte: Elaborada pelo autor.

3.3 Hardware

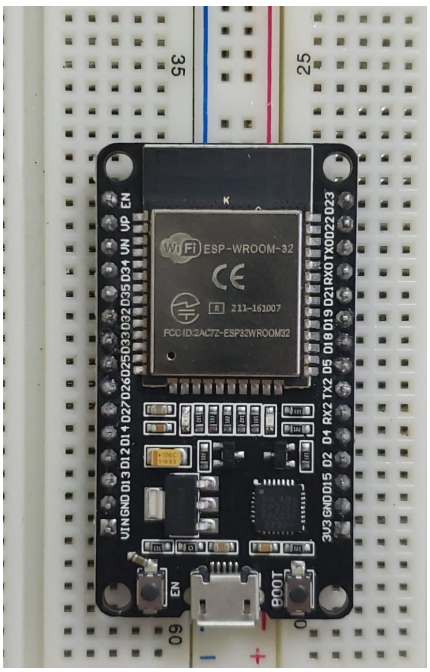
3.3.1 Microcontrolador

Quanto às expectativas relacionadas aos requisitos, a conexão com a rede Wi-Fi é uma característica determinante. Existem dois microcontroladores que se destacam por sua eficiência em operações com internet de baixo custo e baixo consumo: o ESP32 e o Raspberry Pi. Embora o Raspberry Pi tenha vantagens, como maior capacidade de

armazenamento interno, processamento e conexões com periféricos por ser um microcomputador, o ESP32 pode ser uma alternativa viável em algumas aplicações. O ESP32 é mais barato e oferece recursos semelhantes, tornando-se adequado para projetos que não exigem um processamento intensivo (AGNOL, 2018).

Para o projeto RainDetector, que visa manter um custo baixo e não requer grande capacidade de processamento, a escolha do ESP32 se mostra apropriada. O microcontrolador da Espressif oferece diversas conexões com periféricos utilizando diferentes protocolos de comunicação, como SPI, UART e I2C. Isso permite a comunicação do projeto com o GPS através do protocolo UART, com o sensor de chuva por meio da leitura do conversor analógico-digital, com o RTC via protocolo SPI e com o pluviômetro por leitura de um pino digital. Além disso, o ESP32 possui uma boa capacidade de armazenamento interno, com 1,3 MB alocados na configuração padrão de divisão da memória flash. Na Figura 12, é mostrado o ESP32 DevKit V1 utilizado no projeto.

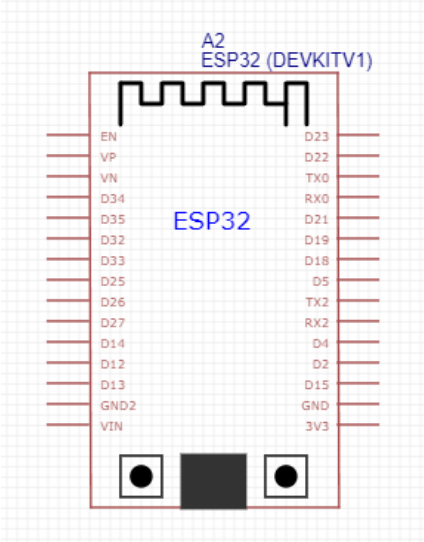
Figura 12 – ESP32 DevKit V1



Fonte: Elaborada pelo autor.

Pode-se ver, na Figura 13, que na ferramenta online EasyEDA foi escolhido um dispositivo para representação do esquemático do ESP32, mantendo os mesmos pinos e dimensões do componente real, como ilustrado na Figura 13.

Figura 13 – ESP32 no EasyEDA



Fonte: Elaborada pelo autor.

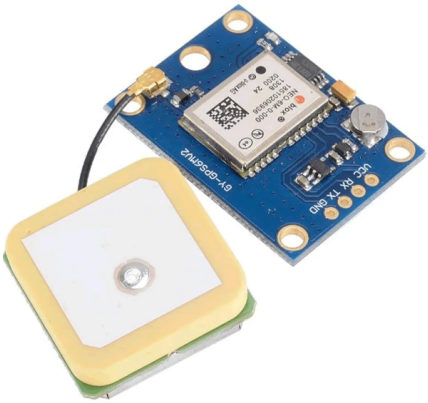
3.3.2 Sensores e módulos

A escolha dos sensores foi baseada nas facilidades de instalação, como o protocolo de comunicação e a pinagem, bem como na programação, considerando a quantidade de memória necessária para alocar o código. Devido ao uso da biblioteca HTTP para comunicação com a API, é importante monitorar a ocupação do espaço na memória para garantir que o programa não exceda a capacidade estabelecida.

3.3.2.1 Global Positioning System

Para o recurso de localização via *Global Positioning System* (GPS), é utilizado o módulo GPS NEO-6MV2 com o chip NEO-6M, produzido pela ublox, e uma antena, conforme mostrado na Figura 14.

Figura 14 – Módulo GPS

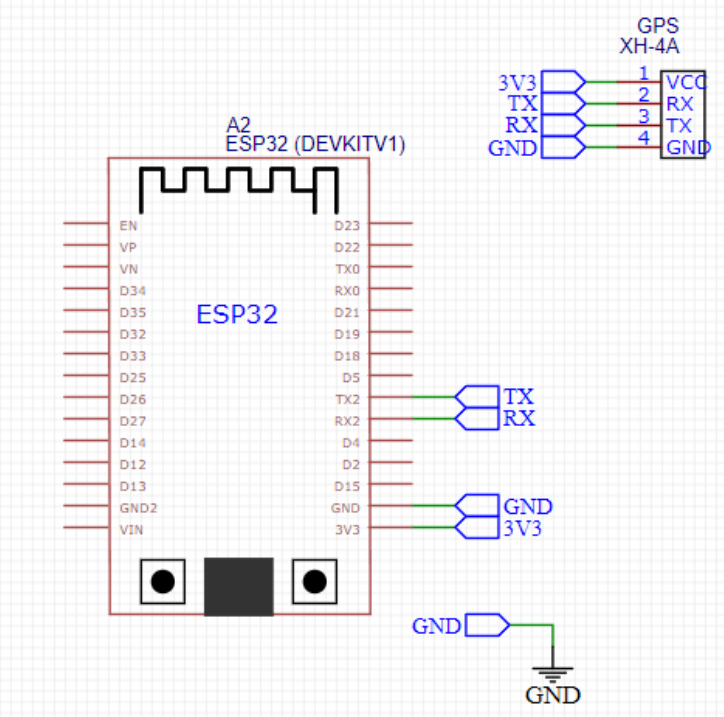


Fonte: (OLIVEIRA, 2022)

O módulo GPS é conectado ao ESP32 pelos pinos RX e TX devido ao seu protocolo de comunicação ser o *Universal Asynchronous Receiver Transmitter* (UART), além de ser alimentado por 3,3 V e estar conectado ao terra do sistema.

Na Figura 15, é ilustrado por meio do EasyEDA a pinagem e conexões do ESP32 com o módulo.

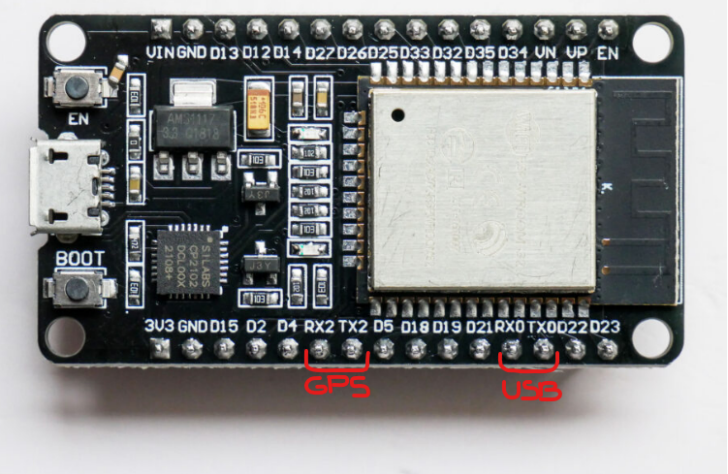
Figura 15 – Conexão do módulo GPS com o ESP32



Fonte: Elaborada pelo autor.

Pode-se notar que não foram utilizadas as portas TX0 e RX0 para a aplicação do GPS, pois tais pinos são utilizados para a comunicação serial pela porta USB do microcontrolador. Dessa forma, se a comunicação serial estiver ativa, por exemplo, durante testes ou manutenção do sistema, pode haver a possibilidade de perda de pacotes de informação devido ao compartilhamento da porta serial. Na Figura 16, estão ilustrados os pinos comentados: as GPIO3 e GPIO1 são as portas RX0 e TX0, respectivamente, usadas para comunicação serial USB, utilizadas para programação e monitoramento serial. As GPIO16 e GPIO17 são conexões disponíveis para o GPS.

Figura 16 – Pinos RX0, TX0, RX2 e TX2 do ESP32



Fonte: (MOHANAN, 2023)

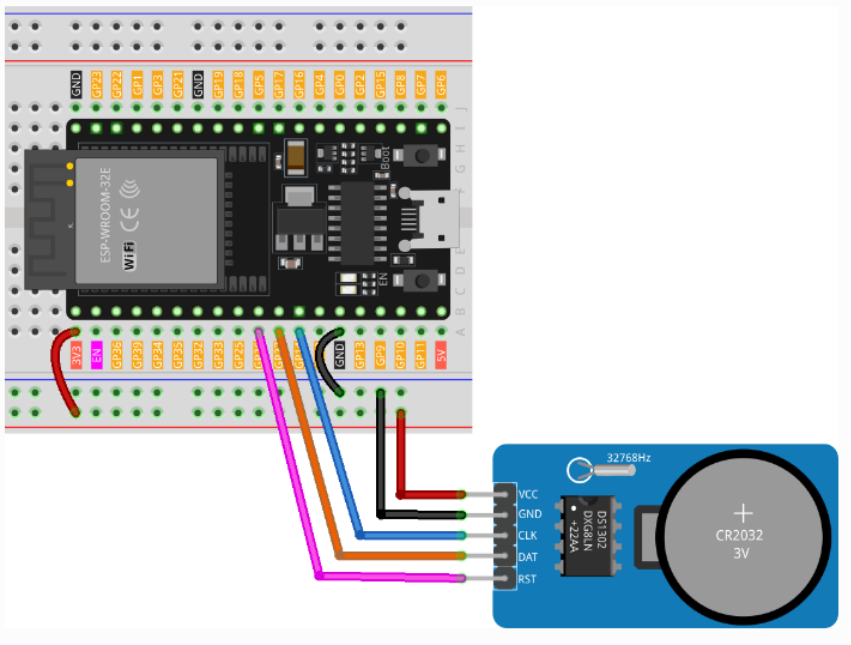
A obra é dividida em acessos, e cada acesso tem sua região mapeada pelo time de geoprocessamento, com resultados que incluem pontos de latitude e longitude. Com essas informações advindas do mapeamento, é possível determinar em qual acesso está chovendo usando as informações de latitude e longitude enviadas pelo módulo GPS, o que é crucial para identificar o ponto exato de onde ocorre a precipitação. Além disso, para garantir a segurança dos dados, há uma redundância de informações proporcionada pelo módulo RTC, que fornece a hora ao controlador e possui uma fonte de alimentação separada para casos de perda de energia. O módulo NEO-6M também fornece a data e a hora exatas de acordo com o fuso horário determinado e pode ser utilizado para atualizar o horário do RTC caso este tenha perdido a contagem devido a algum incidente. Os valores de latitude e longitude também são exibidos no painel de monitoramento do sistema de chuva, permitindo visualizar, através de um mapa, onde está chovendo.

3.3.2.2 *Real Time Clock*

O sistema de captura dos dados de data e hora oferece robustez devido à redundância proporcionada pelos módulos GPS e DS1302, que é o dispositivo de contagem de calendário escolhido para o sistema abordado. O dispositivo de relógio em tempo real é capaz de fornecer dados como data, hora, minutos, segundos, dia da semana, mês e ano, além de identificar variações relacionadas a anos bissextos e alternância dos meses com 28, 30 ou 31 dias. Esse conjunto de informações é essencial para garantir a precisão na associação entre o momento em que a chuva é detectada e o horário em que começou a chover.

O DS1302 se conecta ao ESP32 através de 5 pinos: alimentação de 3,3 V, *GROUND*, *SCLK* (relógio serial para sincronizar o movimento de dados) e *RST* (utilizado para resetar o horário do sistema do módulo). A configuração e montagem foram feitas em uma placa de prototipagem, como mostrado na Figura 17.

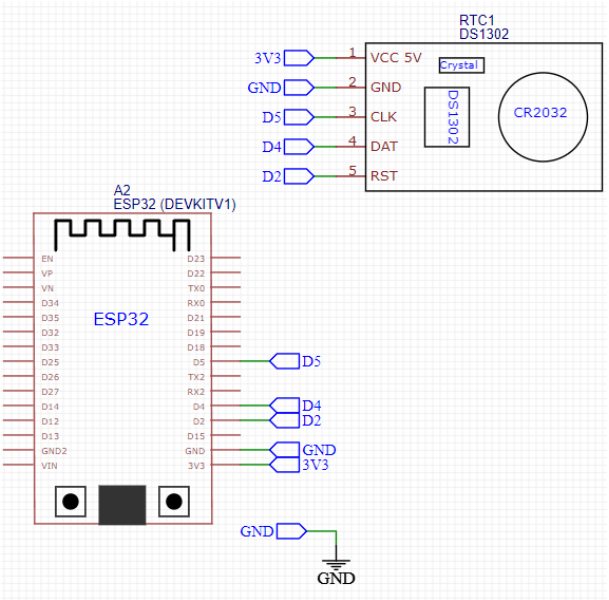
Figura 17 – Montagem do DS1302 com o ESP32



Fonte: (SUNFOUNDER, n.d.)

Para a produção da placa de circuito impressa (PCB), foi produzida um esquemático das conexões no EasyEDA, como mostrado na Figura 18.

Figura 18 – Esquemático do DS1302 com o ESP32



Fonte: Elaborada pelo autor.

3.3.2.3 Pluviômetro

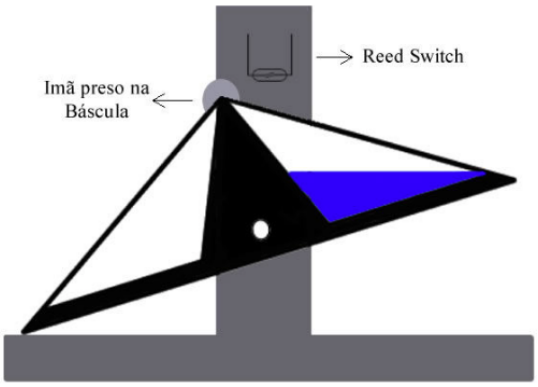
A medição da quantidade de milímetros de precipitação é crucial para determinar a intensidade da chuva e, assim, avaliar se a operação das máquinas deve ser interrompida,

o que pode levar à parada da produção da obra.

Enquanto o pluviômetro convencional exige que o usuário vá até o dispositivo para verificar a quantidade de precipitação, o dispositivo desenvolvido para o projeto utiliza um sistema de medição eletrônico. Este sistema eletrônico é mais eficiente, permitindo a medição da quantidade de milímetros de chuva sem necessidade de inspeção manual.

Um exemplo do sistema de báscula utilizado para a medição da quantidade de milímetros de chuva pode ser visto na Figura 19.

Figura 19 – Sistema de báscula para pluviômetro



Fonte: (STRAUB, 2018)

O funcionamento do aparelho é simples: a báscula, quando cheia, tende a inclinar para o lado mais pesado, isto é, o lado com água, funcionando como uma gangorra. Na sua estrutura, há um ímã que se movimenta junto à báscula e passa por um sensor que detecta seu campo magnético, enviando a informação para o microcontrolador. Conhecendo a quantidade de milímetros da coluna de água que a báscula acumula, é possível calcular o somatório dos movimentos e determinar a medida da precipitação em milímetros.

Na Figura 19, foi utilizado um interruptor magnético (*Reed Switch*), mas neste projeto, o sensor de efeito Hall foi escolhido para detectar a passagem do ímã. Embora o funcionamento seja semelhante, no caso do sensor de efeito Hall, quando o campo magnético é detectado, o circuito é descontinuado e não há corrente. O microcontrolador então identifica a mudança e soma a quantidade ao acumulado de água de chuva calculado previamente.

Para calcular a quantidade de água em um pluviômetro com base nas dimensões do recipiente, pode-se usar a fórmula do volume de um paralelepípedo. Suponha que a caixa tenha comprimento de 20 cm, largura de 10 cm e altura de 5 cm. O volume de água quando a altura da água é, por exemplo, 5 cm, o volume de água é dado por:

$$\begin{aligned} \text{Volume} &= \text{Comprimento} \times \text{Largura} \times \text{Altura} \\ \text{Volume} &= 20\,\text{cm} \times 10\,\text{cm} \times 5\,\text{cm} \\ \text{Volume} &= 1000\,\text{cm}^3 \\ &\approx 1000\,\text{ml ou } 1\,\text{litro de água} \end{aligned} \tag{3.1}$$

Pode-se fazer o caminho inverso, já que o sistema conhece o valor de mililitros de água recebidos da chuva. Para determinar a altura da coluna de água em um pluviômetro com base na quantidade de água acumulada, segue-se o processo:

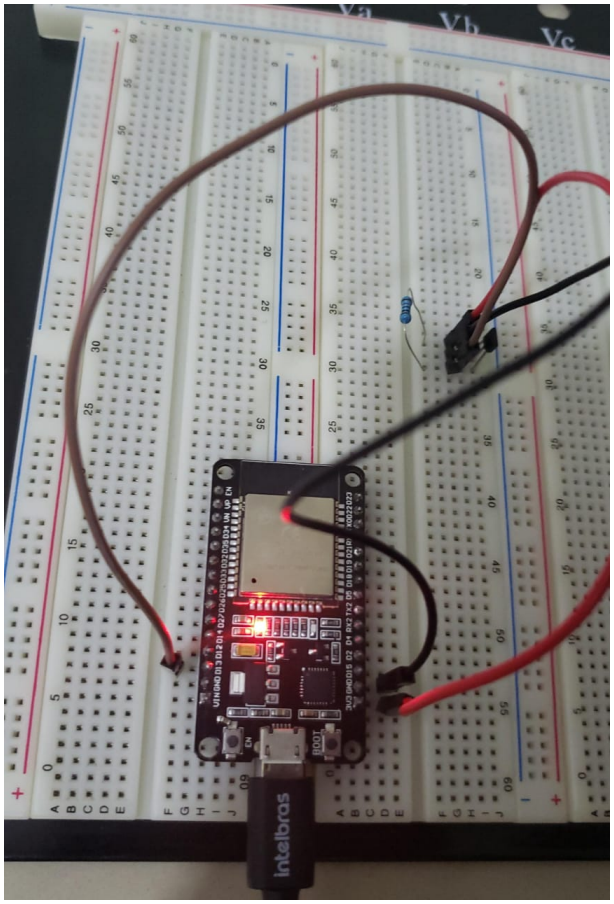
$$\begin{aligned} \text{Volume} &= \text{Comprimento} \times \text{Largura} \times \text{Altura} \\ 1000\,\text{ml} &= 20\,\text{cm} \times 10\,\text{cm} \times x\,\text{cm} \\ x &= \frac{1000}{20 \times 10} \\ x &= \frac{1000}{200} = 5\,\text{cm} \\ &= 5\,\text{cm} \times 10\frac{\text{mm}}{\text{cm}} = 50\,\text{mm de precipitação de água} \end{aligned} \tag{3.2}$$

Para converter é necessário conhecer as dimensões do equipamento que está capturando a água. No projeto, tem-se um equipamento retangular de 20 centímetros de comprimento por 10 centímetros de largura e sabe-se que a báscula acumula 8 mililitros por cada movimento. Com isso, tem-se que:

$$\begin{aligned} \text{Volume} &= \text{Comprimento} \times \text{Largura} \times \text{Altura} \\ 8\,\text{ml} &= 20\,\text{cm} \times 10\,\text{cm} \times x\,\text{cm} \\ x &= \frac{8}{20 \times 10} \\ x &= \frac{8}{200} = 0,04\,\text{cm} \\ &= 0,04\,\text{cm} \times 10\frac{\text{mm}}{\text{cm}} = 0,4\,\text{mm de precipitação de água} \end{aligned} \tag{3.3}$$

Isso significa que a cada movimento da báscula, ocorre um acréscimo de 0,4 mm na coluna de água acumulada. O sensor de efeito Hall escolhido foi conectado ao microcontrolador por meio de uma porta digital configurada com um resistor de *pull-up* externo de 10 k. Essa configuração foi necessária porque a porta GPIO13, selecionada para receber o sinal do pluviômetro, não possui o recurso de *pull-up* interno. Na Figura 20, é ilustrado o resistor de 10k nos testes com o sensor.

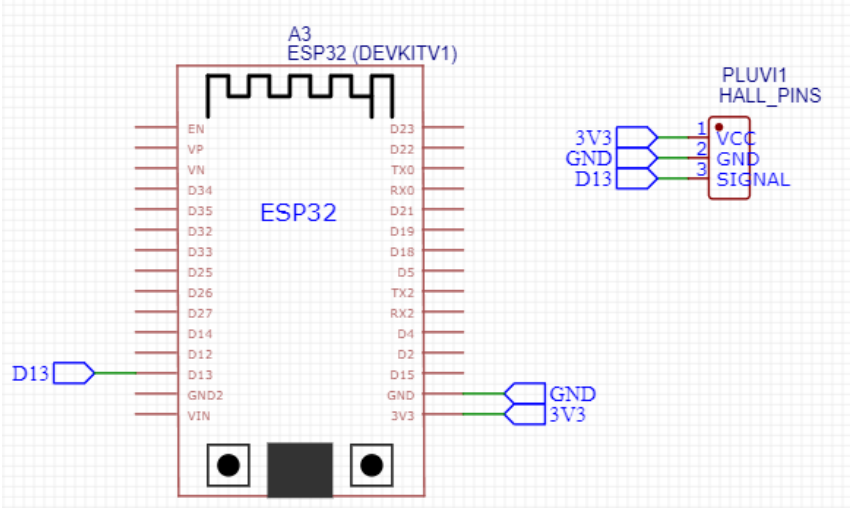
Figura 20 – Teste do sensor de efeito Hall com resistor de 10k



Fonte: Elaborada pelo autor.

As conexões com o ESP32 são mostradas na Figura 21.

Figura 21 – Sistema de báscula para pluviômetro no EasyEDA



Fonte: Elaborada pelo autor.

3.4 Estrutura protótipo

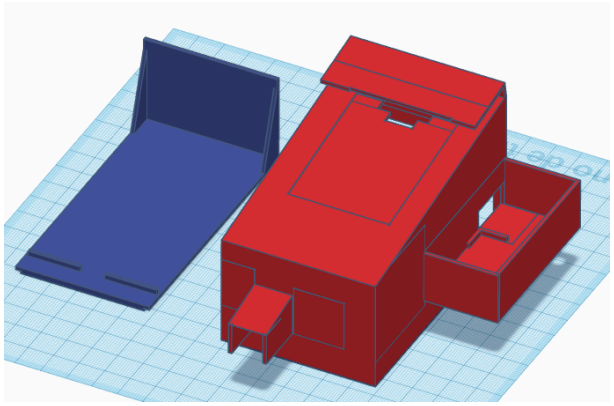
Existem duas estruturas principais: uma estrutura de proteção para o microcontrolador, o sensor de chuva, o RTC e o GPS, e outra estrutura destinada ao sistema do pluviômetro eletrônico.

3.4.1 Caixa de proteção do ESP32

Como o dispositivo é um detector de chuva, deve estar exposto ao ambiente externo. Embora o sensor de chuva possa receber os pingos de água, os demais componentes não devem ficar molhados para evitar danos como oxidação ou condutividade indesejada, que poderiam levar a problemas como curto-circuitos.

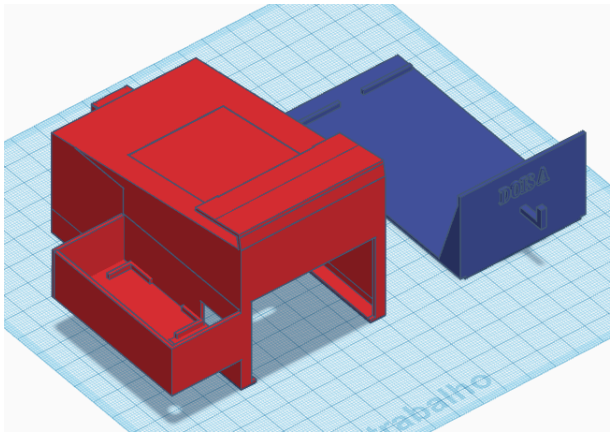
Os desenhos em 3D foram elaborados utilizando a ferramenta online gratuita AUTODESK Tinkercad. Nas Figuras 22 e 23, é possível observar a ilustração da caixa de proteção para a placa de circuito impresso, o sensor de chuva e o GPS.

Figura 22 – Caixa de proteção: visão frontal no Tinkercad



Fonte: Elaborada pelo autor.

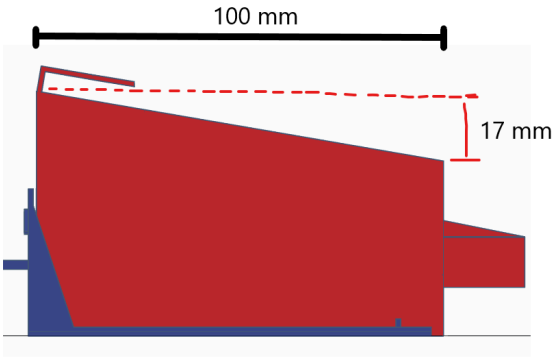
Figura 23 – Caixa de proteção: visão traseira no Tinkercad



Fonte: Elaborada pelo autor.

Além disso, para evitar o acúmulo de água na superfície do sensor de chuva e dispensar a necessidade de um dispositivo limpador adicional para detectar a parada da chuva, foi projetada uma superfície inclinada. A inclinação permite que, devido ao efeito da gravidade e à força do vento, a água escorra naturalmente pelo sensor, saindo do seu topo. Assim, o valor da tensão recebida pelo microcontrolador varia, possibilitando a detecção da interrupção da chuva.

Figura 24 – Desnível de inclinação



Fonte: Elaborada pelo autor.

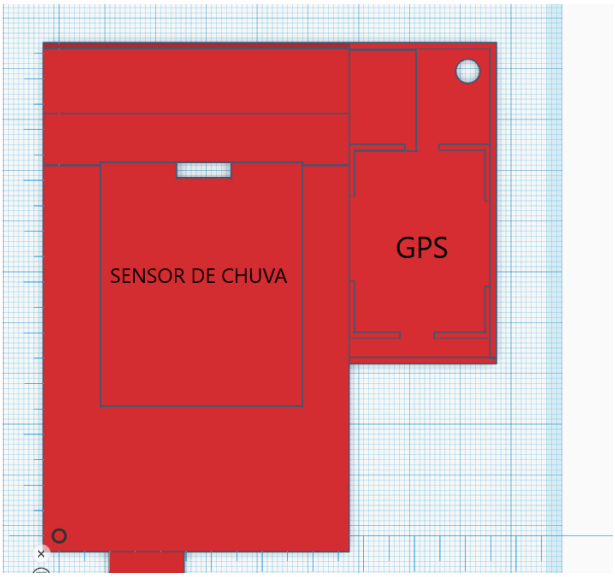
Na Figura 24, nota-se a inclinação da caixa para melhorar o escoamento da água. Para determinar o ângulo de inclinação da superfície, dado a diferença de altura e o comprimento da caixa, utiliza-se a fórmula para o cálculo do ângulo de inclinação:

$$\theta = \tan^{-1}\left(\frac{\text{cateto oposto}}{\text{cateto adjacente}}\right) = \frac{17}{100} = 9,65° \tag{3.4}$$

A chuva pode vir de todas as direções de acordo com a força do vento, então o ângulo de inclinação não pode ser muito alto para não impedir a captação de água em determinados ângulos.

Na estrutura de proteção, há um local específico para o encaixe do sensor de chuva e do módulo GPS. O sensor de chuva está posicionado no topo da caixa, enquanto o módulo GPS está localizado na lateral, como ilustrado na Figura 25.

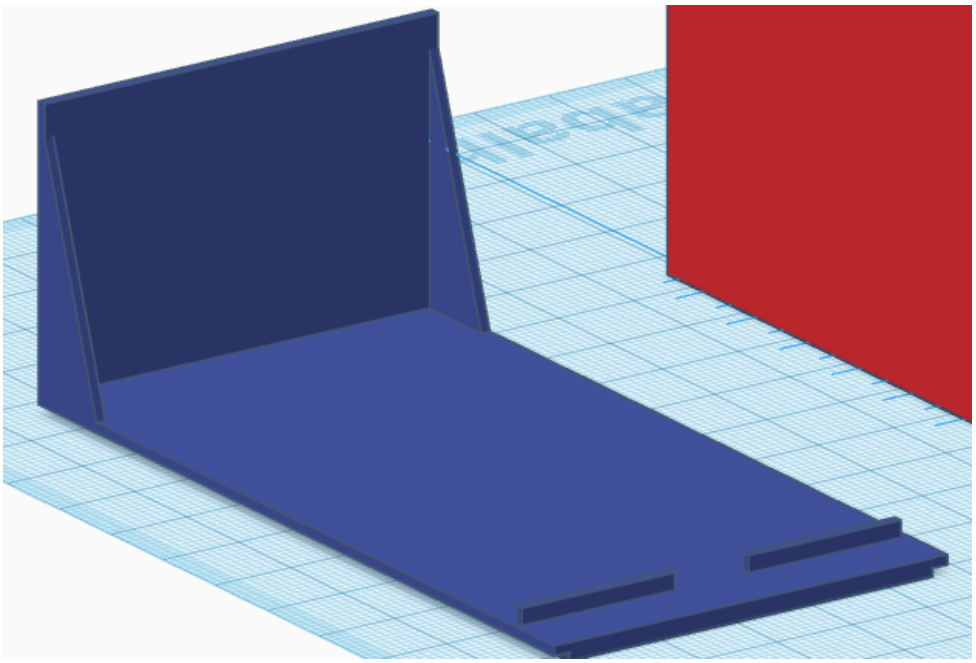
Figura 25 – Localização e encaixe do sensor de chuva e módulo GPS



Fonte: Elaborada pelo autor.

Na base de encaixe da placa de circuito impresso, conforme mostrado na Figura 26, observa-se a presença de pequenas divisórias projetadas para posicionar a placa de forma estável. Essas divisórias têm a finalidade de minimizar o movimento da placa dentro da estrutura, prevenindo possíveis problemas com cabos soltos e danos aos equipamentos.

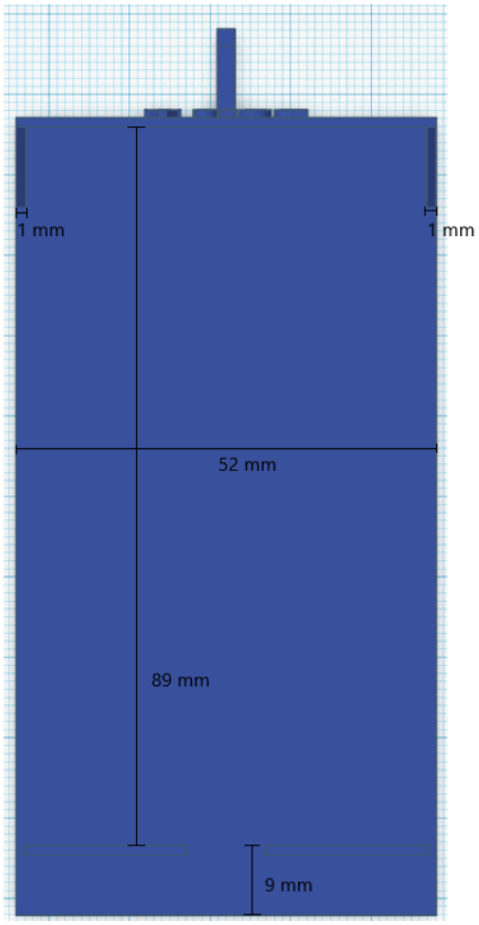
Figura 26 – Base para encaixe da placa de circuito impresso



Fonte: Elaborada pelo autor.

As medidas da base são similares com as medidas da PCB cujas medidas são 89 mm de comprimento e 50 mm de largura. Como mostra a Figura 27.

Figura 27 – Medidas da base

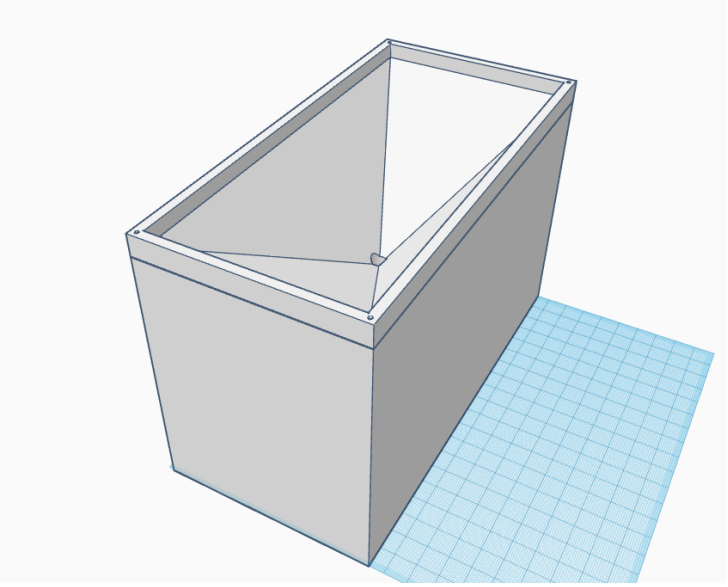


Fonte: Elaborada pelo autor.

3.4.2 Pluviômetro eletrônico

Na medição da quantidade de água precipitada, considerou-se a utilização de um pluviômetro. Contudo, para garantir a visualização do volume de água em um painel que obtém suas informações de um banco de dados, optou-se por um pluviômetro eletrônico, ao invés de um modelo convencional, pois o modelo eletrônico em questão usa um sensor magnético para reconhecer a movimentação do sistema de básculas (Figura 19) e envia o sinal elétrico para que o sistema aumente a quantidade de milímetros de chuva devido ao movimento das básculas, sendo o microcontrolador responsável pelo envio das informações do pluviômetro conectado. Devido ao elevado custo de um pluviômetro eletrônico, foi desenvolvido um projeto CAD para sua fabricação por impressão 3D, assim como o caso da caixa de proteção do microcontrolador. O projeto é composto por um recipiente retangular com as dimensões de 200 mm de comprimento, 100 mm de largura e 130 mm de altura. O desenho 3D está ilustrado na Figura 28.

Figura 28 – Desenho 3D do pluviômetro

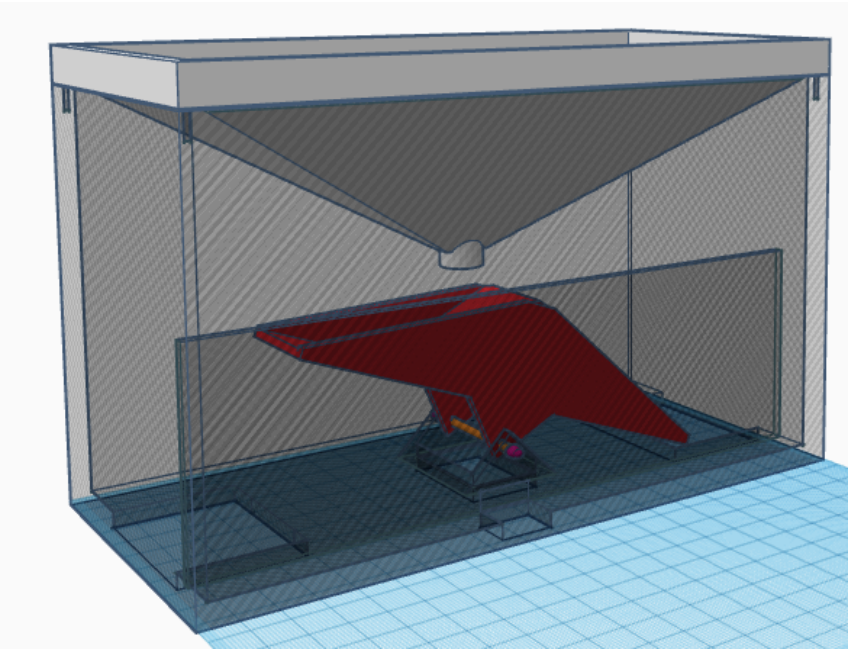


Fonte: Elaborada pelo autor.

O pluviômetro é composto por duas peças principais: a parte superior, que inclui o funil, e a base, que abriga o sistema de básculas.

No topo do pluviômetro, foi projetado um funil para direcionar a água em direção ao sistema de básculas. Esta característica é evidenciada na Figura 29.

Figura 29 – Parte interior do pluviômetro



Fonte: Elaborada pelo autor.

O pluviômetro eletrônico é composto pela estrutura mecânica das básculas, que

acumulam o líquido da chuva, e pelo sistema elétrico, que calcula a quantidade de milímetros coletados. Cada báscula despeja a água acumulada quando o peso do líquido exerce uma força suficiente para gerar um torque maior do que o peso da báscula, fazendo com que ela se mova.

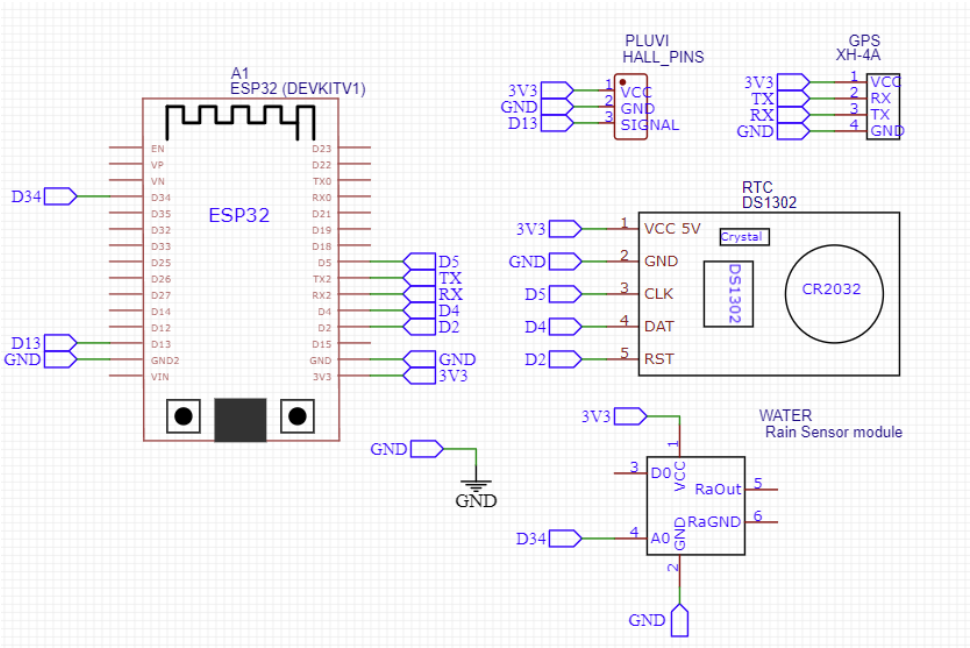
No sistema, o sensor de efeito Hall é instalado na parede lateral do mecanismo de básculas. Esse sensor detecta o ímã acoplado ao centro do sistema de básculas e registrado pelo sensor. A quantidade de milímetros de água é convertida a partir do volume de mililitros detectado. A contagem de líquido é realizada somente quando o sensor de chuva detecta água. Caso contrário, movimentos não são registrados para evitar contagens errôneas durante o transporte ou manuseio do dispositivo.

3.5 Projeto eletrônico do dispositivo

3.5.1 Projeto PCB

Como parte do protótipo, com o objetivo de organizar os componentes e facilitar a montagem do circuito, foi criada uma placa de circuito impresso (PCB). A confecção da PCB foi realizada utilizando a plataforma online EasyEDA, que permitiu tanto a elaboração do esquemático quanto a criação do design da placa. O circuito completo está ilustrado na Figura 30.

Figura 30 – Circuito do dispositivo no EasyEDA

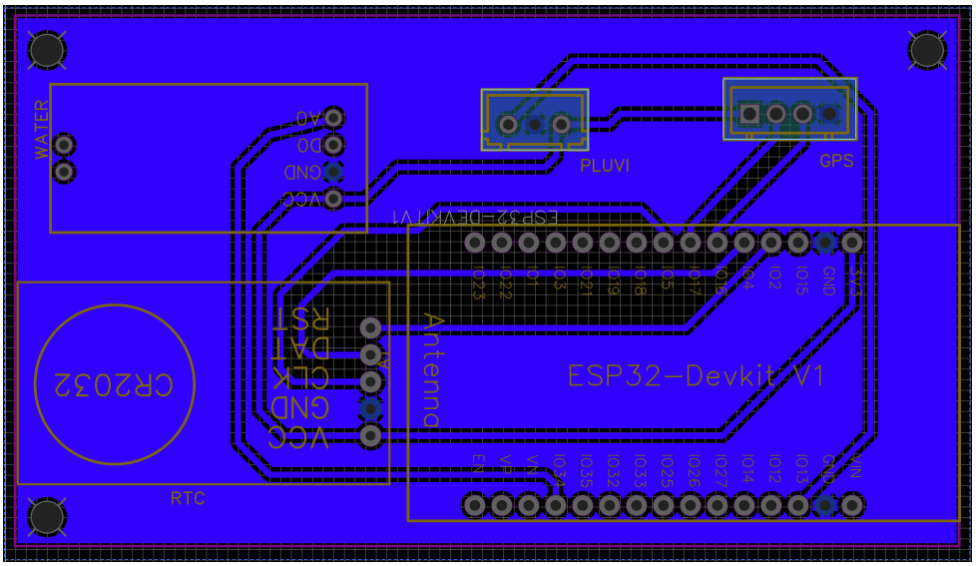


Fonte: Elaborada pelo autor.

A placa possui 89 mm de comprimento e 50 mm de largura. A PCB ficou montada

como mostra a Figura 32.

Figura 31 – Placa de circuito impresso



Fonte: Elaborada pelo autor.

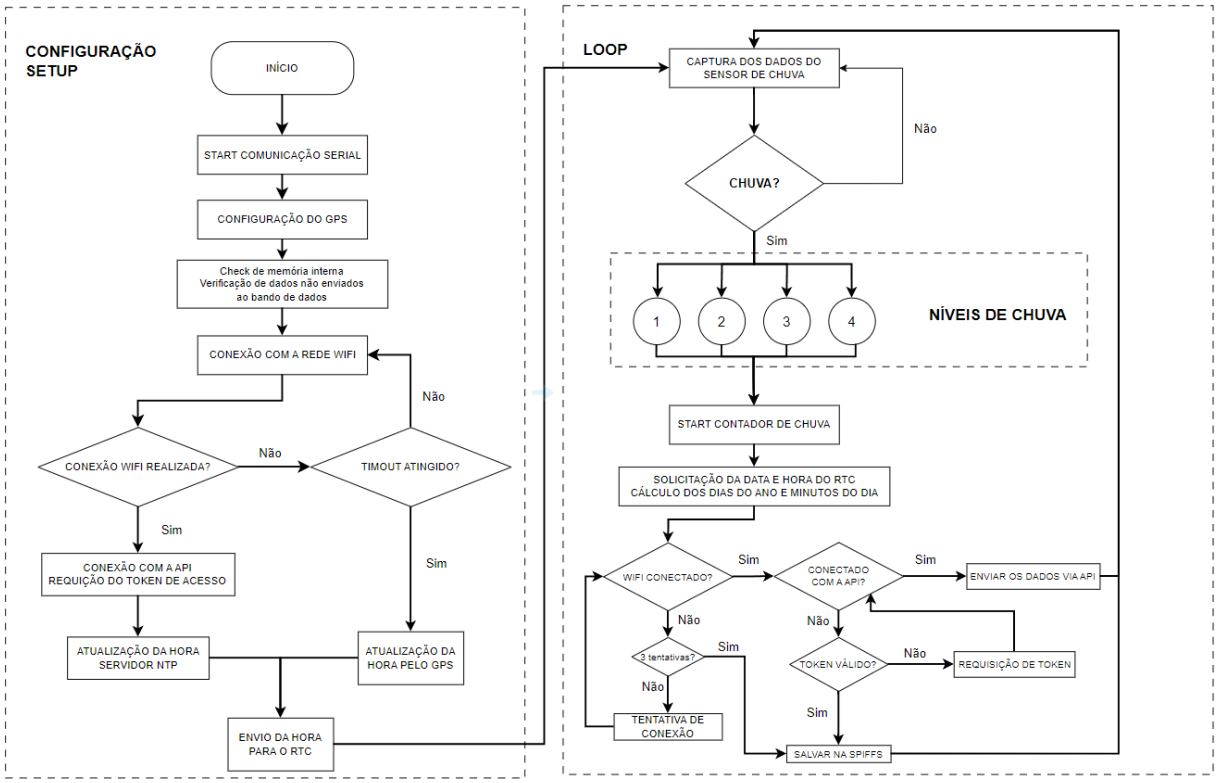
Nota-se que os pinos destacados como “PLUVI” são aqueles destinados ao sensor de efeito hall, localizado na estrutura do pluviômetro, enquanto os pinos “GPS” são destinados ao módulo GPS. O módulo do sensor de chuva e o RTC estão integrados na própria placa para garantir melhor estabilidade e conexão, além de não necessitarem de exposição externa e não apresentarem comprimentos extensos.

3.5.2 *Firmware*

O *Firmware* do microcontrolador da Espressif foi desenvolvido utilizando o Arduino IDE na versão 2.3.3, que suporta as linguagens de programação C e C++. O desenvolvimento foi realizado em C++, e o programa também funciona como compilador, permitindo que o microcontrolador receba e execute as instruções internamente em seu sistema. Para facilitar o entendimento do fluxo do algoritmo, elaborou-se um fluxograma que ilustra o processo de execução do código.

O código é dividido em dois blocos principais: o *setup* (configuração) e o *loop* (laço). O bloco *setup* é responsável por configurar todos os valores iniciais das variáveis globais do sistema, além de inicializar os sensores e módulos, bem como suas respectivas configurações. Por outro lado, o bloco *loop* executa continuamente as rotinas de operação do dispositivo, garantindo o funcionamento contínuo do sistema.

Figura 32 – Fluxograma da lógica do sistema



Fonte: Elaborada pelo autor.

A plataforma Arduino IDE oferece suporte a um vasto número de bibliotecas e também permite a inserção de outras através de arquivos no formato ZIP. Isso facilita o uso de módulos complexos. No código, foram utilizadas oito bibliotecas, que são listadas a seguir:

- SPIFFS.h - Biblioteca encarregada da comunicação com a divisão SPIFFS da memória Flash do ESP32;
- WiFi.h - Responsável pela comunicação com o módulo Wifi do microcontrolador;
- ThreeWire.h - Responsável pela comunicação com o DS1302 que se comunica de forma síncrona com o ESP32 através de 3 fios: CLK; DAT; RST.
- time.h - Responsável pela atualização do horário ao ligar o microcontrolador, caso haja internet.
- HTTPClient.h - Responsável pela comunicação web fazendo requisições HTTP como GET, POST, PUT, DELETE, entre outros. - Usada para comunicação com a API.
- RtcDS1302.h - Usada para comunicação entre o ESP32 e o DS1302, além da biblioteca ThreeWire.

- ArduinoJson.h - Responsável por facilitar a construção das mensagens de comunicação com a API.
- TinyGPS++.h - Responsável pela comunicação com o módulo GPS NEO 6M.

3.5.2.1 Configuração do dispositivo

Sabe-se que, antes de iniciar as rotinas de procedimentos, é necessário realizar as configurações e calibrações dos equipamentos do sistema. Para isso, no Arduino IDE, utiliza-se a função *setup()*. Nessa função, são configuradas as comunicações e os parâmetros iniciais dos sensores e módulos.

Primeiramente, configura-se a comunicação com a porta serial do sistema para conectar ao Micro USB, caso o dispositivo seja conectado a um PC para inspeção ou manutenção. A comunicação serial é iniciada com uma taxa de transmissão (*baud rate*) de 115200. Essa taxa é escolhida com base na capacidade do hardware, na probabilidade de criação de erros de comunicação e nas necessidades do programador. No caso do ESP32, a velocidade declarada é suportada sem causar erros de comunicação.

Em seguida, são feitas as configurações específicas para o módulo GPS, a conexão com a rede Wi-Fi, a sincronização com o servidor *Network Time Protocol* (NTP) para atualização da hora, caso haja conexão com a internet, e a configuração do módulo *Real-Time Clock* (RTC) com a hora atualizada.

Na Configuração do GPS, é realizada a tentativa de espera de 2 minutos por um sinal de satélite, caso não encontrado, o sistema prossegue sem a validação e no decorrer é checada a conexão com algum satélite novamente.

A seguir, há uma demonstração do algoritmo, mas não é a mesma sintaxe da linguagem de programação C++, apenas a lógica é igual.

Algoritmo 1: Configuração do GPS e busca por satélites

```
Data: Variáveis e início da comunicação e configuração com o GPS
HardwareSerial gpsSerial(2);
TinyGPSPlus gps;
short int satelites = 0;
int time_out = millis();
while satelites < 2 and (millis() - time_out) < 120000 do
    if gpsSerial.available() > 0 then
        gps.encode(gpsSerial.read());
        satelites = gps.satellites.value();
        Serial.print("Satélites encontrados: ");
        Serial.println(satelites);
    end
    else
        Serial.println("Nenhum dado disponível no GPS");
        break;
    end
    delay(1000);
end
```

A velocidade padrão de comunicação do módulo GPS é de 9600 bps. Como as unidades de controle da comunicação serial são distintas, não é necessário que a taxa de transferência do GPS seja igual à da comunicação serial da porta USB. Assim, a comunicação serial do GPS pode operar a 9600 bps sem conflitos com outras interações seriais no sistema.

No código, define-se que a unidade de comunicação utilizada será a 2. No ESP32, existem três unidades de comunicação serial disponíveis, permitindo a configuração e uso de múltiplas portas seriais de forma independente.

Em seguida, realiza-se a configuração da memória interna do microcontrolador para possibilitar a leitura e escrita de dados. Isso inclui a verificação de dados que ainda não foram enviados para a nuvem. Essa etapa é essencial para garantir que os dados capturados pelo sistema sejam armazenados corretamente e que, em caso de perda de conexão com a internet, as informações possam ser preservadas até que a comunicação seja restabelecida e os dados possam ser enviados.

Algoritmo 2: Manipulação de Arquivos na SPIFFS

```
   Data: Inicializa e verifica a SPIFFS
 1 if !SPIFFS.begin(true) then
 2 |   Serial.println("Erro na montagem da SPIFFS");
 3 end
 4 else
 5 |   Serial.println("Memória preparada para manipulação");
 6 |   if !SPIFFS.exists(Arquivo_SPIFFS) then
 7 |   |   Serial.println("O arquivo não existe");
 8 |   end
 9 |   else
10 |   |   File file = SPIFFS.open(Arquivo_SPIFFS, "r");
11 |   |   if !file then
12 |   |   |   Serial.println("Falha ao abrir o arquivo");
13 |   |   end
14 |   |   else
15 |   |   |   while file.available() do
16 |   |   |   |   String line = file.readStringUntil("");
17 |   |   |   |   dados_buffer++;
18 |   |   |   end
19 |   |   |   Serial.println(dados_buffer);
20 |   |   |   file.close();
21 |   |   end
22 |   end
23 end
```

Após a configuração do SPIFFS, realiza-se o teste de conexão com a rede Wi-Fi, utilizando as credenciais de rede preestabelecidas nas variáveis *ssid* e *password*. O sistema dispõe de um tempo limite de 1 minuto para estabelecer a conexão. Caso a conexão não seja estabelecida dentro desse período, o sistema continua operando sem interrupções, mas não se conecta ao servidor NTP para atualização da data e hora, nem requisita o *token* de acesso necessário para o envio de informações à API.

Se a conexão com a rede Wi-Fi for bem-sucedida, o sistema realiza a consulta ao servidor de data e hora para obter a atualização necessária. O *token* de acesso à API é então requisitado, permitindo que o sistema atualize o RTC e confirme a precisão da hora no sistema.

Durante a configuração, a API é utilizada para retornar o *token* de acesso, que é essencial para salvar os dados do dispositivo no banco de dados. A requisição é feita usando um login e senha preestabelecidos.

O módulo RTC será atualizado com a data e hora corretas, se o módulo GPS estiver conectado a um satélite ou se for possível a conexão com o servidor NTP. Para mais detalhes sobre o código, consulte o Anexo A (5).

3.5.2.2 Operações de rotina

Após a conclusão da configuração, o sistema inicia suas rotinas operacionais. Ele começa monitorando o sensor de chuva e, com base nas leituras obtidas, procede com a análise de outros fatores como o pluviômetro, o RTC e o GPS. Esses dados são correlacionados ao período de chuva para fornecer informações precisas sobre a intensidade e a localização da precipitação.

NNo laço operacional, o sistema recebe o valor do Conversor Analógico-Digital (ADC), que converte sinais analógicos em digitais, conforme a resolução especificada pelo fabricante do microcontrolador. No caso do ESP32, a resolução do ADC é de 12 bits, o que implica que o valor máximo que o ADC pode produzir é 4095, conforme descrito na seção 2.

Quando o valor do ADC varia, isso indica a presença de água no sensor. Se o valor digital for maior que 4000, o sistema interpreta que não há chuva. No entanto, se o valor estiver abaixo desse limite, o dispositivo considera que está chovendo e inicia a gravação dos dados, marcando o início da chuva e continuando a medir para monitorar a sua duração.

As rotinas em caso de chuva seguem:

- *short int QntDias(short int ano, short int mes, short int dia)*;
- *void Print_horario()*;
- *void conexao_rede()*;
- *void check_buffer()*;
- *bool enviar_db_(bool chuva_s, short int Intens_ch_s, short int min_s, short int data_s)*;
- *void salvar_SPIFFS()*;
- *int kalman(float sensor)*;
- *void requisitar_token(bool setup)*;
- *void deletarArquivo()*;
- *void IRAM_ATTR handleHallSensor()*.

Na função *QntDias()*, os dados são recebidos do módulo RTC para calcular a quantidade de dias desde o primeiro dia do ano de 2024. Esse método permite reduzir o tamanho da variável destinada a armazenar o valor dos dias, uma vez que não é necessário armazenar o ano completo dos dados. O ano de início é fixo, facilitando o gerenciamento e a precisão das informações temporais.

Na função *Print_horario()*, a data e o horário atuais são solicitados ao módulo RTC e armazenados em uma variável local do tipo RtcDateTime, conforme a biblioteca DS1302.h. Essa variável, denominada now, é então enviada à porta serial USB para permitir a inspeção dos dados pelo usuário durante a execução do sistema. Após a execução dessa rotina, a função chama o método *conexao_rede()*.

Na função *conexao_rede()*, os valores de data e hora são extraídos novamente do módulo RTC e armazenados em uma variável local, também denominada "now". A função então envia esses dados para o método *QntDias()*, que calcula a quantidade de dias desde o início do ano e retorna esse valor. O retorno é alocado e enviado para a nuvem, caso o sistema esteja conectado à internet.

Se a conexão com a rede Wi-Fi estiver estabelecida, o método explora a memória interna SPIFFS através da função *check_buffer()*. Esta função verifica se há dados armazenados que ainda não foram enviados para o banco de dados. Se dados forem encontrados, eles serão enviados para o banco de dados. Se a conexão com a rede Wi-Fi não estiver disponível, o sistema tentará reconectar-se à rede. Se a reconexão for bem-sucedida, a rotina de envio será executada. Caso contrário, os dados serão armazenados na memória Flash do ESP32 para envio posterior.

No método de checagem da memória interna do microcontrolador, a função verifica se há valores alocados na memória SPIFFS. Se valores forem encontrados, a função *enviar_db_()*é chamada para enviar essas informações para o banco de dados via API, incluindo os valores atuais de chuva. Após o envio dos dados alocados, a função *deletarArquivo()*, é utilizada para remover o arquivo interno da SPIFFS. Isso é necessário porque, mesmo que o arquivo não contenha dados, ele ainda ocupa 512 KB de espaço na memória. Se não houver conteúdo na SPIFFS, o sistema procederá enviando apenas os dados atuais de chuva.

Na rotina *salvar_SPIFFS()*, os dados são inseridos na memória Flash. Se já houver informações escritas, o arquivo é aberto com o comando "FILE_APPEND" como em *File file = SPIFFS.open(Arquivo_SPIFFS, FILE_APPEND)* , cuja finalidade é levar o ponteiro de escrita para a última linha, sem a necessidade de percorrer todo o arquivo procurando a última linha. Se não houver conteúdo no local, apenas será feita a inserção com o comando "FILE_WRITE" cuja finalidade é abrir o arquivo para escrita na primeira linha. O arquivo está sendo aberto como tipo ".txt", então no final de cada linha, há o

código “\n”.

Na rotina *salvar_SPIFFS()*, os dados são armazenados na memória Flash do microcontrolador. Se já existirem informações no arquivo, ele é aberto em modo de anexação com o comando “FILE_APPEND”, conforme o comando *File file = SPIFFS.open(Arquivo_SPIFFS, FILE_APPEND)* . Isso move o ponteiro de escrita para a última linha do arquivo, evitando a necessidade de percorrer todo o arquivo para encontrar a última linha. Caso o arquivo não contenha dados, ele é aberto em modo de escrita com o comando “FILE_WRITE”, que inicializa a escrita a partir da primeira linha. O arquivo é salvo com a extensão ".txt", e cada nova linha é terminada com o caractere de nova linha “\n”.

O código de todas as funções estão ilustradas no Anexo A (5).

4 RESULTADOS

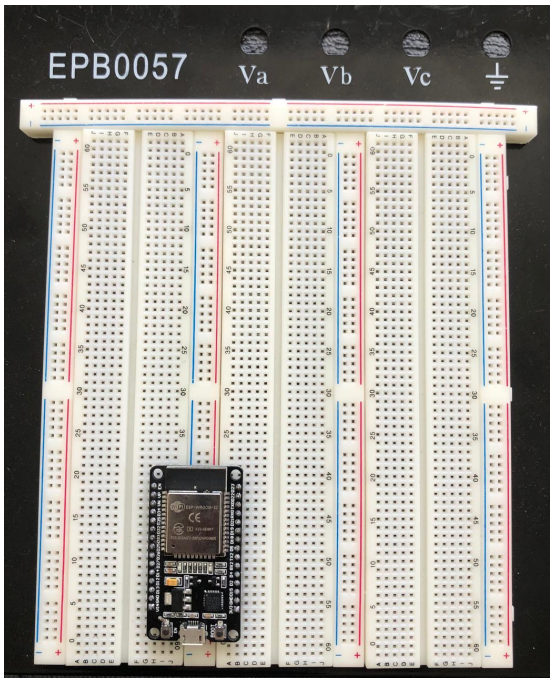
Os testes foram elaborados e executados em cada etapa do sistema, incluindo testes individuais para cada módulo e sensor, bem como testes com o sistema completo em funcionamento. O objetivo foi validar as conexões e o código, assegurando que a placa de circuito impresso e a estrutura 3D do dispositivo fossem adequadas. Foram realizados testes tanto em ambientes controlados (*indoor*) quanto em ambientes externos (*outdoor*), onde o dispositivo será efetivamente utilizado.

Os testes em ambiente controlado foram realizados no Conlab do nPITI, que faz parte do Instituto Metrópole Digital (IMD) da Universidade Federal do Rio Grande do Norte (UFRN), e no Laboratório EVO da Dois A Engenharia e Tecnologia LTDA. Durante esses testes, foram obtidos resultados satisfatórios que corroboram os objetivos do projeto.

4.1 Testes dos sensores e módulos

Nos testes de bancada, foi utilizada uma placa de prototipagem ou *protoboard* para uma maior facilidade de construção dos testes. A placa usada tem 2390 pontos e é mostrada na Figura 33 junto ao ESP32 que foi utilizado para os testes.

Figura 33 – Placa de prototipagem de 2390 pontos



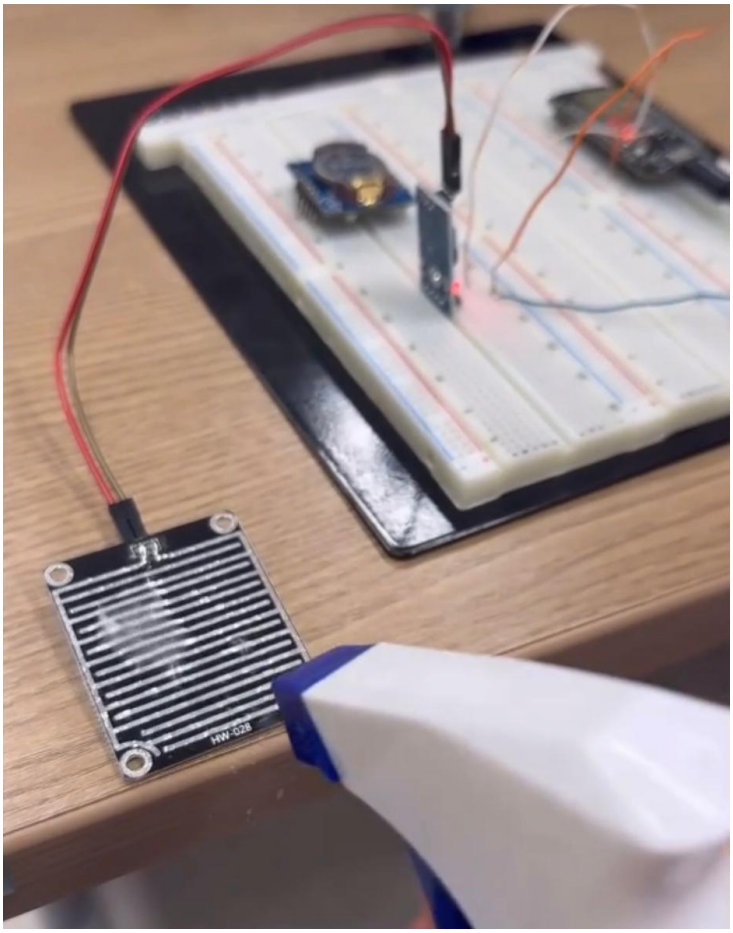
Fonte: Elaborada pelo autor.

Primeiramente, foi testado o módulo sensor de chuva como mostra a Figura 34.

Nos testes, foram produzidas rajadas de água com um borrifador para simular a chuva. A água de chuva atinge o solo com uma força que fragmenta as gotas em vários pingos menores, que se espalham em diferentes direções. No entanto, o jato de água do borrifador não possui a mesma velocidade e força, o que pode levar a uma calibração imprecisa do sensor se utilizado exclusivamente esse método de análise.

Apesar dessa limitação, o uso do borrifador permitiu obter uma base sólida sobre o funcionamento do sensor e observar a variação dos valores digitais provenientes da conversão analógica-digital do microcontrolador. Esses testes forneceram informações valiosas para ajustar e calibrar o sistema de forma mais eficaz.

Figura 34 – Primeiro teste do sensor de chuva



Fonte: Elaborada pelo autor.

Houve testes para verificar o tempo necessário para que o sensor cesse a detecção de chuva após a parada da queda de água sobre sua superfície. Na Figura 35, o sensor foi colocado distante do microcontrolador, com um fio passando pela janela, e o sensor foi apoiado em uma árvore. Esse arranjo representou um teste em ambiente aberto.

Figura 35 – Teste *outdoor* do sensor de chuva

Fonte: Elaborada pelo autor.

O teste revelou que, com o borrifador, o sensor demorava para relatar o término da chuva. Contudo, em uma chuva real, o período de detecção do fim da precipitação é significativamente menor, devido ao ambiente ao ar livre, onde o vento e o efeito do sol influenciam mais intensamente o sensor.

Para testar o sensor de chuva, foi utilizado um código básico, visto que o módulo do sensor requer apenas uma porta analógica do microcontrolador para obter os valores, como mostra o código 4.1.

Listing 4.1 – Código para GPS com ESP32

```
1 void setup() {
2   Serial.begin(115200);
3 }
4
5 void loop() {
6   Valor_sensor_chuva = analogRead();
7   Serial.println(Valor_sensor_chuva);
8 }
```

O teste do GPS, foi montado um circuito simples com o objetivo de determinar a quantidade de satélites conectados e avaliar a precisão do módulo GPS em relação aos valores de latitude, longitude, data e hora. O teste permitiu verificar a capacidade do módulo de fornecer informações geográficas e temporais precisas, essencial para validar o desempenho do sistema de localização.

Usando o código 4.2, foi possível receber os dados de latitude, longitude.

Listing 4.2 – Código para GPS com ESP32

```
#include <TinyGPS++.h>

// Escolhendo a comunicacao serial 2 do esp32 cujos pinos sao 16 e 17
HardwareSerial gpsSerial(2);

// declaracao da variavel do gps
TinyGPSPlus gps;

void setup() {
  // Inicializa a serial de debug
  Serial.begin(115200);
  // Inicializa a serial do GPS
  gpsSerial.begin(9600, SERIAL_8N1, 16, 17); // RX = 16, TX = 17
}

void loop() {
  while (gpsSerial.available() > 0) {
    gps.encode(gpsSerial.read());
    if (gps.location.isUpdated()) {
      Serial.print("Latitude: ");
      Serial.println(gps.location.lat(), 6);
      Serial.print("Longitude: ");
      Serial.println(gps.location.lng(), 6);
      Serial.print("Altitude: ");
      Serial.println(gps.altitude.meters());
      Serial.print("Satellites: ");
      Serial.println(gps.satellites.value());
    }
  }
}
```

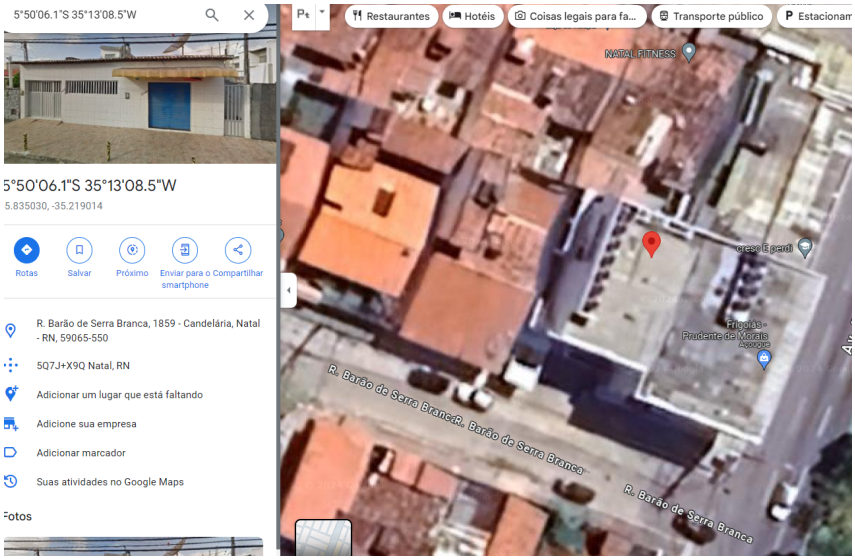
Para verificar a precisão das medições do GPS, foi calculada a distância entre os pontos utilizando o Google Maps. Observou-se os valores de latitude e longitude do local de teste e comparou-se com os valores fornecidos pelo Google Maps, que foram -5.83501654, -35.21899232, conforme ilustrado na Figura 36.

Figura 36 – Localização do Google Maps

Fonte: Elaborada pelo autor.

Como mostra a Figura 37, a diferença é muito pequena, sendo diferença de centímetros a alguns metros. Os valores de latitude e longitude do módulo foram: -5.83502983; -35.21901383.

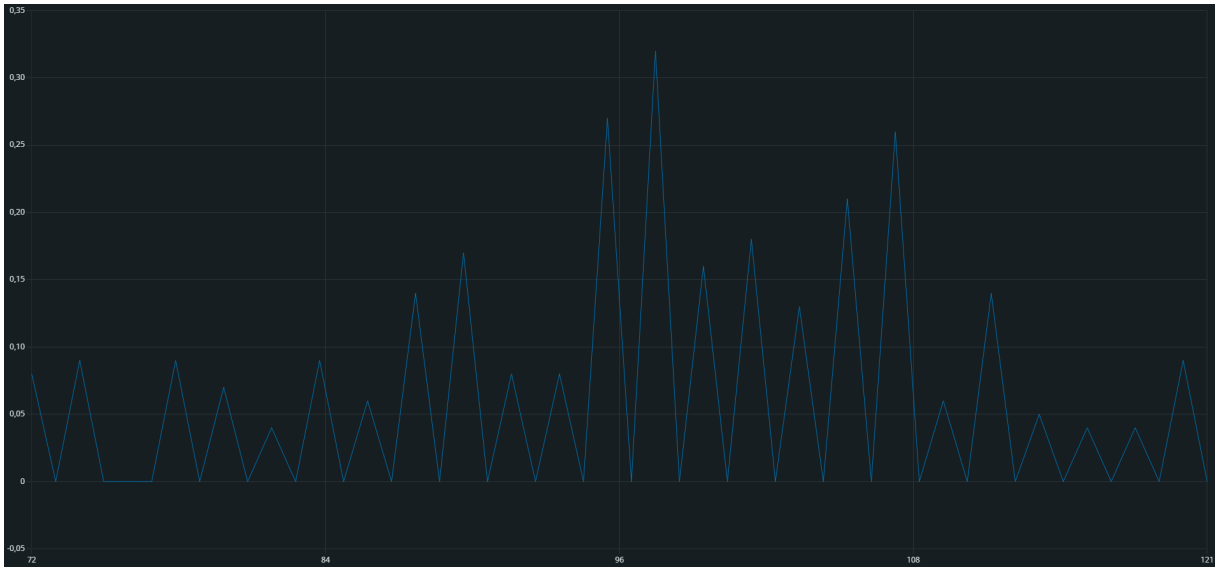
Figura 37 – Localização do GPS plotada no Google Maps



Fonte: Elaborada pelo autor.

Para avaliar a precisão dos valores fornecidos pelo módulo GPS, utilizou-se a fórmula de Haversine para calcular a distância entre os pontos. A maior distância registrada entre dois pontos foi de 32 cm em um total de 121 pontos amostrados durante o teste. O gráfico com a dispersão dos pontos está apresentado na Figura 38.

Figura 38 – Distância entre os pontos do módulo GPS



Fonte: Elaborada pelo autor.

Com a precisão em centímetros, o módulo satisfaz os requisitos de localização, uma vez que é suficientemente preciso para identificar em qual setor de uma obra a chuva está ocorrendo. Vale ressaltar que o teste foi realizado em ambiente fechado e com a conexão de apenas 4 satélites, o que é considerado um valor baixo. Normalmente, a precisão do módulo melhora com a conexão de 7 ou mais satélites.

O GPS também fornece a data e hora, então, com o código 4.3.

Listing 4.3 – Código para GPS com ESP32

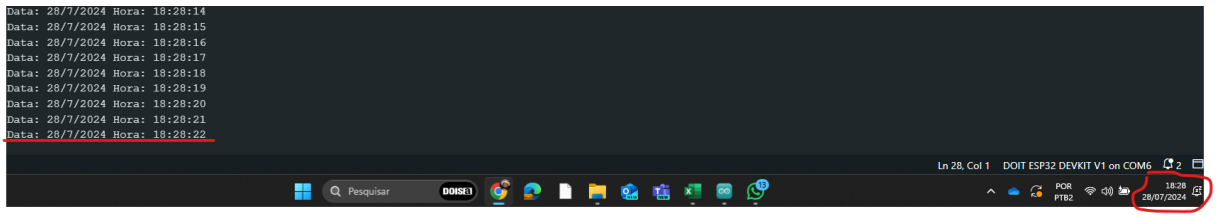
```
1
2 #include <TinyGPS++.h>
3
4 // Configura  o do serial para comunica  o com o GPS
5 HardwareSerial ss(1);
6 TinyGPSPlus gps;
7
8 const int UTC_OFFSET = -3; // Configuracao da hora para o fuso horario
      brasileiro
9
10 void setup() {
11   Serial.begin(115200); // Serial para debug
12   ss.begin(9600, SERIAL_8N1, 16, 17); // Serial1 para GPS (TX, RX)
13
14   Serial.println(F("Iniciando..."));
15 }
16
17 void loop() {
18   while (ss.available() > 0) {
```

```
19      gps.encode(ss.read());
20
21      if (gps.date.isUpdated() && gps.time.isUpdated()) {
22        short int ano = gps.date.year();
23        short int mes = gps.date.month();
24        short int dia = gps.date.day();
25        short int hora = gps.time.hour();
26        short int min = gps.time.minute();
27        short int sec = gps.time.second();
28
29        // ajuste de hora para o fuso horario do Brasil
30        hora += UTC_OFFSET;
31        if (hora >= 24) {
32          hora -= 24;
33          dia++;
34        } else if (hora < 0) {
35          hora += 24;
36          dia--;
37        }
38
39        // ajuste do dia, mes e ano
40        if (dia > 31) {
41          dia = 1;
42          mes++;
43        } else if (dia < 1) {
44          dia = 31;
45          mes--;
46        }
47
48      // ajuste do mes
49        if (mes > 12) {
50          mes = 1;
51          ano++;
52        } else if (mes < 1) {
53          mes = 12;
54          ano--;
55        }
56
57        Serial.print("Data: ");
58        Serial.print(dia);
59        Serial.print("/");
60        Serial.print(mes);
61        Serial.print("/");
62        Serial.print(ano);
63
64        Serial.print(" Hora: ");
65        Serial.print(hora);
```

```
66          Serial.print(":");
67          Serial.print(min);
68          Serial.print(":");
69          Serial.println(sec);
70        }
71    }
72 }
```

O resultado é mostrado na Figura 39.

Figura 39 – Data e hora advindos do NEO6M

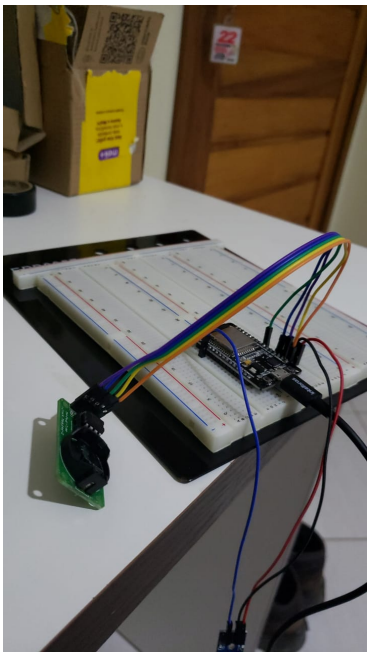


Fonte: Elaborada pelo autor.

O módulo DS1302, que é o *Real Time Clock* externo do dispositivo, também foi testado em bancada, como mostra a Figura 40. O RTC tem sua data e hora atualizados com os dados do servidor NTP ou GPS.

Caso aja internet, sua atualização será pelo servidor NTP, caso contrário, será pelo GPS. O teste foi realizado com o código 4.4.

Figura 40 – Distância entre os pontos do módulo GPS



Fonte: Elaborada pelo autor.

Listing 4.4 – Código para GPS com ESP32

```
#include <ThreeWire.h>
#include <RtcDS1302.h>

ThreeWire MyWire(4,5,2);
RtcDS1302<ThreeWire>rtc(MyWire);
char datestring[20];

void setup() {
  Serial.begin(115200);
  rtc.Begin();
}

void loop() {
  // Recebe a hora e data atuais do modulo DS1302
  RtcDateTime now = rtc.GetDateTime();

  // Exibicao da hora no monitor serial:
  snprintf_P(datestring,
             sizeof(datestring),
             PSTR("%02u/%02u/%04u %02u:%02u:%02u"),
             now.Month(),
             now.Day(),
             now.Year(),
             now.Hour(),
             now.Minute(),
             now.Second() );
  Serial.println(datestring);

  delay(1000);

}
```

Em relação ao pluviômetro, foram realizados testes de bancada com o sensor de efeito Hall para determinar a polaridade de acionamento. O sensor Hall utilizado é do tipo unipolar, o que significa que apenas o polo norte ou sul de um ímã pode ativar o sensor. O teste realizado está ilustrado na Figura 20.

Os resultados dos testes informaram o lado de influência do Imã no sensor para o acoplamento na estrutura impressa em 3D do pluviômetro.

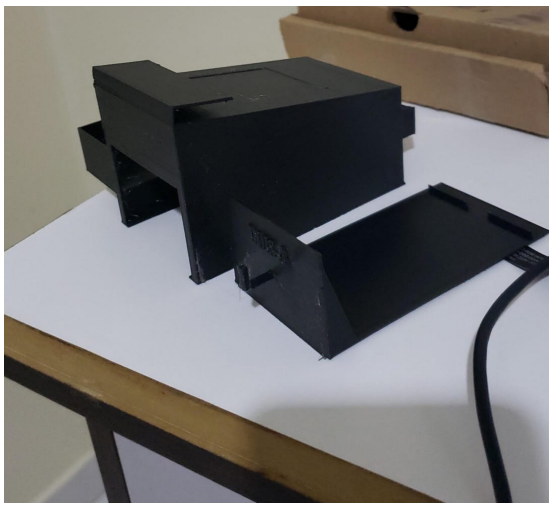
4.2 Estrutura

De acordo com os desenhos ilustrados no Capítulo 3, a estrutura foi enviada para o laboratório para construção por manufatura aditiva. Utilizando um modelo virtual e uma

impressora 3D, foi possível criar cada peça da estrutura por meio da adição de camadas de filamento de polímero. Para o dispositivo, foi utilizado o plástico Acrilonitrila Butadieno Estireno (ABS).

Como mostrada na Figura 22, a caixa de proteção do ESP32 teve sua impressão 3D realizada pelo ProtoLab da Universidade Federal do Rio Grande do Norte e o resultado é ilustrado na Figura 41.

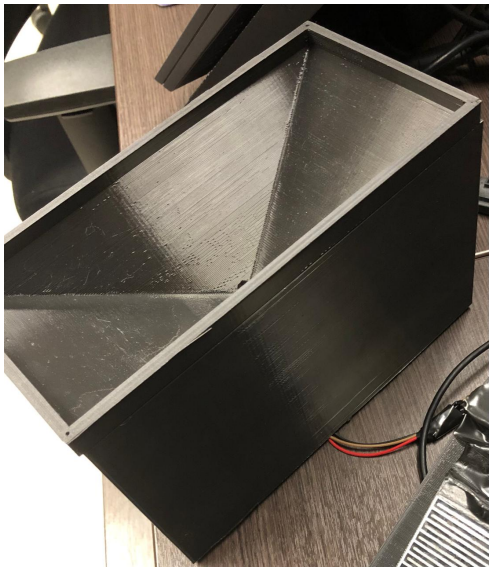
Figura 41 – Caixa de proteção do ESP32



Fonte: Elaborada pelo autor.

Assim como a caixa de proteção, o pluviômetro também foi produzido na impressora 3D com o mesmo filamento da estrutura de proteção do ESP32, respeitando as dimensões do desenho 3D mostrado na Subseção 3.4.2. A Figura 42 ilustra o resultado da estrutura construída.

Figura 42 – Visão lateral do pluviômetro



Fonte: Elaborada pelo autor.

Na Figura 43, todas as peças do pluviômetro estão à mostra, sendo elas o funil da parte superior, a base e o sistema de básculas.

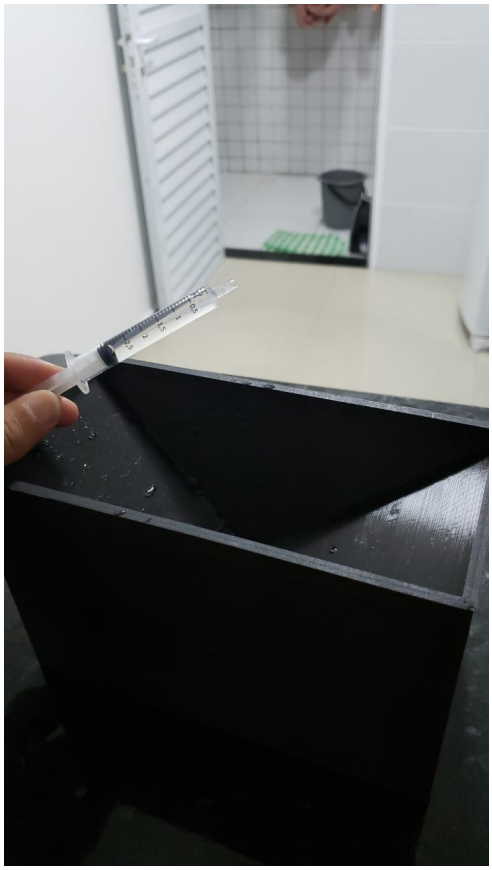
Figura 43 – Cada peça do pluviômetro

Fonte: Elaborada pelo autor.

Após a montagem, o equipamento foi calibrado para a quantidade de mililitros

com a qual a báscula se movimenta. De acordo com o cálculo elaborado na Subsubseção 3.3.2.3, pode-se converter a quantidade de mililitros em milímetros. A Figura 44 ilustra a calibração realizada com uma seringa de 3 ml. A seringa foi completamente carregada com água e, ao ser despejada no topo do pluviômetro, a báscula foi ouvida se movimentando e a água foi observada caindo pelo escorredor no fundo do pluviômetro.

Figura 44 – Calibração do pluviômetro



Fonte: Elaborada pelo autor.

Sabendo que a báscula aguenta 3 mililitros de água, pode-se determinar quantos milímetros de água foram precipitados a cada movimentação. Voltando à Equação (3.3), pode-se afirmar que:

$$\begin{aligned} \text{Volume} &= \text{Comprimento} \times \text{Largura} \times \text{Altura} \\ 3\,\text{ml} &= 20\,\text{cm} \times 10\,\text{cm} \times x\,\text{cm} \\ x &= \frac{3}{20 \times 10} \\ x &= \frac{3}{200} = 0,015\,\text{cm} \\ &= 0,015\,\text{cm} \times 10\frac{\text{mm}}{\text{cm}} = 0,15\,\text{mm de precipitação de água} \end{aligned} \tag{4.1}$$

A cada movimento da báscula, o sistema contabiliza 0,15 mm de água precipitada pela chuva. Esse valor é adicionado conforme o código de interrupção do sensor Hall, que é gerenciado pela rotina do ESP32 quando há detecção de chuva, conforme ilustrado no código 4.5.

Listing 4.5 – Código para GPS com ESP32

```
1 // FUNCAO DE CONTAGEM DO VALOR DO PLUVIOMETRO
2 void IRAM_ATTR handleHallSensor() {
3   if(Intens_ch){
4     qnt_mm += 0.15;
5     Serial.println(qnt_mm);
6   }
7 }
```

4.3 Placa de circuito impresso

A placa seguiu o design apresentado na Subseção 3.5.1 e foi enviada ao laboratório para confecção com uma única camada de cobre, correspondente à camada inferior ou *bottom layer*.

O resultado da estrutura da placa é mostrada na Figura 45.

Figura 45 – Resultado da manufatura da placa de circuito impressa

Fonte: Elaborada pelo autor.

4.4 Protótipo

Antes da construção do protótipo final do projeto, foi elaborado um protótipo de papelão. Esta estrutura foi projetada exclusivamente para testar o conjunto composto pelo sensor de chuva, módulo RTC e microcontrolador. O módulo GPS não foi incluído nesses testes iniciais; sua integração ocorreu somente após a finalização das peças do dispositivo.

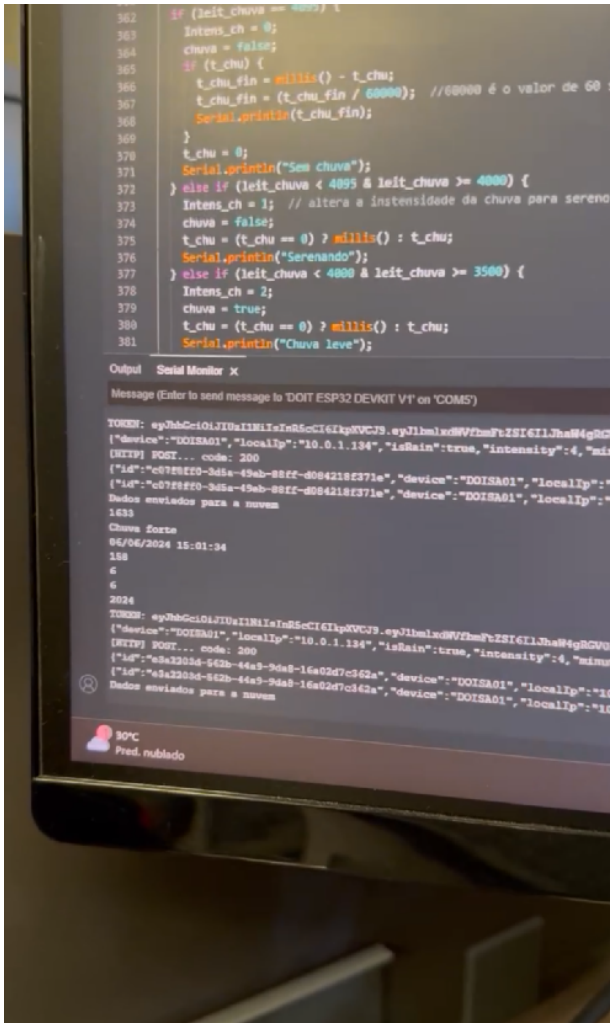
A Figura 46 mostra como ficou a estrutura de papelão.

Figura 46 – Estrutura de papelão

Fonte: Elaborada pelo autor.

O protótipo de papelão foi instalado no teto do laboratório e utilizado para armazenar e enviar dados para o banco de dados por meio da API. A comunicação com a API é ilustrada na Figura 47, que mostra a resposta pelo protocolo HTTP. O retorno do código 200 indica que o comando *POST* foi executado com sucesso.

Figura 47 – Teste de comunicação com a API



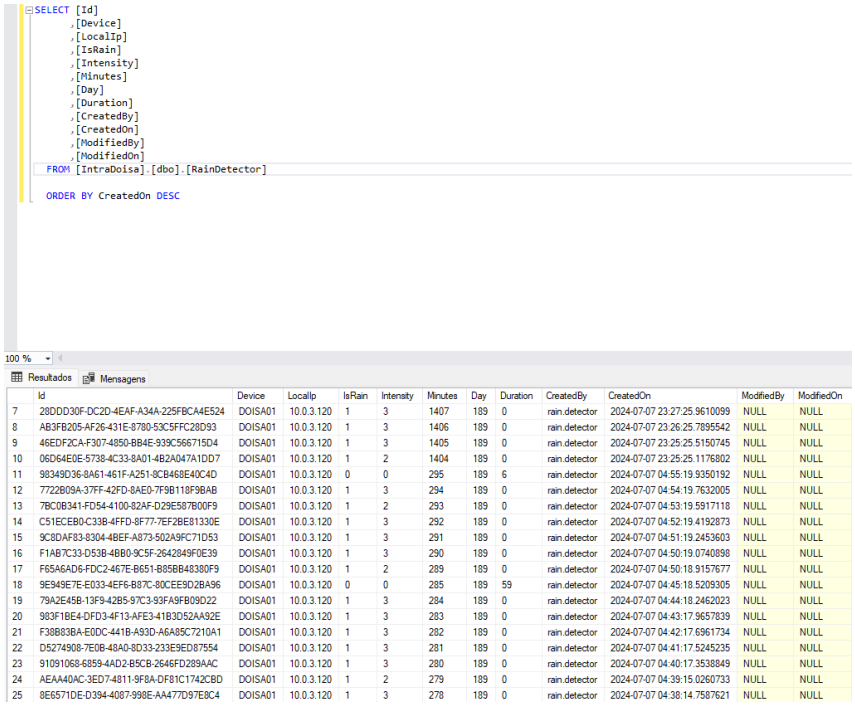
Fonte: Elaborada pelo autor.

4.5 Banco de Dados

No banco de dados, foram criadas as colunas “Device”, “LocalIp”, “IsRain”, “Intensity”, “Minutes”, “Day” e “Duration”. As outras colunas foram criadas para manutenção da própria tabela de dados, mas não são valores enviados pelo dispositivo de chuva.

A Figura 48 ilustra como ficou a base de dados após os testes com o protótipo de papelão.

Figura 48 – Banco de dados SQL



```
SELECT [Id]
      ,[Device]
      ,[LocalIp]
      ,[IsRain]
      ,[Intensity]
      ,[Minutes]
      ,[Day]
      ,[Duration]
      ,[CreatedBy]
      ,[CreatedOn]
      ,[ModifiedBy]
      ,[ModifiedOn]
  FROM [IntraDoisa].[dbo].[RainDetector]

  ORDER BY CreatedOn DESC
```

100 %

Resultados | Mensagens

	Id	Device	LocalIp	IsRain	Intensity	Minutes	Day	Duration	CreatedBy	CreatedOn	ModifiedBy	ModifiedOn
7	28DDD30F-DC2D-4EAF-A34A-225FBCA4E524	DOISA01	10.0.3.120	1	3	1407	189	0	rain.detector	2024-07-07 23:27:25.9610099	NULL	NULL
8	AB3FB205-AF26-431E-8780-53C5FFC28D93	DOISA01	10.0.3.120	1	3	1406	189	0	rain.detector	2024-07-07 23:26:25.7895542	NULL	NULL
9	46EDF2CA-F307-4850-BB4E-939C566715D4	DOISA01	10.0.3.120	1	3	1405	189	0	rain.detector	2024-07-07 23:25:25.5150745	NULL	NULL
10	06D64E0E-5738-4C33-8A01-4B2A047A1DD7	DOISA01	10.0.3.120	1	2	1404	189	0	rain.detector	2024-07-07 23:25:25.1176802	NULL	NULL
11	98349D36-8A61-461F-A251-8CB468E40C4D	DOISA01	10.0.3.120	0	0	295	189	6	rain.detector	2024-07-07 04:55:19.9350192	NULL	NULL
12	7722B09A-37FF-42FD-8AE0-7F9B118F9BAB	DOISA01	10.0.3.120	1	3	294	189	0	rain.detector	2024-07-07 04:54:19.7632005	NULL	NULL
13	7BC0B341-FD54-4100-82AF-D29E587B00F9	DOISA01	10.0.3.120	1	2	293	189	0	rain.detector	2024-07-07 04:53:19.5917118	NULL	NULL
14	C51ECEB0-C33B-4FFD-8F77-7EF2BE81330E	DOISA01	10.0.3.120	1	3	292	189	0	rain.detector	2024-07-07 04:52:19.4192873	NULL	NULL
15	9C8DAF83-8304-4BEF-A873-502A9FC71D53	DOISA01	10.0.3.120	1	3	291	189	0	rain.detector	2024-07-07 04:51:19.2453603	NULL	NULL
16	F1AB7C33-D53B-4BB0-9C5F-2642849F0E39	DOISA01	10.0.3.120	1	3	290	189	0	rain.detector	2024-07-07 04:50:19.0740898	NULL	NULL
17	F65A6AD6-FDC2-467E-B651-B85BB48380F9	DOISA01	10.0.3.120	1	2	289	189	0	rain.detector	2024-07-07 04:50:18.9157677	NULL	NULL
18	9E949E7E-E033-4EF6-B87C-80CEE9D2BA96	DOISA01	10.0.3.120	0	0	285	189	59	rain.detector	2024-07-07 04:45:18.5209305	NULL	NULL
19	79A2E45B-13F9-42B5-97C3-93FA9FB09D22	DOISA01	10.0.3.120	1	3	284	189	0	rain.detector	2024-07-07 04:44:18.2462023	NULL	NULL
20	983F1BE4-DFD3-4F13-AFE3-41B3D52AA92E	DOISA01	10.0.3.120	1	3	283	189	0	rain.detector	2024-07-07 04:43:17.9657839	NULL	NULL
21	F38B83BA-E0DC-441B-A93D-A6A85C7210A1	DOISA01	10.0.3.120	1	3	282	189	0	rain.detector	2024-07-07 04:42:17.6961734	NULL	NULL
22	D5274908-7E0B-48A0-8D33-233E9ED87554	DOISA01	10.0.3.120	1	3	281	189	0	rain.detector	2024-07-07 04:41:17.5245235	NULL	NULL
23	91091068-6859-4AD2-B5CB-2646FD289AAC	DOISA01	10.0.3.120	1	3	280	189	0	rain.detector	2024-07-07 04:40:17.3538849	NULL	NULL
24	AEAA40AC-3ED7-4811-9F8A-DF81C1742CBD	DOISA01	10.0.3.120	1	2	279	189	0	rain.detector	2024-07-07 04:39:15.0260733	NULL	NULL
25	8E6571DE-D394-4087-998E-AA477D97E8C4	DOISA01	10.0.3.120	1	3	278	189	0	rain.detector	2024-07-07 04:38:14.7587621	NULL	NULL

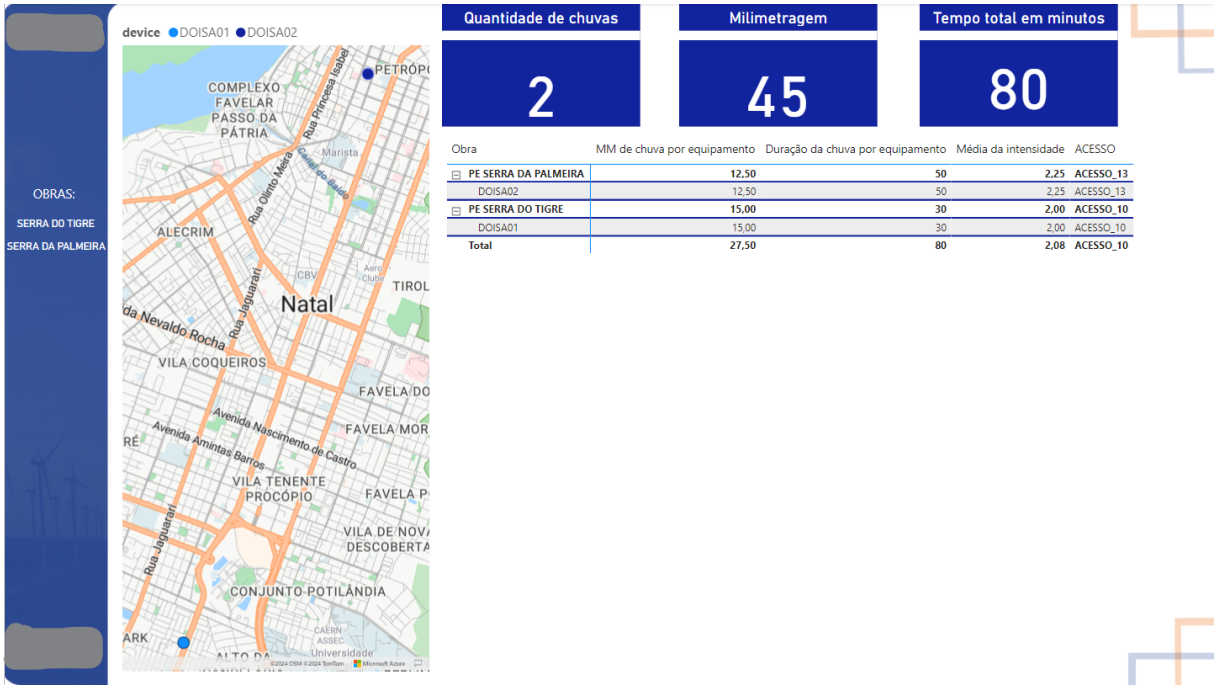
Fonte: Elaborada pelo autor.

Nota-se que, sempre que a chuva cessa, a coluna “IsRain” exibe o valor zero, indicando o fim da precipitação. Além disso, a coluna “Duration” registra o tempo de duração da chuva em minutos. O protótipo coletou dados até o dia 07/07/2024, conforme ilustrado na Figura 48.

4.6 Painel de gestão visual

Para melhor visualização dos dados capturados, relatar o histórico e realizar análises das informações obtidas, foi elaborado um design simples no programa Power BI da Microsoft®. A Figura 49 expõe o painel desenvolvido.

Figura 49 – Painel de gestão visual



Fonte: Elaborada pelo autor.

Os dados apresentados no painel são ilustrativos e foram obtidos de forma aleatória.

Para sinalizar a contagem de eventos de chuva, o software utiliza a linguagem DAX para calcular a quantidade de ocorrências em que o campo “Duration” é igual a zero, indicando que uma chuva foi finalizada e sua duração foi registrada no banco de dados pelo microcontrolador. A informação sobre milímetros por hora de chuva do pluviômetro também é calculada usando DAX em uma coluna adicional. Nessa coluna, a duração em minutos é dividida por 60 para obter o valor em horas, e a quantidade de milímetros é dividida pelo valor da duração em horas, resultando na apresentação do valor em mm por hora.

5 CONCLUSÃO

É notável que diversos conceitos foram abordados neste projeto, tendo sido apresentados cada um deles e sua viabilidade testada de forma prática na construção do dispositivo. Foi realizada a produção do protótipo tanto com material de papelão quanto com impressão 3D, e a placa de circuito impresso foi produzida e testada. Os resultados obtidos nos testes corroboraram todo o estudo realizado, pois os dados da chuva foram detectados, enviados ao banco de dados e puderam ser apresentados em um painel de gestão visual.

Observou-se que o projeto conseguiu atingir seis principais requisitos, como referenciar uma chuva a um horário de início e fim, expondo sua duração e apontando a localização do dispositivo.

O dispositivo pode ser aplicado em diversos cenários onde se deseja obter dados sobre a chuva. Em obras de construção civil, dependendo do acordo entre a construtora e o fornecedor, o valor do contrato das máquinas pesadas, como caminhões basculantes, tratores de esteira e motoniveladoras, por exemplo, considera como desconto as horas paradas devido a condições climáticas. Assim, explorando o campo de aplicações, a inserção em obras seria mais vantajosa em relação aos descontos em contratos, além de agregar valor pelo monitoramento do cenário da obra.

A construção da gestão visual dos dados obtidos pelo dispositivo é de fundamental importância e agrega muito valor, pois não se trata apenas de dados em uma tabela, mas da capacidade de visualizar a informação de forma eficiente, gerando maior possibilidade de tomada de decisão baseada na informação obtida.

Portanto, o trabalho demonstra a aplicação de variados tipos de conhecimento, grande abrangência de temas, resultados da pesquisa e desenvolvimento dos conceitos, gerando um equipamento que pode agregar muito valor à logística. Vale salientar que a melhoria contínua deve ser um dos pontos-chave para todo projeto; sendo assim, há sempre espaço para o desenvolvimento, visando a evolução do projeto.

REFERÊNCIAS

AGNOL, C. D. *Comparação entre microcontroladores e aplicação do FPGA no controle do Conversor Boost*. Lages: [s.n.], 2018. Disponível em: https://www.unifacvest.edu.br/assets/uploads/files/arquivos/6929b-dal-agnol,-c.-comparacao-entre-microcontroldores-e-aplicacao-do-fpga-no-controle-do-conversor-boost.-tcc,-2018..pdf. Acesso em: 30 de jun. de 2024.

Amazon Web Services. *O que é uma API?* 2024. Disponível em: https://aws.amazon.com/pt/what-is/api/. Acesso em: 29 jun. 2024.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Computer Networks*, p. 2787–2805, 10 2010.

CIRCUITSTATE. *Getting Started with Espressif ESP32 WiFi/Bluetooth SoC using DOIT ESP32 DevKit V1 Development Board*. 2024. Disponível em: https://www.circuitstate.com/tutorials/getting-started-with-espressif-esp32-wifi-bluetooth-soc-using-doit-esp32-devkit-v1-development-board/. Acesso em: 27 jun. 2024.

DEVICES, A. *UART: A Hardware Communication Protocol*. 2024. Disponível em: https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html. Acesso em: 28 jun. 2024.

DIGITAL, I. *Módulo RTC DS1302 Real Time Clock*. 2024. Disponível em: https://www.institutodigital.com.br/produto/modulo-rtc-ds1302-real-time-clock/. Acesso em: 27 jun. 2024.

ESCOLA, B. *Tipos de Chuva*. 2024. Disponível em: https://brasilescola.uol.com.br/geografia/tipos-chuva.htm. Acesso em: 20 jun. 2024.

FILHO, M. F. Internet das coisas. In: UNIVERSIDADE DO SUL DE SANTA CATARINA. *Anais da Conferência sobre Internet das Coisas*. Palhoça, 2016. Disponível em: https://www.researchgate.net/publication/319881659_Internet_das_Coisas_Internet_of_Things. Acesso em: 26 jun. de 2024.

GARCIA-MOLINA, H.; ULLMAN, J. D.; WIDOM, J. *Database System Concepts*. [S.l.]: McGraw-Hill, 2024. Disponível em: https://db-book.com/bib-dir/PDF-dir/3.pdf. Acesso: 27 jun. 2024.

KATIE, B. Internet of things (iot) for environmental monitoring. *International Journal of Computing and Engineering*, v. 6, p. 29–42, 07 2024.

MAIER, A.; SHARP, A.; VAGAPOV, Y. Internet of things. In: *Comparative Analysis and Practical Implementation of the ESP32 Microcontroller Module for the Internet of Things*. [S.l.: s.n.], 2017.

MATTERN, F.; FLOERKEMEIER, C. *From the Internet of Computers to the Internet of Things*. 2010. Disponível em: https://link.springer.com/chapter/10.1007/978-3-642-17226-7_15. Acesso em: 25 de jun. de 2024.

MENEZES, A. C. dos S. *DESENVOLVIMENTO E CALIBRAÇÃO DE PLUVIÔMETROS AUTOMÁTICOS DE BAIXO CUSTO*. 2024. Disponível em: https://portais.univasf.edu.br/ppgea/pesquisa/publicacoes-1/arquivos/ana-carla-dos-santos-menezes.pdf. Acesso em: 30 jun. 2024.

MOHANAN, V. *Getting Started with Espressif ESP32 Wi-Fi Bluetooth SoC using DOIT-ESP32-DevKit-V1 Development Board*. Índia: [s.n.], 2023. Disponível em: https://www.unifacvest.edu.br/assets/uploads/files/arquivos/6929b-dal-agnol,-c.-comparacao-entre-microcontroldores-e-aplicacao-do-fpga-no-controle-do-conversor-boost.-tcc,-2018..pdf. Acesso em: 25 de jun. de 2024.

M&T, R. *Edição 151*. 2024. Disponível em: https://revistamt.com.br/Arquivos/Edicoes/MT_151.pdf. Acesso em: 10 jun. 2024.

OLIVEIRA, G. *Módulo GPS GY-NEO6MV2 – Guia completo de como usá-lo com o Arduino*. 2022. Disponível em: https://blogmasterwalkershop.com.br/arduino/modulo-gps-gy-neo6mv2-guia-completo-de-como-usa-lo-com-o-arduino. Acesso em: 05 jun. 2024.

Oracle. *O que é um Banco de Dados?* 2024. Disponível em: https://www.oracle.com/br/database/what-is-database/. Acesso em: 29 jun. 2024.

PIMENTEL, R. *SPIFFS: O sistema de arquivos do ESP8266/32*. 2024. Disponível em: https://embarcados.com.br/spiffs-o-sistema-de-arquivos-do-esp8266-32/. Acesso em: 27 jun. 2024.

ROBOCORE. *Sensor de Chuva*. 2024. Disponível em: https://www.robocore.net/sensor-ambiente/sensor-de-chuva?gad_source=1&gclid=Cj0KCQjwzby1BhCQARIsAJ_0t5OAEiXkr8AKFa8wesherBOfKDUjwBoSvBBdIbdJvEJq6LUdIFtWNNEaAqpREALw_wcB. Acesso em: 28 jun. 2024.

ROSARIO, A. del. *Cadence to Acquire Tensilica*. 2023. Disponível em: https://www.chipestimate.com/Cadence-to-Acquire-Tensilica/Cadence/news/20251. Acesso em: 26 jun. 2024.

SAMIULLAH, M.; IRFAN, M. Z.; RAFIQUE, A. *Microcontrollers: A Comprehensive Overview and Comparative Analysis of Diverse Types*. 2023. Disponível em: https://engrxiv.org/preprint/view/3228. Acesso em: 28 jun. 2024.

SANTOS, S. *ESP32 Pinout Reference: Which GPIO pins should you use?* 2024. Disponível em: https://randomnerdtutorials.com/esp32-pinout-reference-gpios/. Acesso em: 27 jun. 2024.

SEMICONDUCTOR, D. *DS1302 Trickle-Charge Timekeeping Chip*. 1996. Disponível em: https://pdf1.alldatasheet.com/datasheet-pdf/view/58478/DALLAS/DS1302.html. Acesso em: 29 jun. 2024.

SOLUTIONS, Q. W. *UART Interface - Application Guide*. 2024. Disponível em: https://python.quectel.com/doc/Application_guide/en/hardware/peripheral-interfaces/UART.html?creative=&keyword=&matchtype=&network=x&device=c&gad_source=1&gclid=Cj0KCQjwzby1BhCQARIsAJ_0t5P1jOfQsTKPV3LmgyZ-vHuGXdAkIiCxIj3ZV_wRXz1MvgOpyDRylyMaAiPPEALw_wcB. Acesso em: 27 jun. 2024.

SPLabor. *O que é um Pluviômetro e qual a sua função?* 2024. Disponível em: https://www.splabor.com.br/blog/equipamentos-para-laboratorio/o-que-e-um-pluviometro-qual-a-sua-funcao/. Acesso em: 2024-08-05.

STA Eletrônica. *Como utilizar o módulo GPS NEO-6M com o Arduino*. 2024. Disponível em: https://www.sta-eletronica.com.br/artigos/arduinos/como-utilizar-o-modulo-gps-neo-6m-com-o-arduino. Acesso em: 27 jun. 2024.

STRAUB, M. G. *Pluviômetro arduino como sensor de chuva na estação meteriológica*. 2018. Disponível em: https://www.usinainfo.com.br/blog/pluviometro-arduino-como-sensor-de-chuva-na-estacao-meteorologica/. Acesso em: 01 de jul de 2024.

SUNFOUNDER. *ESP32 Lesson 16: DS1302 Real-Time Clock Module*. n.d. Disponível em: https://docs.sunfounder.com/projects/umsk/en/latest/03_esp32/esp32_lesson16_ds1302.html. Acesso: 20 jun. 2024.

SYSTEMS, E. *ESP32 DevKitC - Getting Started*. 2024. Disponível em: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/hw-reference/esp32/get-started-devkitc.html. Acesso em: 27 jun. 2024.

SYSTEMS, E. *Sleep Modes*. 2024. Disponível em: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/sleep_modes.html. Acesso em: 26 jun. 2024.

Texas Instruments. *LM393-N Low Power, Low Offset Voltage, Dual Comparator*. 2020. Disponível em: https://www.ti.com/lit/ds/symlink/lm393-n.pdf. Acesso em: 27 jun. 2024.

u-blox. *NEO-6 Data Sheet*. 2011. Disponível em: https://d229kd5ey79jzj.cloudfront.net/976/NEO-6_DataSheet_(GPS.G6-HW-09005).pdf. Acesso em: 27 jun. 2024.

ANEXO A - Código Completo

```
#include <SPIFFS.h>
#include <WiFi.h>
#include <ThreeWire.h>
#include <time.h>
#include <HTTPClient.h>
#include <RtcDS1302.h>
#include <ArduinoJson.h>
#include <TinyGPS++.h>

//------------------- INICIO DA DEFINIÇÃO DE VARIÁVEIS E OBJETOS --------------------

//EQUPAMENTO

String EQ = "DOISA01";

// GPS

HardwareSerial gpsSerial(2);
TinyGPSPlus gps;
float latitude = 0;
float longitude = 0;

// WIFI
//const char* ssid = "DOISA";
//const char* password = "engenharia.4488@#";
bool wifi_on;
```

```
//PLUVIOMETRO
const int pinoHall = 13; // Pino onde o sensor Hall está conectado
float qnt_ml;
unsigned short int mm_chuva;

//RTC

ThreeWire MyWire(4, 5, 2);
RtcDS1302<ThreeWire> rtc(MyWire);
char datestring[20];

//CONDIÇÕES DE CHUVA

#define PINO_SCH 34

bool chuva = false;
short int Intens_ch = 0;
unsigned long long t_chu = 0;
short int t_chu_fin = 0;
unsigned long long count_millis = 0;
int leit_chuva;

// COMUNICAÇÃO HTTP

HTTPClient http;
const char* server = "http://10.0.1.246:88";
String endpoint = "/api/Login";  // Endpoint da API
String token_API;

// ALOCAÇÃO DE DADOS NA SPIFFS
```

```
short int dados_buffer = 0;
short int dados_enviados = 0;
short int data = 0;
short int minutos = 0;
bool arquivo_lido = false;
short int cont_tentativas = 0;
const char* Arquivo_SPIFFS = "/dados.txt";

// OBJETOS DO FILTRO KALMAN

float R = 0.1, Q = 1e-3;  //Q = process noise covariance, R = measurement noise covariance
float Xpe0 = 0.0;         // Xpe0 = prior estimation of signal X at time t=0 (current state)
float Xe1 = 0.0;          //Xe1 = estimation of X at time t=1 (previous state)
float Ppe0 = 0.0;         //Ppe0 = prior estimation of "error covariance" at t=0,
float P1 = 1, P0 = 0;     //P1 = error covariance at t=1, P0 = error covariance at t=0
float K = 0.0;
float Z = 0.0;  //K = Kalman gain, Xe0 = estimation of signal at t=0, Z = measured signal at t=0

int count_deg;

//---------------------------------------- INICIO DA FUNÇÕES ----------------------------------------

// FUNÇÃO QUE CONTA QUANTIDADE DE DIAS
short int QntDias(short int ano, short int mes, short int dia) {
  short int DiasMes[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
  short int dias = dia;

  DiasMes[1] = (ano % 4 == 0) ? 29 : DiasMes[1];
```

```
  // Adiciona os dias de meses anteriores
  for (int i = 0; i < mes - 1; i++) {
    dias += DiasMes[i];
  }
  dias += ((ano-2024)*365);
  Serial.println(dias);
  Serial.println(ano);
  return dias;
}

// CONFIGURAÇÃO NTP E FUSO
void initializeTime() {
  configTime(0, 0, "pool.ntp.org");  // Configura o NTP usando o servidor "pool.ntp.org"
  setenv("TZ", "BRT3", 1);
  tzset();
}

// IMPRIMIR OS VALORES NA TELA
void Print_horario() {

  // Obtenção da data e hora atual
  RtcDateTime now = rtc.GetDateTime();

  // Exibe a data e hora no monitor serial
  snprintf_P(datestring,
             sizeof(datestring),
             PSTR("%02u/%02u/%04u %02u:%02u:%02u"),
             now.Day(),
             now.Month(),
             now.Year(),
```

```
            now.Hour(),
            now.Minute(),
            now.Second());
  Serial.println(datestring);
  conexao_rede();
}

// ENVIA AS INFORMAÇÕES PARA A API
void conexao_rede() {

  // Obtenção da data e hora atual
  RtcDateTime now = rtc.GetDateTime();
  data = QntDias(now.Year(), now.Month(), now.Day());
  minutos = now.Hour() * 60 + now.Minute();

  if (WiFi.status() == WL_CONNECTED) {
    check_buffer();
  } else {
    Serial.println("Não há conexão Wi-Fi, Haverá uma tentativa de conexão
novamente, caso contrário, os dados serão salvos na memória Flash");
    WiFi.begin(ssid, password);
    short timeout = millis();
    while (millis() - timeout < 10000 && WiFi.status() != WL_CONNECTED) {
      delay(1000);
      Serial.println("TENTANDO CONEXÃO COM A REDE WIFI...");
    }
    if (WiFi.status() == WL_CONNECTED) {
      Serial.println("WIFI CONECTADO");
      wifi_on = true;
      check_buffer();
    } else {
```

```
      Serial.println("Timeout de conexão com Wifi atingido. Salvando os dados no
buffer");
      wifi_on = false;
      salvar_SPIFFS();
    }
  }
}

// VERIFICA SE HÁ DADOS NO BUFFER
void check_buffer() {

  if (!dados_buffer) {
    if (enviar_db_(chuva, Intens_ch, minutos, data)) {
      Serial.println("Dados enviados para a nuvem");
    } else {
      if (!wifi_on) {
        Serial.println("Não foi possível enviar os dados, verificar conexão com WIFI");
      } else {
        Serial.println("Não foi possível enviar os dados, verificar conexão com a API");
      }
      salvar_SPIFFS();
    }
  } else {
    bool chuva_buffer;
    short int Intens_ch_buffer;
    short int min_buffer;
    short int data_buffer;

    File file = SPIFFS.open(Arquivo_SPIFFS, FILE_READ);
    if (!file) {
      Serial.println("ERRO AO ABRIR O ARQUIVO PARA LEITURA");
```

```
  return;
}
short int linha_aqr = 0;
while (dados_enviados < dados_buffer) {
  if (linha_aqr < dados_enviados) {
    file.readStringUntil('\n');
    linha_aqr++;
    continue;
  }
  file.read((uint8_t*)&chuva_buffer, sizeof(chuva_buffer));
  file.read((uint8_t*)&Intens_ch_buffer, sizeof(Intens_ch_buffer));
  file.read((uint8_t*)&min_buffer, sizeof(min_buffer));
  file.read((uint8_t*)&data_buffer, sizeof(data_buffer));
  file.readStringUntil('\n');
  if (enviar_db_(chuva_buffer, Intens_ch_buffer, min_buffer, data_buffer)) {
    Serial.println("Dados enviados para a nuvem");
    dados_enviados++;
  } else {
    if (!wifi_on) {
      Serial.println("Não foi possível enviar os dados, verificar conexão com WIFI");
      break;
    } else {
      Serial.println("Não foi possível enviar os dados, verificar conexão com a API");
      break;
    }
  }
}
if (dados_enviados == dados_buffer) {
  deletarArquivo();
  dados_enviados = 0;
```

```
      dados_buffer = 0;
    } else {
      file.close();
    }
  }
}

//FUNÇÃO QUE ESVAZIA O BUFFER INTERNO, ENVIANDO TUDO PARA O BANDO DE DADOS PELA API

bool enviar_db_(bool chuva_s, short int Intens_ch_s, short int min_s, short int data_s) {
  if (WiFi.status() == WL_CONNECTED) {
    wifi_on = true;
    http.begin(String(server) + endpoint);              // URL da API
    http.addHeader("Content-Type", "application/json");  // Define o tipo de conteúdo como JSON
    http.addHeader("Authorization", "Bearer " + token_API);
    Serial.println("TOKEN: " + token_API);
    String localIP = WiFi.localIP().toString();

    // CRIANDO JSON
    const size_t capacity = JSON_ARRAY_SIZE(1) + JSON_OBJECT_SIZE(8) + 200;
    StaticJsonDocument<capacity> doc;

    JsonObject obj = doc.to<JsonObject>();
    obj["device"] = EQ;
    obj["localIp"] = localIP;
    obj["isRain"] = chuva_s;
    obj["intensity"] = Intens_ch_s;
    obj["minutes"] = min_s;
    obj["day"] = data_s;
```

```
obj["duration"] = t_chu_fin;
obj["millimeter"] = qnt_ml;
obj["longitude"] = latitude;
obj["longitude"] = longitude;

String jsonString;
serializeJson(doc, jsonString);
Serial.println(jsonString);

int httpCode = http.POST(jsonString);  // Envia os dados usando POST

if (httpCode > 0) {
  Serial.printf("[HTTP] POST... code: %d\n", httpCode);

  if (httpCode == HTTP_CODE_OK) {
    String payload = http.getString();
    Serial.println(payload);
    cont_tentativas = 0;
  } else if (httpCode == 401 && cont_tentativas < 3) {
    Serial.print("Erro de segurança - TOKEN negado.. Solicitando um novo");
    delay(400);
    requisitar_token(false);
  }
  if (cont_tentativas >= 3) {
    cont_tentativas = 0;

    return false;
  }
  return true;
} else {
```

```
      Serial.printf("[HTTP] POST... falha, código: %s\n",
http.errorToString(httpCode).c_str());
      return false;
    }

    http.end();
  } else {
    Serial.println("CONEXÃO DE REDE PERDIDA - WIFI DESCONECTADO");
    wifi_on = false;
    return false;
  }
}

//ESCREVE NA MEMÓRIA FLASH DO ESP
void salvar_SPIFFS() {
  if (dados_buffer > 0) {
    File file = SPIFFS.open(Arquivo_SPIFFS, FILE_APPEND);
    if (!file) {
      Serial.println("Failed to open file for appending");
      return;
    }
    file.write((uint8_t*)&chuva, sizeof(chuva)); // É IMPORTANTE DEVIDO AO CASO
DE FINAL DE CHUVA, ONDE ELE VAI PARA 0
    file.write((uint8_t*)&Intens_ch, sizeof(Intens_ch));
    file.write((uint8_t*)&minutos, sizeof(minutos));
    file.write((uint8_t*)&data, sizeof(data));
    file.print("\n");  // comando para pular linha, acabando a linha
    file.close();
    ++dados_buffer;

  } else {
```

```
    File file = SPIFFS.open(Arquivo_SPIFFS, FILE_WRITE);
    if (!file) {
      Serial.println("Failed to open file for appending");
      return;
    }
    file.write((uint8_t*)&chuva, sizeof(chuva));  // VERIFICAR A REAL NECESSIDADE;
    file.write((uint8_t*)&Intens_ch, sizeof(Intens_ch));
    file.write((uint8_t*)&minutos, sizeof(minutos));
    file.write((uint8_t*)&data, sizeof(data));
    file.write((uint8_t*)&qnt_ml, sizeof(data));
    file.write((uint8_t*)&latitude, sizeof(data));
    file.write((uint8_t*)&longitude, sizeof(data));
    file.print("\n");  // comando para pular linha, acabando a linha
    file.close();
    ++dados_buffer;
    Serial.println("Dados salvos na SPIFFS");
  }
}


//FUNÇÃO DE FILTRO KALMAN
int kalman(float sensor) {
  float Xe0;
  Z = sensor;
  Xpe0 = Xe1;                 //Assumption or prediction 1
  Ppe0 = P1 + Q;              //Assumption or prediction 2
  K = Ppe0 / (Ppe0 + R);      // Measurement update or correction of "Kalman gain"
  Xe0 = Xpe0 + K * (Z - Xpe0);  // Measurement update or correction of "estimated
signal"
  P0 = (1 - K) * Ppe0;        // Measurement update or correction of "error covariance"
  Xe1 = Xe0;                  //Update: current t=0 becomes t=1 in the next step
  P1 = P0;                    //Update: current t=0 becomes t=1 in the next step
```

```
  return Xe0;
}

void requisitar_token(bool setup) {

  endpoint = "/api/Login";
  // CONEXÃO COM A API PARA PEGAR O TOKEN
  http.begin(String(server) + endpoint);              // URL da API
  http.addHeader("Content-Type", "application/json");  // Define o tipo de conteúdo como JSON

  // CRIANDO JSON
  const size_t capacity = JSON_ARRAY_SIZE(1) + JSON_OBJECT_SIZE(4) + 300;
  DynamicJsonDocument doc(capacity);

  JsonObject obj = doc.to<JsonObject>();
  obj["user"] = "rain.detector";
  obj["password"] = "123";
  obj["token"] = "";
  obj["error"] = "";

  String jsonString;
  serializeJson(doc, jsonString);
  Serial.println(jsonString);

  int httpCode = http.POST(jsonString);  // Envia os dados usando POST

  if (httpCode > 0) {
    Serial.printf("[HTTP] POST... code: %d\n", httpCode);
    if (httpCode == HTTP_CODE_OK) {
      String payload = http.getString();
```

```
      Serial.println(payload);
      DeserializationError error = deserializeJson(doc, payload);
      if (error) {
        Serial.print(F("Erro ao deserializar JSON: "));
        Serial.println(error.c_str());
      }

      token_API = (String)doc["token"];
    }
  } else {
    Serial.println("Não foi possível conectar a API");
  }
  endpoint = "/api/RainDetector";
  if (!setup) {
    cont_tentativas++;
    delay(400);
    enviar_db_(chuva, Intens_ch, minutos, data);
  }
}

// FUNÇÃO PARA EXCLUIR O ARQUIVO QUANDO LIDO PARA LIBERAR MEMÓRIA CASO PRECISO
void deletarArquivo() {
  if (SPIFFS.exists(Arquivo_SPIFFS)) {
    // removerá o arquivo se ele existir
    if (SPIFFS.remove(Arquivo_SPIFFS)) {
      Serial.println("Arquivo deletado com sucesso");
    } else {
      Serial.println("Erro ao deletar o arquivo");
    }
  } else {
```

```
    Serial.println("Arquivo não encontrado");
  }
}

//FUNÇÃO DE CONTAGEM DO VALOR DO PLUVIOMETRO
void IRAM_ATTR handleHallSensor() {
  if(Intens_ch){
    qnt_ml += 0.15;
    Serial.println(qnt_ml);
  }
}

//------------------------------------ INICIO DA FUNÇÃO SETUP ----------------------------------

void setup() {

  // INICIO DA COMINICAÇÃO SERIAL
  Serial.begin(115200);



  // Inicializa a comunicação serial com o GPS em 9600 bps velocidade padrão
  gpsSerial.begin(9600, SERIAL_8N1, 16, 17);
  short int satelites = 0;
  unsigned int time_out = millis();
  while(satelites < 2 && (millis() - time_out)<120000) {
    // Verifica se há dados disponíveis no GPS
    if(gpsSerial.available() > 0) {
      gps.encode(gpsSerial.read());

      // Atualiza o número de satélites encontrados
```

```
    satelites = gps.satellites.value();

    // Imprime o número de satélites encontrados
    Serial.print("Satélites encontrados: ");
    Serial.println(satelites);
  }
  else{
    Serial.println("Nenhum GPS encontrado");
    break;
  }

  // Pausa para o loop não ficar tão rápido
  delay(1000);
}

//COFIGURAÇÃO PARA O PLUVIOMETRO
pinMode(pinoHall, INPUT); // Seta o pinoHall como entrada
attachInterrupt(digitalPinToInterrupt(pinoHall), handleHallSensor, FALLING); // Configura a interrupção para o pino 13

// INICIO DA INSCRIÇÃO DOS ARQUIVOS NA MEMÓRIA INTERNA
if (!SPIFFS.begin(true)) {
  Serial.println("Erro na montagem da SPIFFS");
} else {
  Serial.println("Memória preparada para manipulação");
}

if (!SPIFFS.exists(Arquivo_SPIFFS)) {
  Serial.println("O arquivo não existe");
} else {
  File file = SPIFFS.open(Arquivo_SPIFFS, "r");
```

```
  if (!file) {
    Serial.println("Falha ao abrir o arquivo");
  }

    while (file.available()) {
      String line = file.readStringUntil('\n');
      dados_buffer++;
    }
    Serial.println(dados_buffer);
  file.close();
}

// INICIO DO ACESSO AO WIFI
WiFi.begin(ssid, password);
unsigned short timeout = millis();
while ((millis() - timeout) < 12000 && WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("TENTANDO CONEXÃO COM A REDE WIFI...");
}

if (WiFi.status() == WL_CONNECTED) {
  wifi_on = true;
  Serial.println("CONECTADO A REDE");

  // INICIANDO A NTP E CONFIGURANDO O FUSO HORÁRIO
  initializeTime();

  // Aguardando até que a hora esteja atualizada
  Serial.println("Esperando por sincronização");
```

```
    timeout = millis();
    while (time(nullptr) < 1609459200 && (millis() - timeout) < 20000) {  // 1609459200 é 01/01/2021 00:00:00 em UNIX timestamp -- conexão testada por 20 segundos
      delay(100);
      Serial.println("Esperando por sincronização");
    }

    if (wifi_on) {
      Serial.println("Sincronização realizada, data e hora atualizados");
      time_t now = time(nullptr);
      struct tm timeinfo;
      localtime_r(&now, &timeinfo);
      //CONFIGURANDO O RTC COM HORA E DATA ATUALIZADOS
      rtc.Begin();
      RtcDateTime dt(timeinfo.tm_year + 1900, timeinfo.tm_mon + 1, timeinfo.tm_mday, timeinfo.tm_hour, timeinfo.tm_min, timeinfo.tm_sec);
      rtc.SetDateTime(dt);
    } else {
      Serial.println("RTC será conectado sem hora atualizada");

      //CONFIGURANDO O RTC SEM DATA E HORA ATUALIZADOS NA WEB
      rtc.Begin();
    }
    // CONEXÃO COM A API PARA PEGAR O TOKEN
    requisitar_token(true);
  } else {
    Serial.println("API não conectada");
  }

  count_millis = millis();
}
```

```
//-------------------------------------- INICIO DA FUNÇÃO LOOP ---------------------------------
----


void loop() {
  leit_chuva = kalman(analogRead(PINO_SCH));
  count_deg++;
  if (millis() - count_millis > 60000) {
      while (gpsSerial.available() > 0)
    {
     gps.encode(gpsSerial.read());
     if (gps.location.isUpdated())
      {
        latitude = gps.location.lat();
        longitude = gps.location.lng();
      }
   }
    if (dados_buffer > 0 && wifi_on) {
      check_buffer();
    }
    Serial.println(count_deg);
    count_deg = 0;
    Serial.println(leit_chuva);
    if (leit_chuva <= 4095 && leit_chuva >= 4000) {
      Intens_ch = 0;
      chuva = false;
      if (t_chu) {
        t_chu_fin = (millis() - t_chu) / 60000;  //60000 é o valor de 60 segundos por cada
minuto vezes 1000 milésimo que dá um segundo, resultando no valor em minutos.
Sendo um int, só pegará o valor inteiro da divisão
        Serial.println(t_chu_fin);
        Print_horario();
```

```
    mm_chuva = qnt_ml
    ;
    t_chu_fin = 0;
  }
  qnt_ml = 0;
  t_chu = 0;
  Serial.println("Sem chuva");
} else if (leit_chuva < 4000 & leit_chuva >= 3800) {
  Intens_ch = 1;  // altera a instensidade da chuva para sereno
  chuva = false;
  t_chu = (t_chu == 0) ? millis() : t_chu;
  Serial.println("Serenando");
} else if (leit_chuva < 3800 & leit_chuva >= 2800) {
  Intens_ch = 2;
  chuva = true;
  t_chu = (t_chu == 0) ? millis() : t_chu;
  Serial.println("Chuva leve");
  Print_horario();
} else if (leit_chuva < 2800 & leit_chuva >= 1600) {

  Intens_ch = 3;
  chuva = true;
  t_chu = (t_chu == 0) ? millis() : t_chu;
  Serial.println("Chuva moderada");
  Print_horario();
} else {
  Intens_ch = 4;
  chuva = true;
  t_chu = (t_chu == 0) ? millis() : t_chu;
  Serial.println("Chuva forte");
```

```
      Print_horario();
    }
    count_millis = millis();
  }
}
```