LUIS GUSTAVO DE SOUZA OLIVEIRA

SOLUÇÃO BASEADA NO KIT DE DESENVOLVIMENTO ESP32 PARA AUTOMAÇÃO DAS BOMBAS DE ABASTECIMENTO DE COMBUSTÍVEL EM POSTOS INTERNOS

LAVRAS - MG
2023

LUIS GUSTAVO DE SOUZA OLIVEIRA

SOLUÇÃO BASEADA NO KIT DE DESENVOLVIMENTO ESP32 PARA AUTOMAÇÃO DAS BOMBAS DE ABASTECIMENTO DE COMBUSTÍVEL EM POSTOS INTERNOS

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do Curso de Engenharia de Controle e Automação, para a obtenção do título de Bacharel.

Prof. Dr. Wilian Soares Lacerda
Orientador

LAVRAS - MG
2023

LUIS GUSTAVO DE SOUZA OLIVEIRA

SOLUÇÃO BASEADA NO KIT DE DESENVOLVIMENTO ESP32 PARA AUTOMAÇÃO DAS BOMBAS DE ABASTECIMENTO DE COMBUSTÍVEL EM POSTOS INTERNOS

SOLUTION BASED ON THE ESP32 DEVELOPMENT KIT FOR AUTOMATION OF FUEL SUPPLY PUMPS IN INTERNAL STATIONS

Monografia apresentada à Universidade Federal de Lavras, como parte das exigências do Curso de Engenharia de Controle e Automação, para a obtenção do título de Bacharel.

APROVADA em 24 de Julho de 2023

Prof. Dr. Daniel Augusto Pereira UFLA

Prof. Dr. Thomaz Chaves de Andrade Oliveira UFLA

Prof. Dr. Wilian Soares Lacerda

(Orientador)

LAVRAS - MG
2023

À minha mãe Rosiane e meu pai Sibelmar por toda a confiança e coragem para me apoiar em minhas decisões e por serem os meus maiores motivos para correr atrás dos meus sonhos.

AGRADECIMENTOS

Agradeço acima de tudo a Deus por iluminar meu caminho diariamente e por me fortalecer diante dos desafios que eu não achava que conseguiria superar. Em segundo lugar, agradeço à minha família, especialmente aos meus pais, por sempre acreditarem em mim e me apoiarem em todas as minhas decisões. Por fim, expresso minha imensa gratidão aos meus professores, especialmente ao meu orientador Wilian Lacerda, pela paciência e interesse em colaborar comigo neste projeto.

Sou grato por todo o conhecimento, experiências e amizades que criei durante a minha graduação!

“O sucesso é constituído por 10% de inspiração e 90% de transpiração.”
(Thomas Edison)

RESUMO

O Brasil ocupa posição entre os maiores mercados de combustível no mundo. Por apresentar grande dispersão geográfica e mais de 40 mil postos de distribuição, a perspectiva é de que oportunidades surgem para novos entrantes, além de uma tendência de investimento em sistemas de automação, principalmente em empresas que possuem postos de distribuição de combustível internos. Partindo deste contexto, a seguinte monografia propõe o desenvolvimento de um protótipo baseado em um sistema embarcado com a finalidade de solucionar demandas de automação e monitoramento de bombas de abastecimento em postos internos de gestoras de frotas. O trabalho apresenta um estudo sobre as vantagens de um sistema baseado nos conceitos de telemetria e no paradigma da Internet das Coisas (IoT). O sistema entrega a possibilidade de supervisionar a quantidade de combustível que está sendo consumida, garantir informação sobre a autonomia média de cada veículo da frota e controlar eventos de reabastecimento. Além disso é uma alternativa menos custosa comparada aos sistemas de automação já existentes no mercado. O sistema foi construído baseando-se no kit de desenvolvimento ESP32 como núcleo, no qual acessa um servidor remoto por meio de APIs REST para trabalhar com dados dos abastecimentos e agir nos atuadores instalados na bomba de combustível. Portanto, os resultados obtidos resumem-se no desenvolvimento do sistema, viabilização com testes em laboratório e disponibilização dos abastecimentos aos gestores por uma aplicação web.

Palavras-chave: ESP32, Internet das Coisas, Telemetria, Automação, Bombas de combustível.

ABSTRACT

Brazil holds a significant position among the largest fuel markets in the world. Due to its wide geographic dispersion and over 40 thousand distribution stations, new opportunities arise for potential entrants, in addition to the tendency of investments in automation systems, especially in companies with internal fuel distribution stations. Based on this context, the following monograph proposes the development of a prototype based on an embedded system with the purpose of addressing automation and monitoring demands for fuel pumps in internal fleet management stations. The paper shows a study on the advantages of a system grounded in telemetry concepts and the Internet of Things (IoT) paradigm. The system offers the possibility to supervise fuel consumption, provide information on the average autonomy of each fleet vehicle and control refueling events. Additionally, it represents a cost-effective alternative when compared to existing automation systems in the market. The system was built using the ESP32 development kit as the core, enabling access to a remote server through REST APIs to work with supply data and interact with actuators installed in the fuel pump. As a result, the achieved outcomes encompass the system's development, laboratory testing for feasibility and the provision of supplies data to managers through a web application.

Keywords: ESP32, Internet of Things, Telemetry, Automation, Fuel pumps.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API	Application Programming Interface
GPIO	General Purpose Input/Output
GPRS	General Packet Radio Services
GPS	Global Positioning System
GSM	Global System for Mobile Communications
I2C	Inter-Integrated Circuit
ICONIX	Iconic Notation for Interfaces and Exchanges
IDE	Integrated Development Environment
IoT	Internet of Things
JSON	JavaScript Object Notation
LED	Light-Emitting Diode
REST	Representational State Transfer
RFID	Radio Frequency Identification
RTC	Real-Time Clock
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo são apresentadas a contextualização e motivação deste trabalho, além dos objetivos propostos, justificativa e como foi organizado todo o texto.

1.1 Contextualização e Motivação

O Brasil é o quarto maior mercado de combustível do mundo, chegando a movimentar anualmente mais de 400 bilhões de reais, localizando em seu território mais de 40.000 postos de combustíveis e 180 empresas distribuidoras (NETO, 2017). Segundo dados da Agência Nacional de Petróleo (ANP, 2021), o setor de distribuição de combustível, considerando a gasolina e o diesel, teve uma alta de 2020 para 2021 em 9,7%, registrando uma máxima de 39,3 bilhões de litros de gasolina e 62,1 bilhões de litros de óleo diesel. Em relação aos preços, os dois combustíveis obtiveram um aumento de 46%, onde a gasolina apresentou um preço médio de R$6,67 e o diesel de R$5,30.

Sabendo disso, e devido à diversos fatores geográficos e econômicos, juntamente aos efeitos da pandemia do COVID-19, o preço do combustível fóssil enfrentou altas e baixas até o ano de 2023, no qual o valor médio da gasolina alcançou R$5,63 por litro e do diesel de R$4,96 por litro (ANP, 2023). Conforme complementa Neto (2017), é notável que este fenômeno acarreta na entrada de novos distribuidores em busca de oportunidades e fortalece cada vez mais a monopolização do mercado por empresas já consolidadas.

Entretanto, as seguidas altas nos preços do combustível fóssil levantam constantes discussões sobre a necessidade de avaliar fontes alternativas para uso nos veículos, o que seria muito benéfico do ponto de vista ambiental. Dessa forma, mesmo com a alta no preço do combustível e com a grande pressão entre as maiores economias do mundo na tentativa de adotar o uso de biocombustíveis e carros elétricos, o combustível fóssil será uma realidade por muito tempo para várias empresas que atuam principalmente no ramo de transporte de cargas e aluguel de veículos, o que torna inevitável sua decisão em implantar um posto de distribuição de combustível internamente em suas instalações.

Os postos internos de distribuição de combustível não são diferentes de um posto de abastecimento de combustível comercial. Atualmente, no Brasil, há uma crescente onda de investimento pelas empresas de logística em postos de distribuição próprios, principalmente quando sua frota de veículos é extensa. Logo, muitas vantagens são garantidas com a sua implantação, pensando em controle de recursos e custos operacionais, visto que a operação de

abastecimento é realizada internamente na empresa.

Contudo, na maioria dos casos de implantação de um posto interno, as empresas ignoram o investimento em sistemas automatizados via software para monitorar todos os seus abastecimentos. Segundo a Confederação Nacional do Transporte (CNT, 2019), o diesel representa 35% do custo operacional de uma transportadora de cargas, o que para um gestor de frotas pode ser um problema se não gerenciar corretamente este recurso no seu negócio. Para algumas empresas, é indiferente ter um sistema de automação até que se tenha conhecimento dos benefícios que pode trazer, o que é difícil demonstrar pois há poucos casos de uma gestão eficiente com o mesmo. Para pequenas empresas, depende muito da relação custo-benefício, pois na maioria das vezes o sistema é um investimento muito alto para a escala da organização.

Entretanto, os sistemas automatizados são responsáveis por gerar eficiência e economia a partir de dados, nos quais são transformados em informações que realimentam a cadeia de processos da organização. Dessa forma, no caso de um posto interno de combustível, um gestor ou analista poderia acessar facilmente uma aplicação na web e, por meio de ferramentas de análise de dados, ter conhecimento dos abastecimentos realizados e mensurar eficiência e economia resultantes da gestão do combustível.

Sabendo disso, a eficiência na gestão de combustível existe a partir do momento que um sistema de monitoramento e medição é instalado nas bombas medidoras de combustível. Logo, é possível investigar todos os abastecimentos realizados e, além disso, é capaz de se conectar remotamente em um servidor de dados no qual o gestor conseguirá transformá-los em informações úteis a partir de planilhas e relatórios.

Complementando, a gestão se torna econômica quando as informações geradas pelo sistema contribuem para a tomada de decisão do gestor ou analista, visto que ele terá controle completo de todas as entradas, saídas e dos momentos de abastecimento de cada veículo da frota. Dessa forma, haverá uma maior visibilidade do negócio, pois a análise de dados possibilita ter um maior controle do processo, sendo possível identificar melhorias de desempenho e gastos desnecessários.

Nesse contexto, um sistema automatizado adota a estratégia de inserir algum recurso tecnológico dentro de um processo de forma a otimizar suas tarefas e agregar valor para a empresa. Assim, junto a atual transformação digital que o mundo está presenciando, a Internet das Coisas (IoT) é um paradigma de comunicação que vem ganhando muito espaço dentre os projetistas de equipamentos de telemetria, principalmente por inserir tecnologias acessíveis, escaláveis e de baixo custo. Apesar de ser um conceito recente, há uma comunidade dedicada

que mantém sua importância no ramo dos sistemas automatizados, visto que os objetos da vida cotidiana são capazes de se comunicar entre si e com usuários de toda parte do mundo, tornando-os partes ativas na Internet (ANDRADE et al., 2018). Dessa forma, os equipamentos instalados nas bombas de combustíveis se tornam auto sustentáveis, garantindo a capacidade de compartilhar dados dos abastecimentos entre si e enviá-los para uma plataforma de gestão remota.

Portanto, tem-se como motivação os benefícios de se implantar um posto de distribuição de combustível internamente em empresas de logística e as vantagens de se inserir um sistema de automação no processo da empresa. Logo, esse trabalho aborda o desenvolvimento de um equipamento de telemetria dentro da perspectiva de IoT, com a função de gerenciar os dados dos abastecimentos realizados na bomba de combustível. Basicamente, será utilizado o kit de desenvolvimento ESP32 para controlar o acesso do frentista na bomba por meio de um leitor de identificação RFID, no qual, em seguida, realiza automaticamente o desbloqueio da bomba de combustível acionando um relé. Como consequência, o desbloqueio do relé libera o usuário para iniciar o abastecimento do veículo. O ESP32 coleta todos os dados do abastecimento e aciona um módulo SIM800L para enviá-los via rede móvel para uma aplicação web que opera como interface de monitoramento e gestão.

1.2 Objetivo geral

O presente trabalho tem como objetivo o desenvolvimento de um protótipo de sistema baseado no conceito de telemetria e IoT para a automação das bombas de distribuição de combustível em postos internos através da utilização do kit de desenvolvimento ESP32.

1.3 Objetivos específicos

Para alcançar o objetivo geral apresentado na seção anterior foi necessário atender os seguintes tópicos:

- compreender os principais conceitos sobre sistemas embarcados, IoT e telemetria;
- implementar o projeto físico e eletricamente, sustentado tanto por ilustrações feitas em softwares de modelagem como em protótipos montados em bancada;
- realizar a montagem do equipamento e realizar testes em laboratório;

- desenvolver o software que será executado pelo ESP32 bem como a aplicação web que opera como interface de monitoramento.

1.4 Justificativa

A decisão por escolher o kit de desenvolvimento ESP32 juntamente aos módulos de comunicação como alternativa para o sistema de automação nos postos internos se fundamenta principalmente no baixo custo de desenvolvimento, escalabilidade e pela versatilidade do projeto. Sabendo disso, a priori, o equipamento tem como objetivo entregar uma gestão de combustível mais eficiente, garantindo-se por ser uma opção mais barata, acessível e flexível para alterações e melhorias. Logo, um projeto de automação com este sistema embarcado supre perfeitamente a ideia de necessidade com simplicidade, sabendo do seu potencial para agregar valor para as empresas em seus postos internos de combustível.

1.5 Organização do texto

Este trabalho é desenvolvido em 5 capítulos, sendo eles:

- Capítulo 1: durante a introdução é apresentada a motivação para a escolha deste tema e como ele está vinculado ao contexto atual do Brasil. Além disso, são esclarecidos os objetivos que se pretende alcançar com o desenvolvimento da solução em geral e específicos;
- Capítulo 2: no referencial teórico discorre-se sobre as definições e conceitos que fundamentam o embasamento teórico do trabalho. O texto é desenvolvido com a finalidade de simplificar e entregar conhecimento suficiente para entender a metodologia;
- Capítulo 3: em materiais e métodos é apresentado o tipo da pesquisa, os materiais utilizados e a metodologia adotada no desenvolvimento do hardware e software, além de como os conceitos da teoria se contemplam com os objetivos;
- Capítulo 4: em resultados são esclarecidos quais objetivos foram alcançados ao seguir a metodologia apresentada, assim como uma breve discussão;
- Capítulo 5: por fim na conclusão são apresentadas as considerações finais, recapitulando os objetivos que foram alcançados, algumas propostas de trabalhos futuros e contribuições.

2 REFERENCIAL TEÓRICO

Neste capítulo, serão expostas as áreas de conhecimento que servirão de base para o desenvolvimento do projeto. Portanto, serão apresentadas as referências teóricas que fundamentam a implementação do sistema de automação utilizando o ESP32, enfatizadas por discussões de vários autores, além também da compreensão dos equipamentos dimensionados, os sistemas de telemetria, sensoriamento e monitoramento.

.

2.1 Sistemas Embarcados

Segundo Heath (2003), sistema embarcado é um sistema baseado em um microprocessador no qual é desenvolvido para controlar uma função ou um conjunto de funções, não sendo designado para ser programado pelo usuário final. Ou seja, é designado para executar uma determinada tarefa sem a possibilidade de modificar seu software.

Complementando, Wilmshurst (2007) define sistema embarcado como um sistema no qual a sua principal função não é computacional, mas o mesmo é controlado por um computador embarcado dentro dele. Basicamente, eles são formados por um processador, memória, periféricos, firmware e um software programado, entretanto diferem de um computador tradicional pelo simples fato de executarem uma única tarefa ao invés de multitarefas (CHASE, 2007).

Assim, o responsável por desenvolver sistemas embarcados ou utilizar um modelo do mercado tem como objetivo pensar em como as interfaces de entrada e saída, periféricos, poder de processamento, memória, ambiente de aplicação e sistemas auxiliares serão aproveitados, enquanto que o usuário final se preocupa em como o sistema vai agregar valor à determinada aplicação, em termos de redução de custos, funcionalidade e desempenho.

2.1.1 Microcontroladores

Os microcontroladores podem ser considerados como um sistema independente com um processador, memória e periféricos onde na maioria dos casos é necessário somente adicionar um software quando implementado em sistemas embarcados (HEATH, 2003).

Além disso, Wilmshurst (2007) aponta que os microcontroladores são uma adaptação dos microprocessadores para tarefas onde não se exige poder computacional, grandes quantidades de memória ou alta velocidade de processamento, mas a necessidade de executar

algum controle ou interação com o ambiente externo. Consequentemente, os microcontroladores evoluíram para tamanhos menores, a um custo-benefício satisfatório e se adaptando a ambientes cada vez mais extremos, tomando a responsabilidade de um mini computador nos sistemas embarcados.

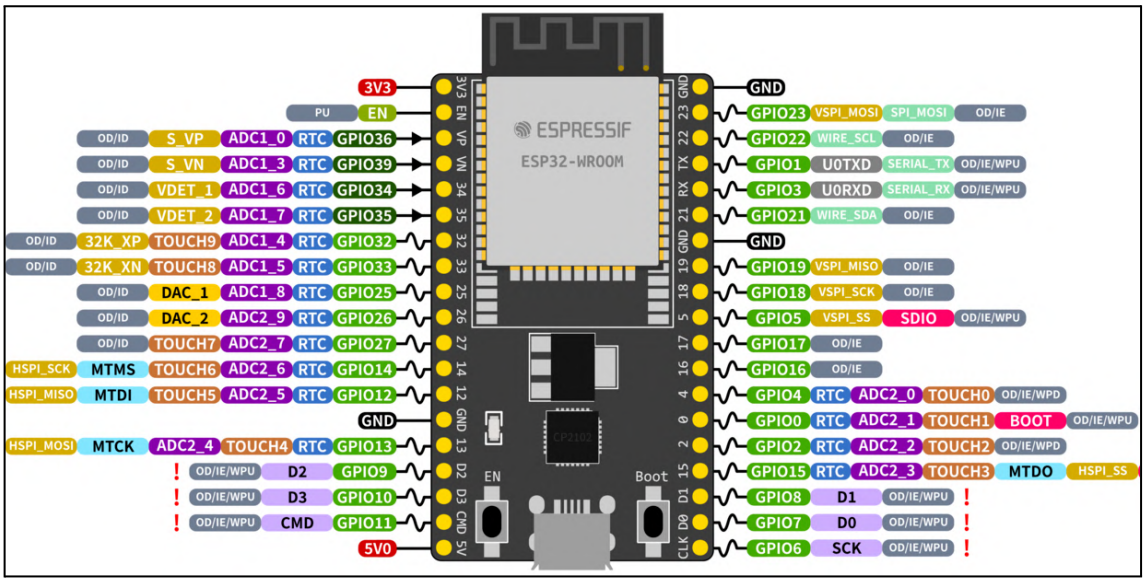
2.1.2 Interfaces de entrada e saída

Conforme apresentado por Balachandran (2009), as interfaces de entrada e saída estão disponíveis nos microcontroladores mais modernos para garantir uma forma fácil de acessar suas propriedades internas. Basicamente, são um conjunto de pinos com a capacidade de enviar e receber sinais elétricos sem um propósito em específico.

Os microcontroladores apresentam diversas vantagens na construção de circuitos eletrônicos, devido aos baixos custos de projeto, consumo de energia e praticidade quanto a programação e reposição. Entretanto, sua principal característica é a integração com diversos componentes por conta das interfaces de entrada e saída (OLIVEIRA, 2017).

Dessa forma, as interfaces mais comuns são as *General Purpose Inputs/Outputs* (GPIO), projetadas para serem configuradas via software e receber níveis de tensão que assumem valores 0, baixo, ou 1, alto (OLIVEIRA, 2017). É possível visualizar pela Figura 2.1 as GPIOs presentes em um módulo ESP32-WROOM-32, onde esta característica permite aplicações simples como receber sinais de um sensor ou dar partida em um motor. Além disso, alguns microcontroladores apresentam interface de entrada analógica, no qual assume-se uma tensão de entrada variável que é convertida em um intervalo de valores proporcionais.

Figura 2.1 - Detalhes das GPIOs do módulo ESP32-WROOM-32 no DevKitC da Espressif Systems.



Fonte: Espressif Systems (2023).

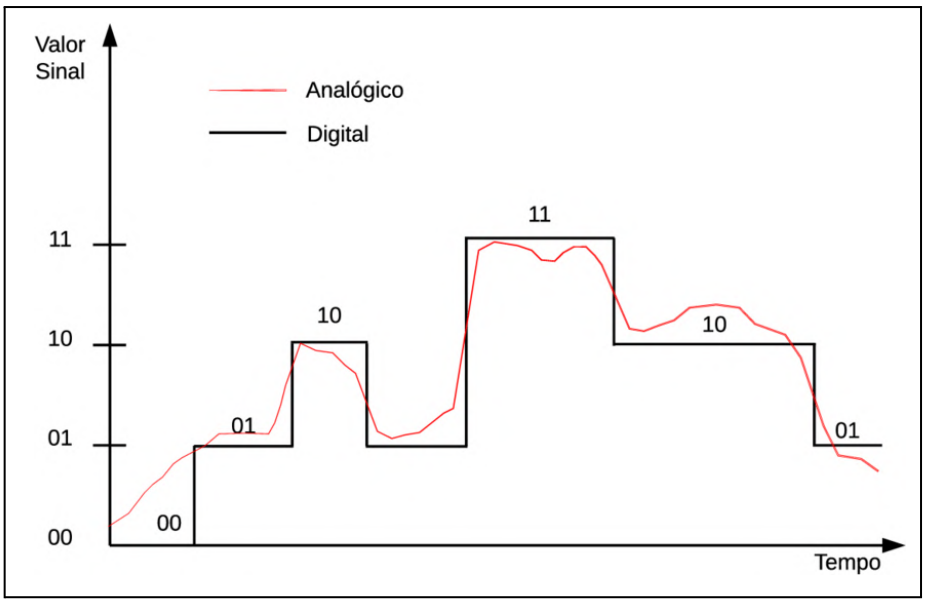
2.1.3 Sinais digitais e analógicos

Os fenômenos no mundo real podem ser mensurados. Assim, de certa forma, é possível caracterizá-los como um tipo de sinal analógico, por se tratar de algo contínuo no tempo. Como exemplo tem-se a intensidade luminosa, temperatura, umidade, pressão e assim por diante (OLIVEIRA; ANDRADE, 2010).

Entretanto, estes sinais são digitalizados quando devem ser manipulados por computadores e sistemas embarcados, no qual a transformação se baseia em uma sequência discreta, ou descontínua, no tempo e definida para determinados instantes dependendo da referência de bits de conversão (OLIVEIRA; ANDRADE, 2010). A Figura 2.2 exemplifica como um sinal analógico pode ser representado após uma conversão, onde o resultado pode ser observado a partir de cada degrau.

Logo, em projetos de sistemas embarcados que envolvam coletas de grandes volumes de dados é indispensável ter módulos conversores para o tratamento dos sinais mensurados no ambiente ou dos sinais que trafegam pelas GPIOs. Os conversores são do tipo analógico para digital (A/D), onde a conversão é para digital, ou digital para analógico (D/A), onde há uma transformação para analógico.

Figura 2.2 - Representação da conversão de um sinal analógico em um sinal digital.



Fonte: Adaptado de Oliveira e Andrade (2010).

2.1.4 Interfaces de comunicação

Segundo IDEALI (2021), as interfaces de comunicação são protocolos pré-definidos com a função de criar uma rede de conexão entre o microcontrolador e outros componentes. Em sistemas embarcados, os protocolos são utilizados principalmente para garantir o uso de poucas portas de entradas e saídas, ou GPIOs, além de que alguns tipos de sensores por padrão requerem o uso de uma interface de comunicação para facilitar a integração por parte do projetista.

A rede de comunicação é importante em sistemas mais complexos que envolvam vários sensores e processadores, onde é possível criar uma interface serial com fio ou sem fio e com transmissão síncrona ou assíncrona. Dentre os protocolos de comunicação serial mais comuns nos sistemas embarcados encontram-se o *Inter-Integrated Circuit* (I2C), *Serial Peripheral Interface* (SPI), *Universal Asynchronous Receiver/Transmitter* (UART) e *One Wire* (1-WIRE) (IDEALI, 2021), com algumas de suas características resumidas na Tabela 2.1.

Tabela 2.1 - Característica e aplicação dos protocolos de comunicação.

Protocolo	Direção	Comunicação	Sentido	Taxa [kbps]	Arquitetura	Aplicação
I2C	Bidirecional	Síncrona	Half duplex	400	Mestre/ escravo	Vários mestres e escravos, até 127 dispositivos, curtas distâncias
SPI	Unidirecional	Síncrono	Full duplex	3000	Ponto a ponto	Complexo de conectar, para um mestre e vários escravos, curtas distâncias
UART	Unidirecional	Assíncrono	Full duplex	115	Ponto a ponto	Simples de conectar, para somente dois dispositivos, longas distâncias
1-WIRE	Bidirecional	Assíncrono	Half duplex	16	Mestre/ escravo	Simples de conectar, para um mestre e vários escravos, longas distâncias

Fonte: Autor (2023)

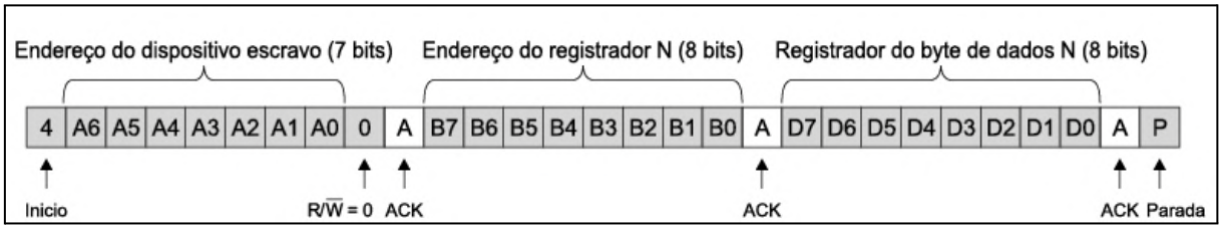
2.1.4.1 I2C

A comunicação ou interface serial I2C é um protocolo presente nos microcontroladores para integrar periféricos de baixa velocidade, desenvolvida na década de 1980 pela Phillips, (OLIVEIRA, 2017). Segundo Oliveira (2010) e Almeida (2016), suas principais características são:

- O protocolo provê duas vias bidirecionais: *Serial Data* (SDA) para dados seriais e *Serial Clock* (SCL) para sincronismo.
- O sentido é *Half-duplex*, ou seja, o remetente da informação não pode receber e enviar dados ao mesmo tempo.
- A comunicação é bidirecional e síncrona.
- Suporta por padrão taxas de comunicação de 100kbps a 400kbps. É possível alcançar taxas de 5Mbps em modelos mais recentes do protocolo.
- Trabalha em tensões de 0V a 5V ou 3,3V, dependendo do microcontrolador.
- Arquitetura do tipo mestre/escravo, ou seja, permite mais de um dispositivo escravo no barramento, sendo este com um identificador único.

Segundo IDEALI (2021), na transmissão de dados, a comunicação é sempre iniciada pelo dispositivo master, no qual envia pacotes em grupos de 8 bits mais dois bits que marcam o início, ou *Start*, e o fim, ou *Stop*, da transmissão, conforme demonstrado na Figura 2.3. Durante a comunicação, o master também envia um byte para identificar com qual escravo quer se comunicar, indicado pelo *Address Frame*. Em seguida, um bit é enviado para identificar um processo de escrita ou leitura, sendo este o *Read/ Write Bit*. Na operação de escrita um novo byte é enviado com os dados e na operação de leitura o byte é enviado pelo escravo, ambos indicados pelos *Data Frames*. Além disso, cada frame é acompanhado por um *Acknowledgement Bit* (ACK) ou por um *Negative Acknowledgement Bit* (NACK). Estes bits são respostas do dispositivo escravo para identificar o sucesso da transmissão.

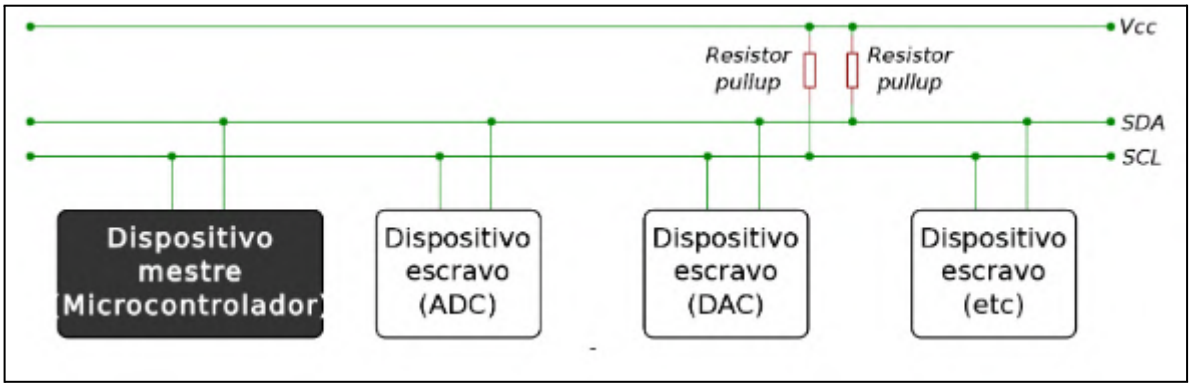
Figura 2.3 - Formato da mensagem de escrita em um barramento do padrão I2C.



Fonte: Ideali (2021).

A Figura 2.4 ilustra um exemplo de conexão pelo barramento I2C com três dispositivos escravos. É notável a presença de resistores de *pull-up* na via SDA para manter o barramento em nível lógico alto justamente porque o bit de início é identificado a partir de um nível lógico baixo, ou seja, o mestre altera o valor do terminal de alto para baixo aterrando o sinal.

Figura 2.4 - Exemplo de conexão com um mestre e três escravos.



Fonte: Almeida (2016).

2.1.4.2 1-WIRE

O protocolo de comunicação 1-WIRE tem um conceito similar ao I2C, porém com taxas de transmissão de dados menores e maior alcance. O padrão foi desenvolvido pela Dallas Semiconductor (Maxim Integrated Products Inc., Califórnia, Estados Unidos) com o objetivo de permitir a construção de sensores, atuadores e entre outros dispositivos periféricos com o uso mínimo de recursos de hardware e software (PEREIRA, 2009).

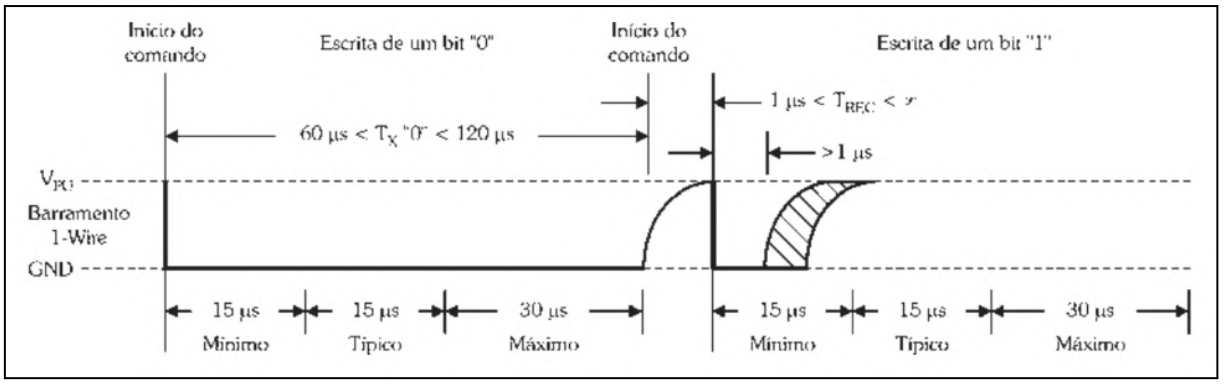
Assim, conforme demonstra Pereira (2009), algumas de suas principais características são:

- Utiliza-se somente uma via bidirecional para comunicação.
- O sentido é *Half-duplex*, ou seja, o remetente da informação não pode receber e enviar dados ao mesmo tempo.
- A comunicação é bidirecional e assíncrona.
- A velocidade máxima de comunicação é de 16kbps para o modo normal e 143kbps para o modo *Overdrive*.
- Trabalha em tensões de 0V a 5V ou 3,3V, dependendo do microcontrolador.
- Arquitetura do tipo mestre/escravo, no qual permite a conexão de vários dispositivos escravos no mesmo barramento.

Sabendo disso, seu princípio de funcionamento se baseia em uma conexão de dreno-aberto. Isso significa que deve existir um resistor de *pull-up* para garantir nível lógico alto para que os dispositivos conectados possam levar o sinal para nível baixo e iniciar uma tentativa de comunicação. Cada dispositivo é identificado com um número único de 64 bits e pode ser alimentado a partir do próprio barramento, portanto é necessário uma outra via para aterramento do circuito (PEREIRA, 2009).

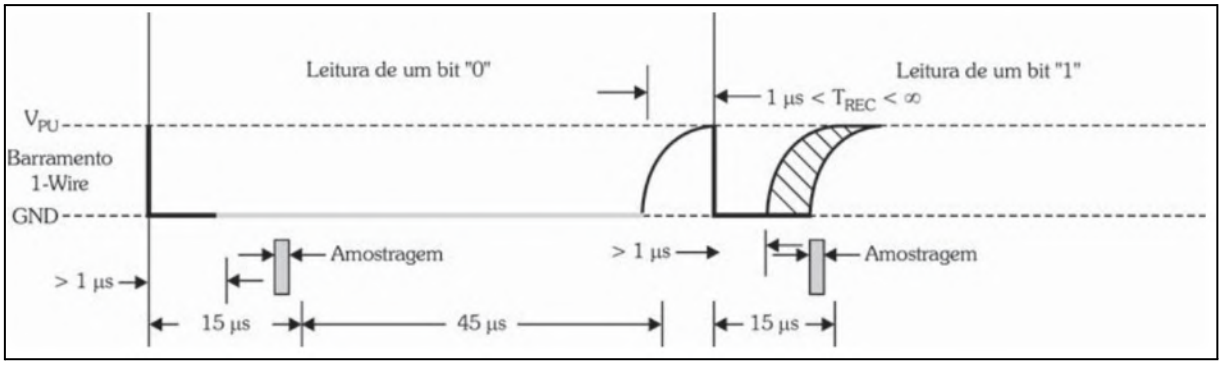
Durante a transmissão de dados, o mestre sempre inicia a comunicação no circuito levando o barramento para nível lógico zero. Basicamente, um bit é comunicado segundo o intervalo de tempo que o nível lógico se mantém, conforme pode ser observado na Figura 2.5 onde para escrever o bit 1 deve-se deixar em nível lógico zero por 15µs e o bit 0 por 60µs. A Figura 2.6 ilustra o processo de leitura, onde um amostrador (em cor cinza) é passado no sinal em cada comando após 15µs que o sinal foi enviado justamente para identificar o bit 1 ou bit 0. A duração de cada comando deve estar entre 60µs e 120µs (MAXIM INTEGRATED, 2002).

Figura 2.5 - Comportamento de um sinal de escrita de bit 1 e bit 0 no barramento 1-WIRE.



Fonte: Pereira (2009).

Figura 2.6 - Processo de leitura do bit 1 e bit 0 no barramento 1-WIRE.



Fonte: Pereira (2009).

2.1.4.3 UART

A comunicação UART é o protocolo mais conhecido e utilizado na grande maioria dos microcontroladores. Normalmente é usada em situações com pouca transferência de informação ou como mecanismo de depuração dos programas presentes nos sistemas embarcados. Estes programas conseguem enviar as mensagens de depuração pela UART implementada na *Universal Serial Bus* (USB) até a aplicação que consegue interpretar a comunicação serial, como por exemplo um computador *desktop* (OLIVEIRA, 2010).

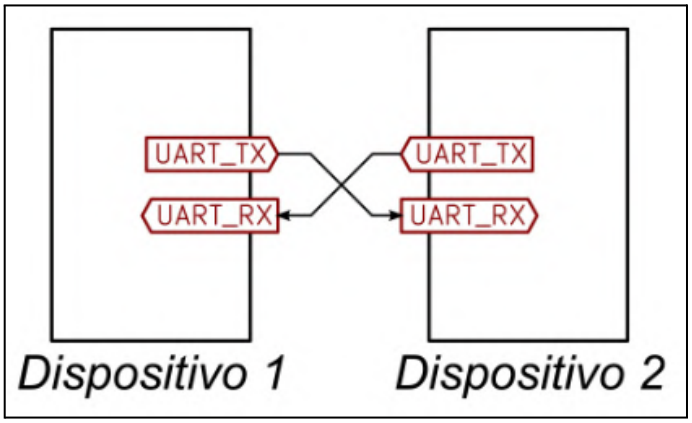
Conforme apresentado por Oliveira (2010) e Almeida (2016), algumas de suas principais características são:

- O protocolo provê duas vias unidirecionais: *Receiver* (RX) para receber dados e *Transmitter* (TX) para enviar dados.

- O sentido é *Full-duplex*, ou seja, o remetente da informação pode receber e enviar dados ao mesmo tempo.
- A comunicação é unidirecional e assíncrona.
- Suporta por padrão taxas de comunicação até 115,2kbps.
- Trabalha em tensões de 0V a 5V ou 3,3V, dependendo do microcontrolador.
- Arquitetura do tipo ponto a ponto, ou seja, permite somente dois dispositivos no barramento.

À vista disso, para criar uma conexão pela UART é necessário que os terminais RX e TX do primeiro dispositivo estejam conectados, respectivamente, nos terminais TX e RX do segundo dispositivo, conforme apresentado na Figura 2.7.

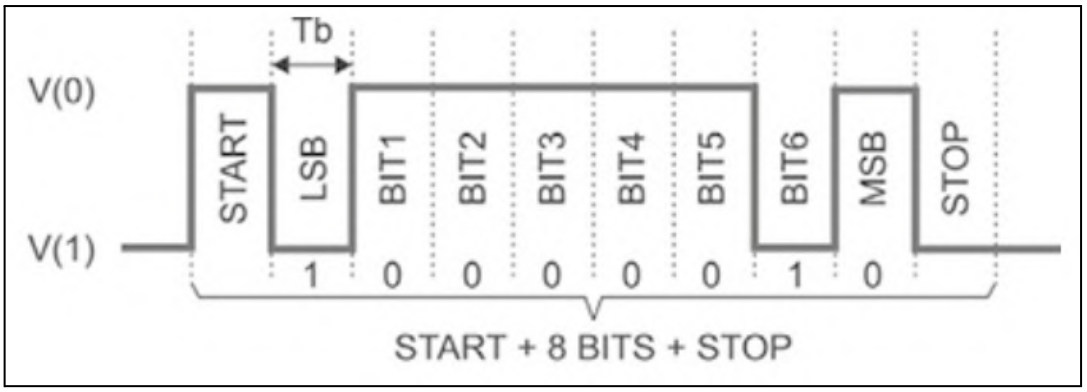
Figura 2.7 - Conexão feita entre dois dispositivos no protocolo UART.



Fonte: Almeida (2016).

A Figura 2.8 demonstra o formato do pacote que é transmitido durante a conexão. Os dados transmitidos são organizados a partir de um bit de início ou *Start*, um byte de dados, ou *Data Frame*, um bit de paridade e um bit de parada, ou *Stop*. O pacote é enviado a partir de uma operação de *bit-shift* até que o bit mais significativo (de parada) seja enviado e finalize a transmissão (ALMEIDA, 2016).

Figura 2.8 - Formato do pacote em um barramento do padrão UART.



Fonte: Ideali (2023).

2.1.4.4 SPI

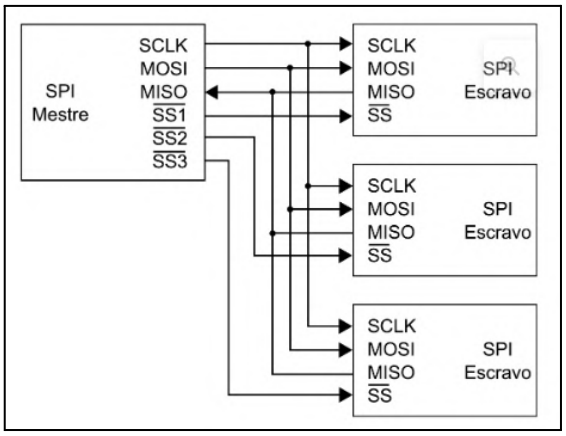
A comunicação ou interface serial SPI é um padrão presente na maioria dos sistemas embarcados, comumente destinada para integrar dispositivos em uma distância curta, prezando principalmente por uma maior velocidade de transmissão e flexibilidade no volume da cadeia de dados (ALMEIDA, 2016). Desenvolvida pela Motorola (Motorola Inc., Illinois, Estados Unidos) em 2000, as interfaces que implementam o SPI tem as seguintes características, conforme comenta Almeida (2016):

- O protocolo provê quatro vias unidirecionais: *Serial Clock* (SCLK, SCK ou CLK) para sincronismo, *Master Out Slave In* (MOSI, SDO, DO ou SO) como a via de transmissão para o mestre, *Master In Slave Out* (MISO, SDI, DI ou SI) como a via de transmissão para o escravo e *Slave Select* (SS ou CS) para selecionar o escravo.
- O sentido é *Full-duplex*, ou seja, o remetente da informação pode receber e enviar dados ao mesmo tempo.
- A comunicação é unidirecional e síncrona.
- Suporta por padrão taxas de comunicação de até 3Mbps.
- Trabalha em tensões de 0V a 5V ou 3,3V, dependendo do microcontrolador.
- Arquitetura do tipo mestre/escravo, no qual permite a conexão de vários dispositivos escravos em um mesmo barramento.

A comunicação SPI traz consigo a mesma ideia de implementar uma comunicação serial para diversos dispositivos, ou seja, não há a necessidade de estender comunicações em paralelo. Dessa forma, para utilizar o protocolo deve-se conectar os pinos MISO, MOSI e SCLK com os respectivos terminais do módulo a ser utilizado. Assim, quando a quantidade

de dispositivos no barramento aumenta, um terminal a mais para a seleção de módulo SS precisa ser projetado, conforme a Figura 2.9.

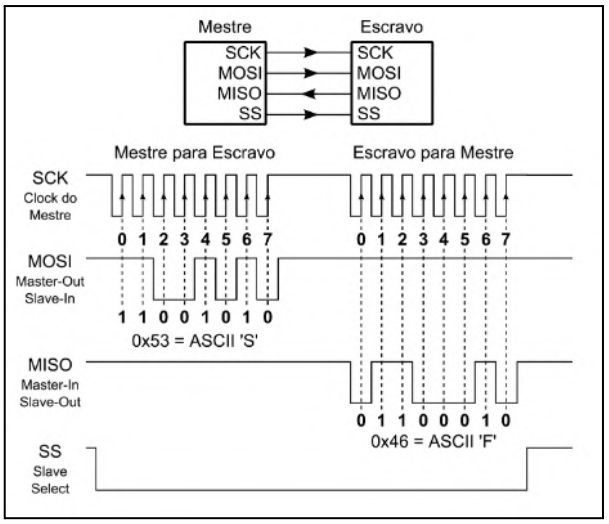
Figura 2.9 - Conexão dos terminais para três dispositivos escravos independentes em um barramento que implementa o SPI.



Fonte: Almeida (2016).

Portanto, o processo de transmissão de dados funciona por *bit-shift* da mesma forma que o protocolo UART, com a diferença de que os dispositivos mestre e escravo transmitem, simultaneamente, um bit por vez para cada pulso de *clock*. Ou seja, sempre que o mestre transmite um bit, obrigatoriamente o escravo deve transmitir um bit, mesmo que a informação seja irrelevante (ALMEIDA, 2016). A Figura 2.10 ilustra o funcionamento da comunicação, onde o mestre é responsável por configurar uma frequência para o sinal de *clock* (SCK) e também pela seleção do dispositivo escravo (SS) mantendo o terminal em nível lógico baixo. Em seguida, o byte desejado é enviado pela via MOSI até o escravo, no qual retorna seus dados pela via MISO na mesma sequência de pulsos de *clock* do mestre.

Figura 2.10 - Funcionamento da comunicação no protocolo SPI.



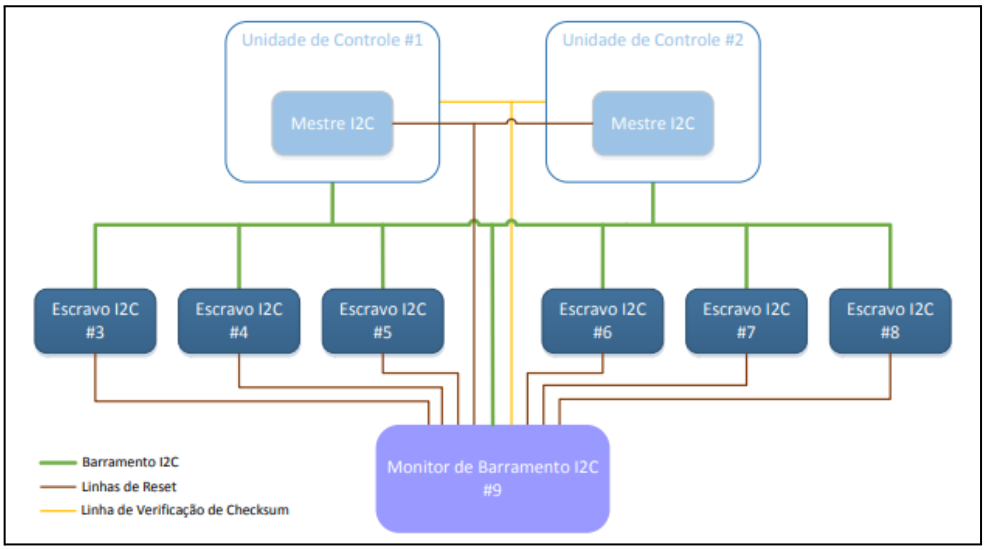
Fonte: Almeida (2009).

2.1.5 Master / Slave

A configuração *master/slave* (mestre/escravo) é um paradigma da programação paralela no qual possui sua classe de algoritmos e controle baseado em duas entidades distintas: o dispositivo com o processo mestre e o conjunto de dispositivos com o processo escravo. Por se tratar de um modelo que consegue entregar bom desempenho e elevada escalabilidade, é possível encontrar muitos sistemas de controle que adotam este paradigma como arquitetura de comunicação (ROCHA, 2008).

A Figura 2.11 demonstra o exemplo de uma aplicação comum da arquitetura para uma malha com dois microcontroladores mestre e vários dispositivos que recebem tarefas do mestre. Conforme é ilustrado, os dispositivos estão conectados por um barramento baseado no protocolo I2C, onde o mestre fica responsável por centralizar o controle, manter o sincronismo, iniciar a comunicação com os escravos, delegar as tarefas e recolher os resultados gerados, enquanto que os escravos obtém e processa a tarefa para assim retornar com uma resposta ao mestre pelo mesmo barramento (ALMEIDA, 2016).

Figura 2.11 - Configuração master/slave em uma malha com dois mestres e seis escravos. O barramento é conectado a um monitorador.



Fonte: Carvalho (2016).

2.2 Sensores

Segundo a definição de Wendling (2010), o termo sensor é utilizado para designar dispositivos que são sensíveis a algum determinado tipo de energia do ambiente, no qual pode ser luminosa, térmica, cinética, sonora, entre outras formas. As informações são relacionadas para determinar as grandezas físicas que precisam ser mensuradas como temperatura, pressão, velocidade, corrente, aceleração, posição e assim por diante.

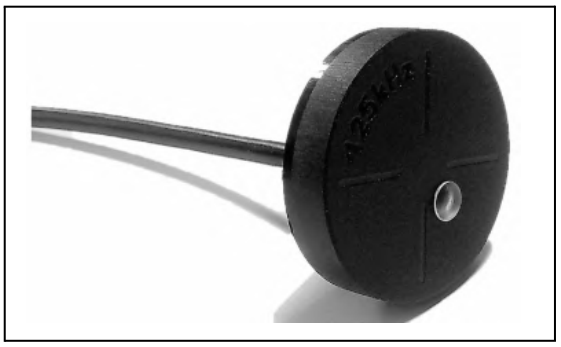
Sabendo disso, Thomazini e Albuquerque (2020) cita que é essencial ter atenção na seleção de um tipo de sensor de acordo com a necessidade da aplicação e suas propriedades, como o tipo de saída discreta ou analógica, sensibilidade, exatidão, velocidade de resposta, alcance, linearidade, entre outras características que determinam o desempenho do sistema.

2.2.1 Leitor RFID Dréxia

O leitor de *Radio Frequency Identification* (RFID) modelo 1W-H3-05 produzido pela empresa Dréxia (Drexia, Lódz, Polônia), conhecido popularmente como leitor Dréxia, é um módulo/sensor IoT utilizado na identificação sem fio de *transponders* passivos como cartões e tags. O módulo, ilustrado na Figura 2.12, é muito utilizado em aplicações voltadas para gerenciamento de frotas e controle de acesso, principalmente porque sua implementação é

baseada no protocolo 1-WIRE. Consequentemente não há complexidades para configurá-lo (DREXIA, 2022).

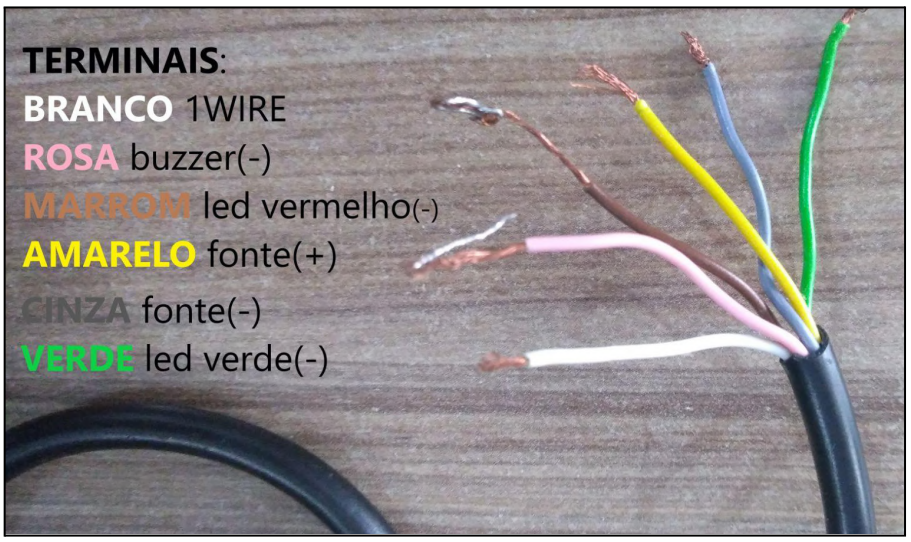
Figura 2.12 - Representação de um leitor RFID Dréxia.



Fonte: Dréxia (2022).

Em relação às suas características elétricas, o módulo apresenta tensão nominal de 12V, um pico de corrente de 160mA e identifica dispositivos na frequência de 125kHz por meio do protocolo EM4100. Além do terminal de alimentação, de aterramento e de dados utilizado pelo protocolo 1-WIRE, há dois *Light-Emitting Diodes* (LEDs) e um *Buzzer* embutidos que podem ser utilizados simplesmente conectando seus contatos ao terminal negativo de uma fonte de alimentação (DREXIA, 2022). Seus terminais estão indicados pela Figura 2.13.

Figura 2.13 - Terminais em um sensor RFID Dréxia.

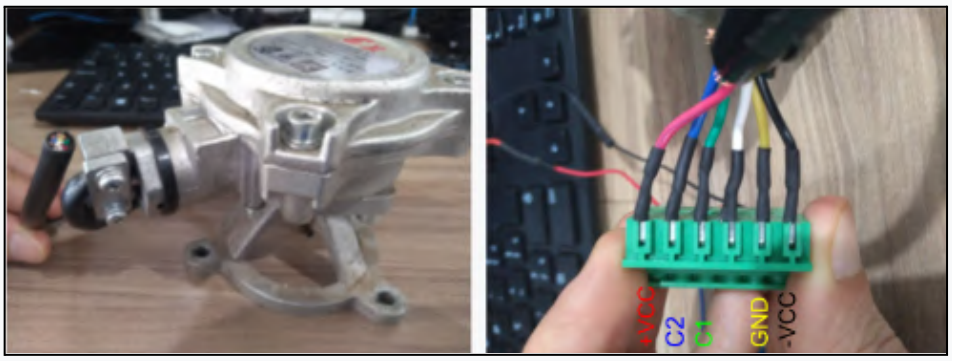


Fonte: Autor (2023).

2.2.2 Gerador de pulsos FBCGQ-3

Segundo a documentação disponibilizada pela Exzotron Technology (EXZOTRON TECHNOLOGY, 2018), o sensor FBCGQ-3 é um gerador de pulsos elétricos utilizado comumente em bombas de combustível das empresas Topaz e the Shelf, porém pode ser também encontrado em outras marcas de bombas. À esquerda da Figura 2.14 tem-se uma imagem apresentando o sensor isolado e à direita os seus terminais de conexão.

Figura 2.14 - À esquerda o sensor FBCGQ-3 e à direita seus terminais de conexão.



Fonte: Autor (2023).

Os modelos desta categoria de sensor foram projetados para operar especificamente em conjunto com sistemas de bombas de combustível. Ele é instalado acima do bloco medidor de combustível e é um dispositivo essencial para o funcionamento adequado do sistema de bombeamento e cálculo da saída de fluido, pois sua função é converter a rotação do eixo mecânico do bloco medidor de combustível em pulsos elétricos sucessivos. Ou seja, a partir da lógica de programação presente na *Central Processing Unit* (CPU) da bomba de combustível, é possível resgatar os pulsos elétricos pelos terminais C1/C2 do sensor e calcular a quantidade de combustível que sai pelos bicos injetores da bomba. A metodologia que se adota para o cálculo da saída de combustível pode ser tanto por calibração ou por meio do ângulo de rotação do eixo gerador (EXZOTRON TECHNOLOGY, 2018).

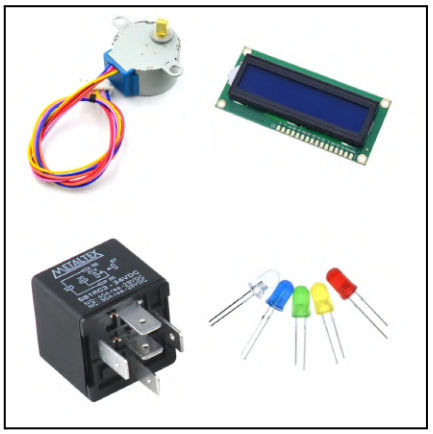
Além disso, o sensor tem a capacidade de gerar 100 pulsos por revolução, onde cada pulso tem nível lógico entre 0V e 5V, com uma corrente elétrica máxima de 20mA. Dessa forma, o seu terminal pode ser facilmente interceptado e conectado em uma CPU externa, como por exemplo em um kit de desenvolvimento ESP32 ou Arduino UNO, para determinar a quantidade de combustível que está saindo da bomba desde o início da retirada (EXZOTRON TECHNOLOGY, 2018).

2.3 Atuadores

Os atuadores são dispositivos que realizam movimentações ou atuam a partir de sinais de controle em um determinado processo. Pode ser operado a partir de energia ou por fluido pressurizado onde há a conversão em movimentos ou força. Segundo Thomazini e Albuquerque (2020), a principal função de um atuador é modificar o estado de uma variável controlada, ou seja, recebe um sinal proveniente de um controlador e age sobre o sistema controlado.

No escopo de projetos com sistemas embarcados é comum encontrar motores elétricos de corrente contínua ou de passo, servomotores, lâmpadas, LEDs, displays gráficos, sinalizadores sonoros, relés, entre outros dispositivos. A finalidade é atuar de alguma forma no ambiente externo, conforme exemplos apresentados na Figura 2.15. Com os atuadores combinados aos sensores, um microcontrolador e às redes de computadores, é possível detectar, monitorar e modificar qualquer variável do processo (ELETRONICS, 2020).

Figura 2.15 - Exemplos de atuadores.



Fonte: Autor (2023).

2.3.1 Relé para bloqueio

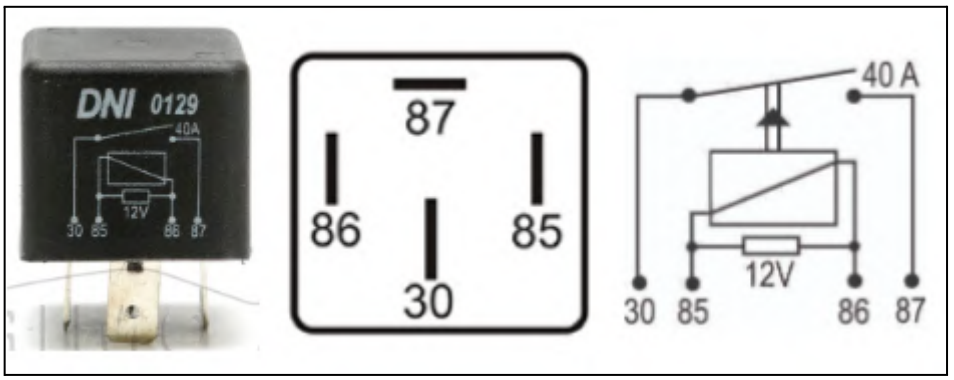
O relé é um dispositivo formado por uma chave eletromecânica que converte energia elétrica em energia mecânica. A chave apresenta dois estágios distintos: um normalmente aberto e outro normalmente fechado. Assim que a bobina do relé é alimentada, um campo eletromagnético é gerado, induzindo a troca de estado da chave. O relé é uma alternativa

robusta para interfaces que necessitam suportar uma carga de tensão e corrente elétrica elevada (OLIVEIRA, 2010).

Sabendo disso, o relé de bloqueio é um componente comumente encontrado em circuitos de partida de automóveis. O bloqueio ocorre quando o motorista não é autorizado a conduzir o veículo. A ativação do dispositivo pode se exercer de duas formas distintas: a partir de um comando remoto ou por algum leitor de identificador único dentro do veículo.

Conforme é utilizado como solução para gerenciamento de frotas, é possível adaptá-lo em outras situações onde o objetivo é realizar um bloqueio ou desbloqueio de acesso. A Figura 2.16 apresenta o relé auxiliar modelo DNI0129 da empresa DNI com a finalidade de ser conectado tanto em motores de veículos à diesel quanto em bombas de abastecimento de combustível. O módulo apresenta tensão nominal de 12V e corrente máxima de 40A, sendo o mais comum de se encontrar no mercado (DNI, 2020).

Figura 2.16 - Relé auxiliar modelo DNI0129 e seus terminais de conexão.



Fonte: DNI (2020).

2.3.2 Buzzer

O *Buzzer* é um pequeno alto-falante aplicado em projetos que seja necessário emitir sinais e alertas sonoros. O pequeno dispositivo possui uma célula piezoelétrica que vibra de acordo com o sinal elétrico que é aplicado em seus terminais. Esta vibração mecânica é responsável por gerar as ondas sonoras (GABRIEL; KURIA, 2020).

Assim, com o advento dos microcontroladores, o *buzzer* se tornou um atuador padrão em kits de desenvolvimento e também em alguns sensores, como os leitores RFID. Em alguns casos quando a saída do microcontrolador não tem a capacidade de fornecer corrente suficiente para o *buzzer*, é possível utilizar um circuito de amplificação (ALMEIDA, 2016).

No mercado é comum encontrar modelos entre 3V a 24V, além de serem pequenos, baratos e consumirem valores de corrente elétrica inferiores a 12mA, conforme modelo da Figura 2.17.

Figura 2.17 - Exemplo de um dispositivo *Buzzer*.

Fonte: Autor (2023).

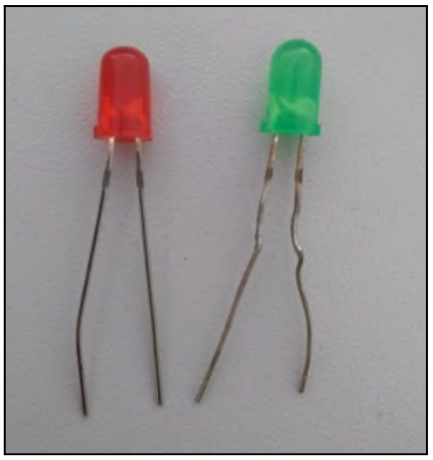
2.3.3 LED

Para entender o conceito de LED, é essencial ter conhecimento do funcionamento de um diodo. Os diodos são componentes semicondutores formados por germânio ou silício, no qual têm a característica de conduzir corrente elétrica em um sentido (polarização direta) e barrar a corrente elétrica se a condução for no sentido oposto (polarização inversa), devido às suas propriedades (ALMEIDA, 2016).

Dessa forma, os LEDs são diodos que quando polarizados diretamente são capazes de emitir luz. A emissão acontece pois a corrente elétrica percorre a junção PN, no qual emite radiação infravermelha. A radiação que é transformada em luz visível está relacionada diretamente à estrutura atômica dos materiais semicondutores que compõem o LED. Logo, essa compreensão permite a sua fabricação dentro de uma ampla gama de cores (ALMEIDA, 2016).

A Figura 2.18 apresenta exemplos de LEDs coloridos comumente encontrados no mercado de projetos eletrônicos, inclusive muitas soluções envolvendo sistemas embarcados utilizam frequentemente os LEDs como mecanismo de alerta, atualizar sobre o estado do sistema, ou até mesmo indicar se o hardware está em operação. Este componente pode ser encontrado em diferentes diâmetros, alimentado por tensões entre 3,3V e 5V, podendo consumir até 18mA (ALMEIDA, 2016).

Figura 2.18 - Exemplos de LEDs de 5mm com coloração vermelha e verde.



Fonte: Autor (2023).

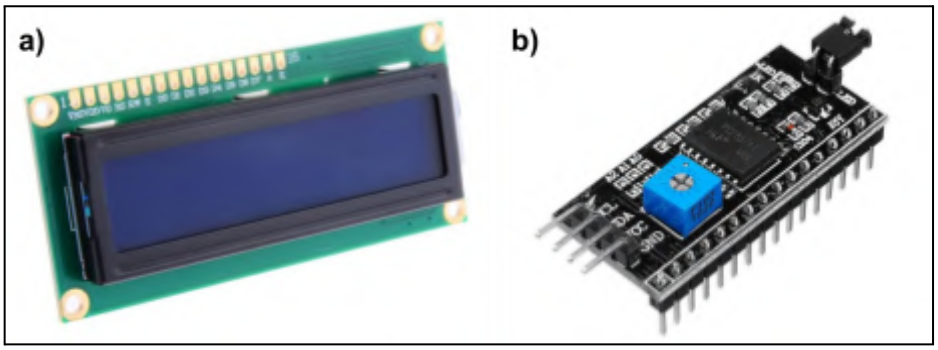
2.3.4 Módulo display LCD

O display LCD é um dispositivo atuador essencial em sistemas que visam apresentar dados ou informações em tempo real ao usuário. Por ser um equipamento simples e prático, ele é utilizado em muitos projetos de IoT e telemetria para notificar o status do sistema visualmente, possibilitando apresentar uma quantidade de dados relativamente grande (PEREIRA, 2009). A Figura 2.19 apresenta um modelo de módulo display LCD 16x2 (16 colunas e 2 linhas), no qual é constituído de um display de cristal líquido, seu chip controlador e um módulo adaptador I2C.

O display de cristal líquido pode operar no formato de caracteres ou gráficos, sendo o primeiro formato mais comum pela capacidade de apresentar letras, símbolos e números, distribuídos em suas linhas e colunas. Para isso, os controladores são utilizados para criar interfaces com sistemas externos, no qual entregam, por padrão, 8 barramentos de dados *data 0* (D0) até *data 7* (D7) e 3 linhas de controle *enable* (EN), *read/write* (R/W) e *register selector* (RS), totalizando 11 terminais de comunicação. Dessa forma, é interessante adicionar o módulo adaptador I2C para diminuir a quantidade dos barramentos do controlador para dois terminais SDA e SCL, consequentemente simplificando a comunicação (PEREIRA, 2009).

Segundo sua folha de dados, o display LCD pode ser alimentado com níveis de tensão entre 3V e 5V, podendo consumir até 16mA se a luz de fundo estiver ativada (XIAMEN AMOTEC, 2008).

Figura 2.19 - Módulo display LCD 16x2, onde estão presentes: a) cristal líquido e terminais do chip controlador, b) módulo adaptador I2C.



Fonte: Autor (2023).

2.4 Módulos externos

Uma das formas de ampliar as possibilidades de um projeto voltado para sistemas embarcados se baseia na adição de módulos externos. Conceitualmente, esse módulo é um dispositivo eletrônico construído em circuitos impressos com a finalidade de criar uma interface entre o sistema embarcado e o periférico a ser conectado. Em muitos casos o circuito elétrico dos módulos é implementado para tornar a conexão entre sistema embarcado e periférico compatível, segura ou para simplificar algum barramento (OLIVEIRA; ANDRADE, 2010).

Sabendo disso, em projetos de IoT é muito comum encontrar módulos para comunicações com ou sem fio que facilitam a prototipagem e o controle de acesso na rede. As tecnologias *General Packet Radio Services* (GPRS), *Global System for Mobile Communication* (GSM) e *Global Positioning System* (GPS) são exemplos de redes acessadas a partir da interface de um módulo externo, bastando somente adequar o circuito elétrico e o microcontrolador necessários para o funcionamento adequado destas tecnologias (NASUTION et al., 2017).

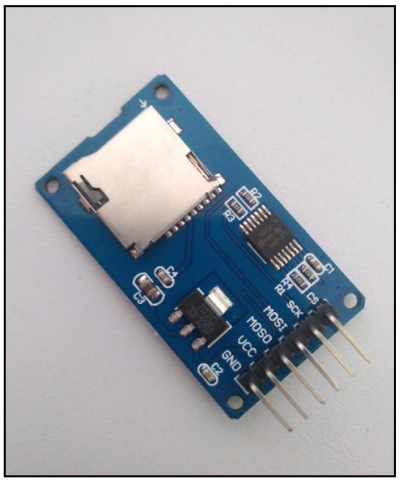
2.4.1 Módulo cartão microSD

Os cartões de memória microSD são memórias flash eletrônicas utilizadas como dispositivos de armazenamento de dados digitais como por exemplo arquivos de texto, imagens e vídeos. A sua memória flash garante velocidades de até 25MBps na gravação ou

leitura de informações, além de reter dados sem energia e disponível em diferentes capacidades de armazenamento (WESTERN DIGITAL, 2022).

Conforme demonstrado pela Figura 2.20, os cartões microSD utilizam a interface de comunicação SPI em sistemas embarcados, sendo sua finalidade expandir a capacidade de armazenamento do microcontrolador e ser utilizado como servidor de arquivos. No escopo de IoT, um cartão de memória garante que arquivos temporários, drivers, aplicativos e até sistemas operacionais possam ser armazenados e acessados remotamente dependendo da aplicação. O módulo suporta alimentação de 5V e pode consumir até 80mA durante sua operação (OLIVEIRA, 2010).

Figura 2.20 - Módulo para cartão microSD que utiliza a interface SPI.



Fonte: Autor (2023).

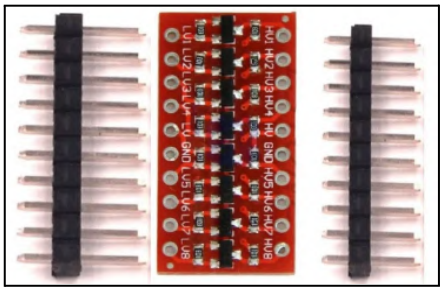
2.4.2 Módulo conversor de nível lógico

O módulo conversor de nível lógico bidirecional de uso geral é um circuito elétrico utilizado para intermediar uma conexão entre dispositivos que trabalham com níveis lógicos diferentes, conforme o indicado na Figura 2.21. No escopo de sistemas embarcados é padrão relacionar o nível lógico alto a tensões de 5V ou 3,3V, sendo que nem todos os embarcados têm compatibilidade para identificar os dois níveis de tensão ao mesmo tempo (SPARKFUN ELECTRONICS, 2022).

Logo, o conversor é indispensável quando há a necessidade de criar uma conexão com fio entre um dispositivo que identifica nível lógico alto em 5V e outro que identifica nível lógico alto em 3,3V. Por exemplo, não é recomendado interligar um sensor que atua em

intervalos de 5V a um ESP32 que trabalha com tensões de no máximo 3,3V. A sugestão é adicionar o conversor no barramento (FAIRCHILD SEMICONDUCTOR, 2005).

Figura 2.21 - Módulo conversor de nível lógico bidirecional.

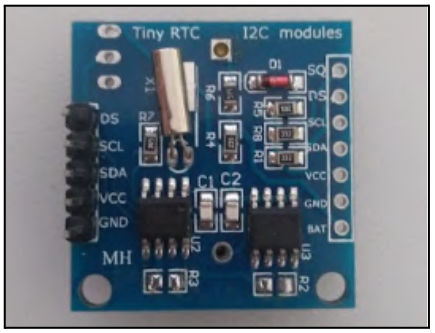


Fonte: Sparkfun Electronics (2022).

2.4.3 Módulo relógio de tempo real

A Figura 2.22 apresenta o módulo *Real-Time Clock* (RTC) DS1307, o qual é um mini relógio em tempo real de baixo custo. O módulo tem um suporte para encaixar uma bateria com o objetivo de manter a precisão do relógio enquanto o sistema principal não estiver alimentando o dispositivo. Em projetos de sistemas embarcados é interessante ter controle da data e horário, pois em uma situação crítica de falta de energia, o módulo entra em ação mantendo as informações em tempo real de segundos, minutos, horas, dia, mês e ano. Da mesma forma, o RTC enriquece um projeto de IoT e telemetria devido a possibilidade de manter atualizado as configurações de horário e garantir o monitoramento constante dos eventos que ocorrem no software implementado. Além disso, este módulo pode ser alimentado com tensões entre 3,3V e 5V, consumindo até 0,3mA de corrente elétrica (MAXIM INTEGRATED, 2015).

Figura 2.22 - Módulo RTC DS3231.

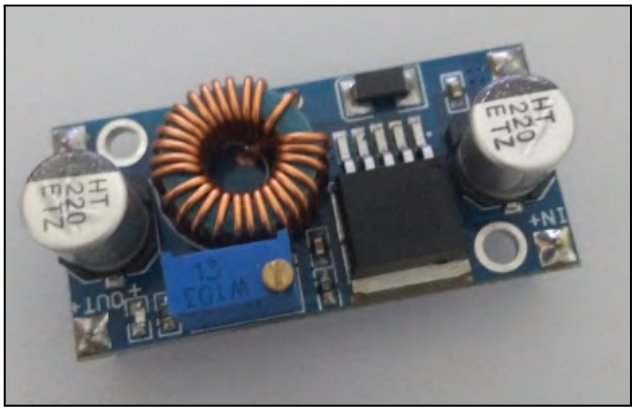


Fonte: Autor (2022).

2.4.4 Módulo regulador de tensão

É comum desenvolver projetos com sistemas embarcados associando um módulo regulador de tensão, devido à necessidade de energizar os dispositivos com uma fonte de alimentação que entrega tensões acima do limite que os mesmos suportam. Assim, a Figura 2.23 apresenta o módulo com o microcontrolador XL4005, no qual é um conversor de modo *step-down* de corrente contínua (DC) comumente aplicado nestas situações. O módulo conduz cargas de até 5A com eficiência independente do nível de tensão regulado, compatível com tensões de entrada entre 5V e 32V e de saída entre 0,8V e 30V, garantindo sua principal característica de simplificar a regulação de tensão com o uso mínimo de componentes (XLSEMI, 2020).

Figura 2.23 - Módulo regulador de tensão XL4005.



Fonte: XLSEMI (2020).

2.4.5 Módulo GPS

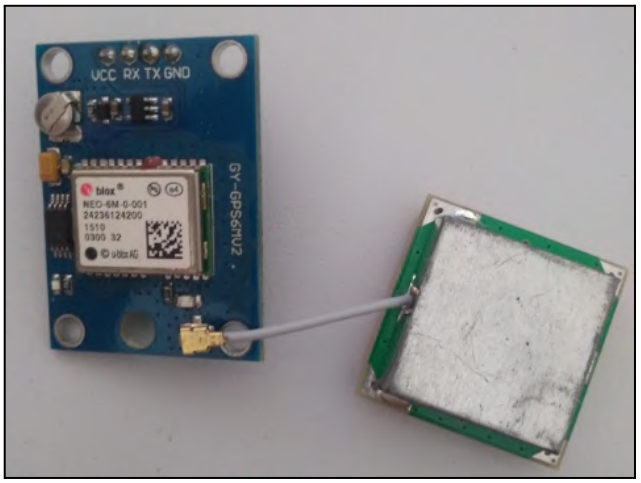
A tecnologia GPS foi desenvolvida com objetivo de localizar com exatidão a posição em latitude, longitude e altitude de qualquer lugar no planeta, além de ser possível determinar a hora exata com precisão entre 5ns a 60ns. Com as informações de posição e tempo, é possível calcular a velocidade e trajetória de algum elemento rastreado (TRINDADE, 2014).

Sabendo disso, o GPS é constituído de um conjunto de satélites, receptores de sinais via rádio e estações de monitoramento e controle. Ao manter uma comunicação, os satélites são responsáveis por emitir um sinal aos receptores GPS com o horário exato em que o sinal foi transmitido. Assim, o receptor consegue calcular a distância que se encontra de cada

satélite e determinar a sua posição pelo processo de triangulação, gerando assim informações de três dimensões: norte, leste e altitude (KAPLAN; HEGARTY, 2005).

Portanto, a tecnologia que envolve os conceitos do GPS pode ser aplicada em projetos voltados para IoT e telemetria. Um sistema embarcado pode facilmente ser rastreado pela rede mundial de satélites a partir de um módulo GPS, onde o receptor entrega ao microcontrolador toda informação de localização via interface serial UART. As informações são repassadas como texto dentro do protocolo *National Marine Electronics Association* (NMEA), no qual sua interface é simples de ser interpretada em um debugador de códigos. A Figura 2.24 ilustra um módulo GPS disponível no mercado e que pode ser integrado com diferentes modelos de sistemas embarcados. Este módulo pode ser inserido facilmente em um sistema embarcado via barramento UART, suporta uma alimentação de 5V e consome até 45mA em operação (OLIVEIRA, 2017).

Figura 2.24. Módulo uBlox NEO 6M com antena.



Fonte: Autor (2023).

2.4.6 Módulo GSM/GPRS

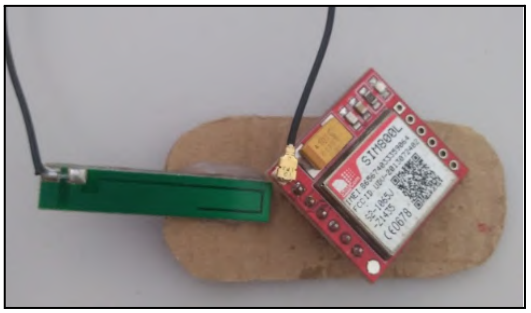
A tecnologia GSM é o sistema de comunicação portátil mais popular para os telefones celulares atualmente, sendo desenvolvida para possibilitar a troca de informações de serviços de voz e mensagens dentro de um canal digital. O canal estabelece uma comunicação ininterrupta entre dois dispositivos celulares onde a transmissão dos dados é feita a partir da comutação de circuitos, ou seja, todo recurso do canal era dedicado somente para aquela conexão criada (TEIXEIRA, 2018).

Assim, a tecnologia GPRS surgiu como uma solução para beneficiar a rede GSM com os protocolos da rede Internet, no qual toda a informação era fragmentada desde a origem, direcionada pelo melhor caminho e até ser remontada no destino pela comutação em pacotes. A vantagem disso é que agora todos os usuários da rede podem compartilhar de um mesmo canal de comunicação para enviar e receber dados. É comum se referir a tecnologia GPRS como GSM/GPRS ou geração 2.5G pois a mesma utiliza a estrutura previamente montada da rede GSM (ROCHOL, 2018).

Logo, é visível a importância da tecnologia GSM/GPRS em projetos IoT e de telemetria, pois ela garante que sistemas complexos e operacionais sem fio possam compartilhar dados com maior controle e confiabilidade à uma base servidora na nuvem. Apesar de já estar ultrapassada devido a ascensão de redes 3G, 4G e 5G, ainda é muito utilizada no ramo da telemetria por questões de custo e benefício.

A Figura 2.25 apresenta o módulo com o chip SIM800L. O dispositivo adota a tecnologia GSM/GPRS e pode ser controlado por diversos modelos de microcontroladores a partir de uma cadeia de comandos *ATtention* (AT). Este dispositivo pode consumir até 2A de corrente elétrica em seu uso extremo, devendo ser alimentado com tensões entre 3,7V e 4,3V (SIMCOM, 2022).

Figura 2.25 - Módulo SIM800L GSM/GPRS com antena.



Fonte: Autor (2023).

2.4.6.1 Comandos AT

Um comando AT é definido como uma sequência de caracteres que começa com o prefixo AT e termina com algum comando que se relaciona a uma operação de discagem, desligamento ou alteração de parâmetros de conexão. O padrão foi introduzido em 1977 pela empresa Hayes Communication (Hayes Communication Inc., Quebec, Canadá) e após sua consolidação no mercado, os comandos podem ser utilizados para programar módulos de

comunicação que são conectados em microcontroladores via barramento serial UART (GERIGAN; OGRUTAN, 2011).

As comunicações que utilizam tecnologia GSM/GPRS adotaram os princípios dos comandos AT para programar os módulos que mantêm essa comunicação, sendo que o portfólio de comandos pode variar dependendo do fornecedor. Dessa forma, por exemplo, os módulos da série SIM800L implementam um padrão de comandos AT a partir de uma combinação das normas definidas pela *3rd Generation Partnership Project* (3GPP) e pela *International Telecommunication Union - Telecommunication Sector* (ITU-T) (SIMCOM, 2015). No geral, a sintaxe de um comando AT tem o formato "AT<x><n>" onde o "<x>" é o comando e "<n>" o argumento do comando. Outros formatos de comandos podem ser executados conforme os modelos da Tabela 2.2.

Tabela 2.2 - Principais formatos de comandos executáveis nos módulos SIM800L.

Função	Comando	Descrição
Teste	AT+<x>=?	O equipamento móvel devolve a lista de parâmetros e intervalos de valores definidos para o comando de escrita correspondente ou processo interno.
Leitura	AT+<x>?	Retorna o intervalo de valores possíveis para o parâmetro(s).
Escrita	AT+<x>=<...>	Permite ao usuário definir os valores de entrada para os parâmetros.
Execução	AT+<x>	Lê parâmetros não variáveis que são afetados por processos internos do módulo GSM.

Fonte: Adaptado de SIMCom (2015).

2.5 ESP32-WROOM-32

O ESP32-WROOM-32 é um *System-on-a-Chip* (SoC) da série de microcontroladores de baixo custo desenvolvido pela Espressif Systems (Espressif Systems, Shanghai, China). Segundo a documentação disponibilizada pelo fabricante, o sistema se baseia em um microcontrolador de 32 bits de baixo consumo de energia, *dual-core* e com algumas funcionalidades como *WI-FI* e *Bluetooth* 4.2. Seu design foi pensado para ser seguro, performático e entregar possibilidades quanto aos seus pinos de propósito geral, sendo uma ótima escolha para projetos baseados em IoT e automação de processos (ESPRESSIF SYSTEMS, 2023).

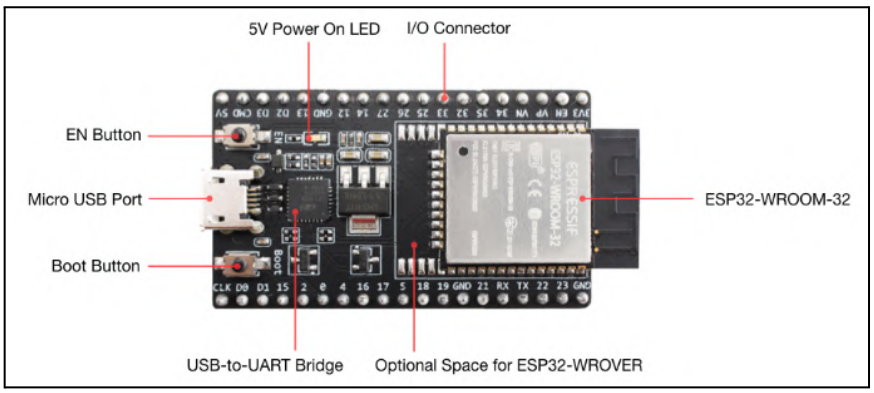
Por conta disso, a Espressif Systems (2023) deixa claro o foco em apresentar um produto seguro com protocolos de criptografia de dados e proteção contra ataques de injeção, incluindo um rico conjunto de periféricos com GPIOs programáveis, o que possibilita fornecer interfaces seriais 1-WIRE, I2C, SPI, UART e USB.

São inúmeras as características que tornam o microcontrolador o coração de muitos projetos. Dessa forma, a Espressif Systems desenvolve para o mercado modelos do ESP32-WROOM-32 em um kit de desenvolvimento conhecido como ESP32 *DevKit*, justamente para facilitar o desenvolvimento de interfaces de comunicação e principalmente para a prototipagem. Conforme a Figura 2.26, o kit contém em uma placa dois botões de *reset* e *boot*, conectores de entrada/saída, indicadores de alimentação e uma porta micro USB de conexão em uma máquina para fazer *upload* de programas.

A lista de projetos aplicáveis envolvendo o ESP32 DevKit é exaustiva, sendo os mais conhecidos (ESPRESSIF SYSTEMS, 2023):

- Automação residencial no controle de iluminação, fechamento inteligente de portas, janelas e monitoramento de energia;
- Automação industrial na robótica, controle wireless de processos;
- Automação agrícola nas irrigações e estufas inteligentes.

Figura 2.26 - Kit de desenvolvimento ESP32 com o módulo ESP32-WROOM-32 soldado e algumas interfaces da placa.



Fonte: Espressif Systems (2023).

2.6 Plataforma de desenvolvimento de software

As plataformas de desenvolvimento, também conhecidas como ambientes de desenvolvimento integrado (IDEs), são softwares de programação de alto nível que combinam ferramentas comuns de desenvolvimento dentro de uma interface gráfica para o usuário final (OLIVEIRA, 2017). O objetivo das IDEs é entregar simplicidade e resultados rápidos no desenvolvimento de aplicações.

No escopo de sistemas embarcados é possível encontrar diversas opções de IDE, inclusive empresas especializadas em soluções embarcadas já disponibilizam ambientes próprios como a Arduino IDE para placas Arduino e a ESP-IDF para placas ESP32 e ESP8266.

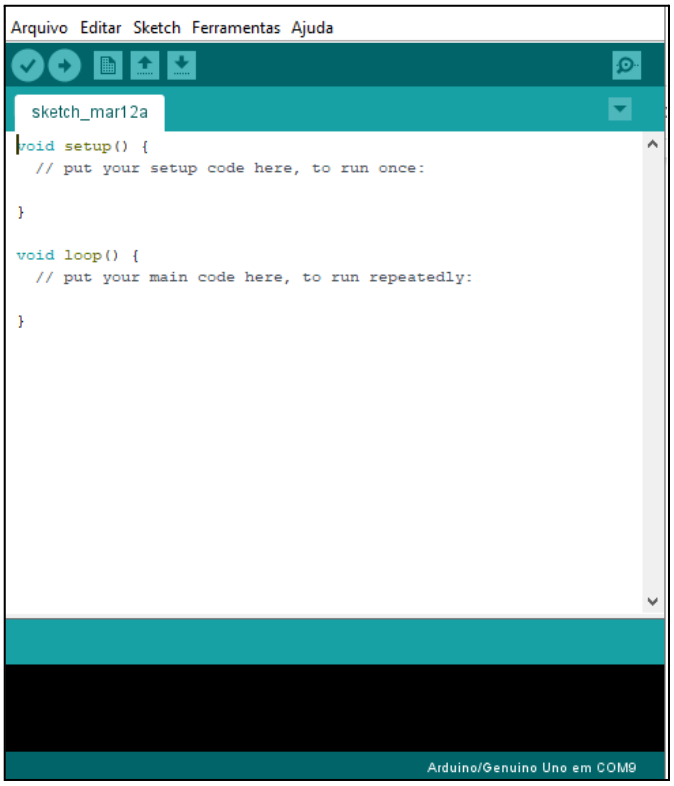
2.6.1 Arduino IDE

A Figura 2.27 ilustra o ambiente de desenvolvimento integrado (IDE) *open-source* fornecido pela marca Arduino , sendo a mais conhecida entre todos os entusiastas por microcontroladores e eletrônica. Oliveira (2017) comenta que o impulso da indústria eletrônica na China e o lançamento de hardwares com código aberto transformaram a Arduino IDE na principal plataforma a receber atualizações e bibliotecas de inúmeras empresas que lançam constantemente tecnologias voltadas para eletrônica e automação.

Assim, a acessibilidade se tornou sua principal característica, tão notável que alguns sistemas embarcados como a placa ESP32 podem receber upload de códigos e são compatíveis com a maioria das bibliotecas disponíveis na plataforma.

Todo tipo de funcionalidade, biblioteca e código é escrito em um subconjunto da linguagem C++, compilado em um bloco monolítico e gravado na memória do microcontrolador. A vantagem desse processo é entregar maior poder de processamento, deixando o processador totalmente dedicado (OLIVEIRA, 2017).

Figura 2.27 - Ambiente de Desenvolvimento Arduino IDE.



Fonte: Arduino Docs [1] (2023).

2.6.2 Visual Studio Code

O *Visual Studio Code* [2] (VSCode) é um editor de código-fonte que entrega muitos benefícios, principalmente por ser um software livre de código aberto, leve e robusto. Ou seja, além do seu editor de código com assistência para autocomplemento e realce de linhas, o

[1] Arduino IDE. **Download**. Disponível em: https://www.arduino.cc/en/software. Acesso em: 28 jun. 2023.

[2] Visual Studio Code. **Download Visual Studio Code - Mac, Linux, Windows**. Disponível em: https://code.visualstudio.com/download. Acesso em: 25 jun. 2023.

software tem extensões para várias linguagens de programação, com terminal integrado para uso de ferramentas de depuração e instalação de bibliotecas externas (MICROSOFT, 2022).

Segundo sua documentação disponibilizada pela Microsoft (2022), o VSCode tem suporte integrado para aplicações em *Node.JS*. Logo, é comum que muitas aplicações fundamentadas no *JavaScript* (JS) sejam desenvolvidas neste editor de código, como por exemplo as aplicações em *React.JS*, *jQuery* ou APIs como *Apexcharts* (análise de dados) e *LeafLet* (mapas interativos).

Além disso, é possível instalar *plug-ins* para gerenciamento de projetos e extensões que dão suporte para outras linguagens de programação, aproximando-o de uma IDE, o que justifica seu uso na criação da grande maioria dos softwares da comunidade de tecnologia (MICROSOFT, 2022).

2.6.3 C++

A linguagem de programação C++ é uma versão aprimorada da linguagem C, ocupando sua posição entre as mais populares e utilizadas na área de ciências da computação. Segundo Barros (2022), sua versatilidade a torna uma excelente opção na criação de softwares para máquinas, videogames, navegadores web e softwares gráficos.

Além disso, o C++ tem um perfil mais rápido e potente em relação às outras linguagens de programação, no qual permite aos desenvolvedores criar aplicações com ótimo desempenho. Adicionalmente, ela combina características de linguagens de baixo e alto nível, sendo ideal para implementar softwares em sistemas embarcados e microcontroladores. O C++ é uma linguagem de programação multiparadigma, ou seja, suporta programação imperativa, genérica e orientada a objetos (JOSUTTIS, 2013).

Dentre seus prós pode-se citar os seguintes:

- Código *open-source*;
- Versátil, entrega muita liberdade e controle, o que justifica sua aplicação em softwares de computadores;
- É tão eficiente quanto seu antecessor C, porém com novas funcionalidades;
- Pode ser utilizada em ambientes de desenvolvimento com poucos recursos.

Segundo Barros (2022), dentre as linguagens de programação mais utilizadas no mercado em 2022 estão Python e C++. Logo, não é de se esperar que alguns sistemas embarcados sejam compatíveis com a linguagem C++, no qual apresentam suas características particulares para tornar o desenvolvimento de programas mais intuitivo aos projetistas.

2.6.4 React JS

A área de desenvolvimento de software está evoluindo exponencialmente no sentido de exigir aplicações baseadas na web cada vez mais complexas e escaláveis. Conforme os dados apresentados pela NPM TRENDS (2023), é possível encontrar diversas tecnologias que se fundamentam nos conceitos do JS, *Cascading Style Sheets* (CSS) e *HyperText Markup Language* (HTML) para a criação de aplicações *front-end* na web, sendo que a biblioteca React.JS é atualmente a mais utilizada pelos desenvolvedores.

Segundo a documentação disponibilizada pela META OPEN SOURCE (2023), React.JS tem sua principal vantagem a possibilidade de construir interfaces por meio de componentes e sua eficiência na atualização das páginas web. Logo, cada componente pode ser combinado com outros e formar novas telas, páginas ou aplicações que são renderizadas rapidamente em produção.

Esta característica é muito importante pois permite que os componentes tenham responsabilidades únicas e sejam reaproveitados, ao ponto de conseguir abstrair as representações dos objetos que compõem os conteúdos em uma página web (AGGARWAL, 2018). Dessa forma, pensando em um projeto envolvendo IoT e telemetria, a experiência de desenvolvimento das interfaces em aplicações web se torna muito mais confortável e performática com React.JS, o que é interessante em sistemas onde o objetivo é implementar telas mais simples sem ter que se preocupar com a arquitetura dos documentos da web, possibilitando que se invistam maiores esforços no desenvolvimento das outras etapas do projeto de telemetria.

2.7 Computação em nuvem

O conceito de computação em nuvem pode ser compreendido de diversas formas, pois é uma área onde existe muita curiosidade e pouca informação sobre o assunto. Segundo Taurion (2009), muitos definem a computação em nuvem como a descrição sintetizada de um ambiente de computação baseado em uma imensa rede de servidores virtuais ou físicos. Esses servidores são tratados como o ponto final dentro de uma nuvem, pois são eles que entregam recursos de computação como processamento, plataformas, armazenamento e serviços na Internet.

Sabendo disso, em um cenário onde a Internet começava a ganhar espaço, as empresas que queriam se conectar precisavam investir pesadamente em recursos de infraestrutura para

instalar servidores internos, principalmente porque havia uma grande necessidade de armazenar informações e gerenciar bancos de dados. Entretanto, com o tempo estes serviços começaram a ser terceirizados em data centers externos trazendo muita vantagem competitiva para as empresas (OLIVEIRA, 2017). Conforme aponta Taurion (2009), as principais características que levaram a essa adoção foram:

- A acessibilidade sob demanda, trazendo a ilusão de que os recursos são infinitos;
- A eliminação da necessidade de adquirir recursos internos;
- Elasticidade, no qual permite utilizar os recursos na quantidade que for necessária;
- A possibilidade de pagar os serviços pela quantidade de recursos utilizados, conhecida como *pay-per-use*.

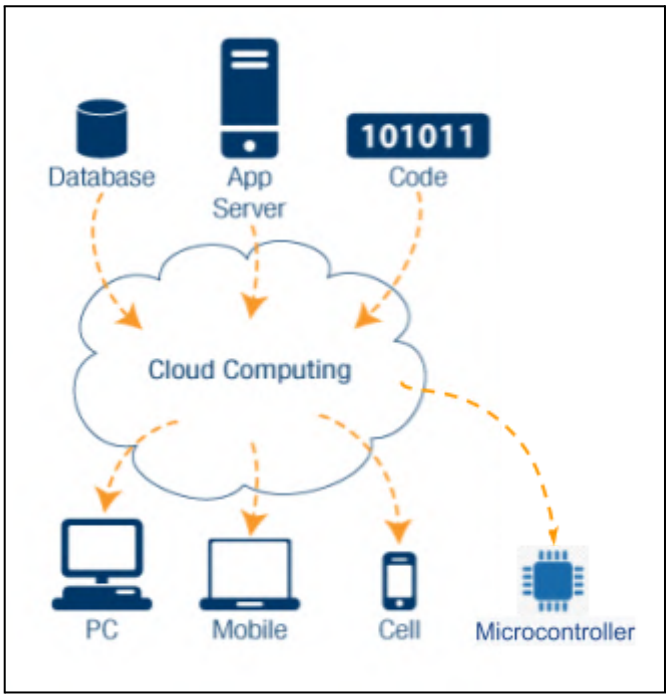
Além disso, o conceito de nuvem ficou mais presente na sociedade com a popularidade dos serviços de armazenamento e processamento na rede, sendo os mais comuns o *Google Drive* da empresa Google para armazenar arquivos e o *PostGreSQL* como sistema gerenciador de bancos de dados.

Para aplicações envolvendo IoT, a nuvem é um fator indispensável. Oliveira (2017) comenta que:

> "Os dispositivos IoT normalmente têm poder de processamento limitado, baixa confiabilidade e alta taxa de erros. Aplicações críticas não devem ficar hospedadas em dispositivos IoT. Esses dispositivos devem enviar suas informações para um elemento centralizador, com maior poder de processamento e disponibilidade".

Portanto, caso esse elemento centralizador esteja na nuvem, ganha-se muito em questão de escalabilidade, segurança e flexibilidade. A Figura 2.28 exemplifica bem a ideia da computação em nuvem como serviço, onde tem-se vários dispositivos conectados na mesma, consumindo serviços de algum fornecedor como os gerenciadores de base de dados e as aplicações web. Dessa forma, é desnecessário se preocupar com servidores locais, sendo possível aproveitar as ferramentas adicionais que a provedora de computação em nuvem oferece.

Figura 2.28 - Ilustração do conceito de computação em nuvem como serviço. Acima tem-se os serviços oferecidos pela provedora e abaixo os dispositivos consumidores.



Fonte: Autor (2023).

2.7.1 PostGreSQL

O *PostGreSQL* [3] é um Sistema de Gerenciamento de Banco de Dados (SGBD) *open-source* onde sua finalidade é armazenar os dados de aplicações. Uma instância do *PostGreSQL* pode ser executada tanto em uma máquina local, neste caso o recurso local é transformado em um servidor de dados, quanto em uma máquina na nuvem, assim as aplicações precisam manter uma conexão com o servidor na rede Internet para ter acesso aos dados (OLIVEIRA, 2017).

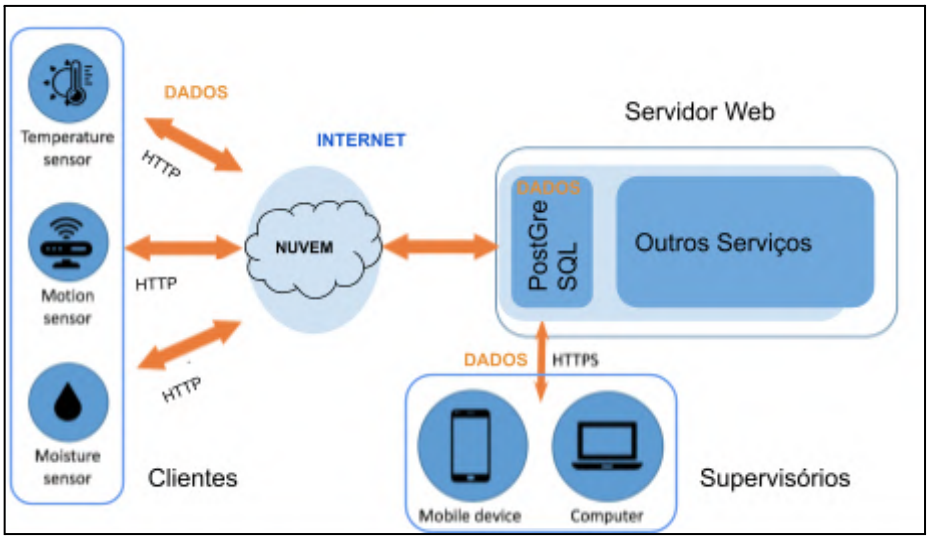
Por ser um dos principais SGBDs relacionais do mercado, ele atende demandas desde os mais simples negócios até em projetos IoT que lidam com grande volume de dados. Dessa forma, muitos servidores para hospedagem de aplicações na Internet tem compatibilidade com o *PostGreSQL*.

Em projetos IoT é comum aproveitar os recursos que os serviços *web* entregam para disponibilizar dados na nuvem. A Figura 2.29 exemplifica uma situação onde um sistema embarcado pode ser um cliente e realizar requisições HTTP para resgatar ou inserir dados de

[3] **PostgreSQL**: The world's most advanced open source database. Disponível em: https://www.postgresql.org/. Acessado em 10 jul. 2023.

sensoriamento no servidor web gerenciado pelo *PostGreSQL* e assim, posteriormente, um supervisório remoto pode ter acesso aos mesmos dados (OLIVEIRA, 2017).

Figura 2.29 - Exemplo de como os dados podem trafegar pela Internet, no qual ficam armazenados em uma instância do *PostGreSQL* na nuvem.



Fonte: Autor (2023).

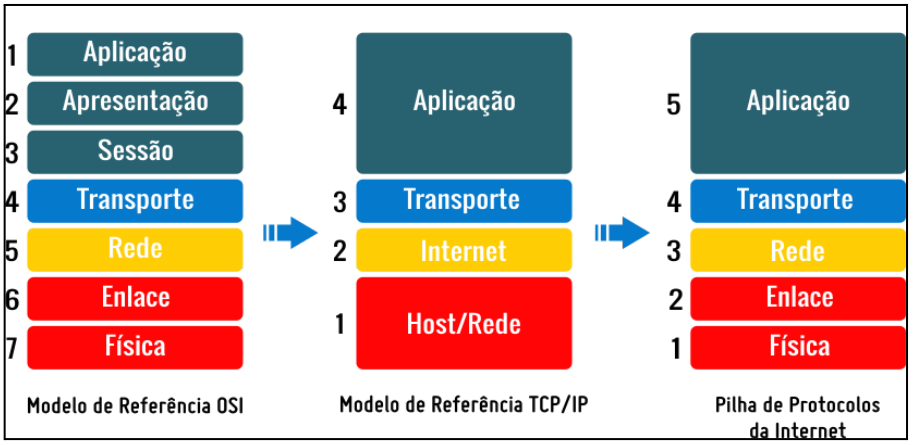
2.8 Redes de computadores

As redes de computadores podem ser definidas como um conjunto de dispositivos de computação interconectados com o objetivo de permitir o compartilhamento de informações, onde deve-se seguir determinados protocolos de comunicação (FOROUZAN; MOSHARRAF, 2013).

Segundo FOROUZAN e MOSHARRAF (2013), as redes e os protocolos de comunicação se tornaram indispensáveis com o advento da Internet e suas tecnologias a partir da década de 1970. Para assegurar que os sistemas de comunicação ou as redes de computadores na Internet sejam completos e eficientes criou-se um conjunto de protocolos divididos em camadas para evitar duplicação de esforço. Cada camada é implementada de forma isolada, porém em conjunto elas cobrem todos os aspectos da comunicação em rede, onde cada abstração consegue se interligar com as partições ou camadas vizinhas. Dessa forma, os projetistas e implementadores de protocolos conseguem gerenciar a complexidade das redes focando somente em um determinado protocolo por vez, evitando com que as outras camadas ou protocolos sofram alguma interferência (COMER, 2016).

A Figura 2.30 clarifica o modelo em camadas no qual regra a implementação dos protocolos da *Internet*. Segundo Oliveira (2017), a *International Standards Organization* (ISO) propôs o modelo *Open Systems Interconnection* (OSI) com sete camadas, porém, na prática, o modelo *Transmission Control Protocol/ Internet Protocol* (TCP/IP) com cinco camadas criado pela *The United States Department of Defense* (USDOD) ficou mais popular, sendo elas: aplicação, transporte, rede, enlace e física. Por fim, a Tabela 2.3 demonstra alguns exemplos de protocolos utilizados em cada camada do modelo TCP/IP.

Figura 2.30 - Modelos em camadas utilizados para a Internet. O modelo de abstração em 5 camadas se tornou mais popular.



Fonte: Santos (2016).

Tabela 2.3 - Exemplos de protocolos implementados em cada camada do modelo TCP/IP.

Camada	Protocolos
Aplicação	HTTP, RDP, SMTP, DNS, FTP, TELNET
Transporte	TCP, UDP, TLS, SSL, SSH
Rede	IPv4, IPv6, ICMP, ARP
Enlace	Ethernet, Wi-fi, Token Ring
Física	RS232, Bluetooth, USB

Fonte: Adaptado de Comer (2016).

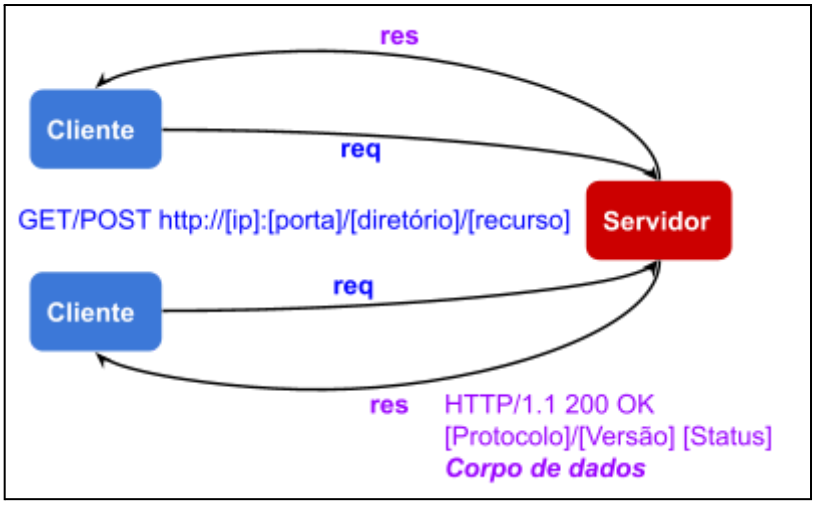
2.8.1 Protocolo HTTP

O *Hypertext Transfer Protocol* (HTTP) é um protocolo que atua na camada de aplicação no qual permite a troca de dados pela Internet a partir da arquitetura

cliente/servidor. O HTTP é predominantemente utilizado pelos navegadores web para buscar documentos de páginas HTML em servidores e redirecionar ao usuário cliente por meio de endereços que são traduzidos pelo navegador em requisições. O fluxo de comunicação do protocolo foi pensado para ser agradável e legível ao usuário, dessa forma o desenvolvimento de testes e aplicações se torna mais simples (BELSHE et al., 2015).

Para que isso seja possível, define-se dois tipos de mensagens dentro do protocolo HTTP: a requisição e a resposta. Baseando-se na Figura 2.31, uma requisição (*req*) feita pelo cliente é formada por um verbo, como GET ou POST, uma *Uniform Resource Locators* (URL) com o domínio que aponta para o servidor e a porta TCP. Para o verbo POST é necessário um corpo de dados para publicar no domínio especificado. A resposta (*res*) enviada pelo servidor chega com a versão do protocolo HTTP, um código de status e um corpo de dados com o recurso requisitado (BELSHE et al., 2015).

Figura 2.31 - Ilustração do fluxo e formato das requisições e respostas na arquitetura cliente/servidor.



Fonte: Autor (2023).

Assim, muitas *Application Programming Interfaces* (APIs) são construídas baseando-se nas mensagens do protocolo HTTP para buscar dados em um servidor na nuvem. Dentro do contexto de IoT, o HTTP é um serviço web que está se tornando tendência no cenário, justamente porque ele é um recurso implementado nas APIs que integram diferentes dispositivos e sistemas, criados em diferentes linguagens e plataformas. As APIs mais populares que seguem este modelo são denominadas como *Representational State Transfer*

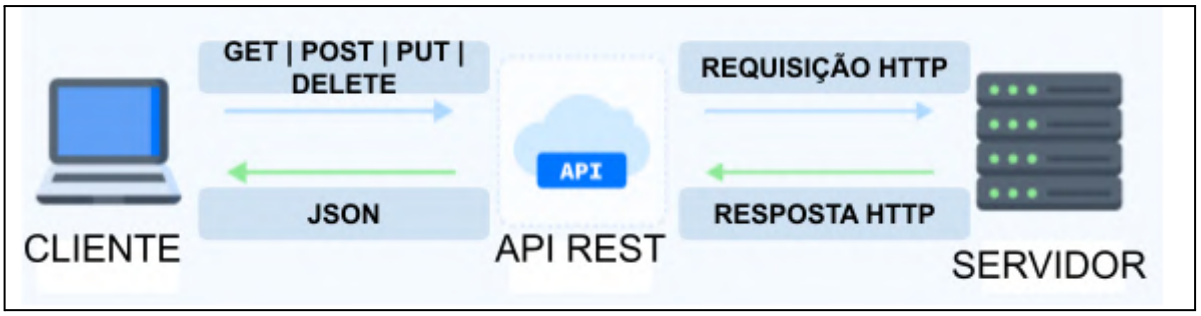
(REST), no qual tem seu diferencial por utilizar dos conceitos do protocolo HTTP para transmitir dados remotamente (OLIVEIRA, 2017).

2.8.2 API REST

Uma API é conceituada como um conjunto de protocolos e ferramentas definido para desenvolver um contrato que permita aplicações, plataformas, sistemas e programas se comunicarem, sem a necessidade de conhecer como as próprias aplicações foram implementadas. Ou seja, a API é um mediador na comunicação entre uma aplicação cliente e outra servidora, de forma que não é necessário ter detalhes de quem quer consumir a resposta na comunicação (FERREIRA, 2021).

Dessa forma, conforme a Figura 2.32 ilustra, uma API REST ou API RESTful é a interface que permite duas aplicações independentes interagirem por meio dos serviços web. A API REST entrega ao solicitante uma representação do estado do recurso gerenciado pelo protocolo HTTP e em diferentes formatos, sendo o mais conhecido o *JavaScript Object Notation* (JSON), pois é facilmente lido por humanos e máquinas (SONI; RANGA, 2019).

Figura 2.32 - Modelo que ilustra a utilização de uma API REST genérica para acessar serviços Web.



Fonte: Autor (2023).

Sendo assim, o modelo REST é a arquitetura padrão na maioria das aplicações web. Além disso, as APIs REST têm todo o potencial para beneficiar os projetos IoT, de tal forma que a facilidade de implementá-las tornou o desenvolvimento para IoT escalável, rápido e independente de qual linguagem cada dispositivo na ponta foi programado. Isso significa que é possível transmitir dados entre dois sistemas embarcados por meio de uma arquitetura formada por clientes e servidores, com a sintaxe das requisições baseada no protocolo HTTP,

no qual é mais familiar para desenvolvedores, trazendo mais praticidade na integração entre diferentes tipos de soluções de tecnologia (RODRIGUES et al., 2020).

2.9 Internet das coisas

A Internet das Coisas é um paradigma que ilustra uma rede de dispositivos inteligentes conectados pela Internet. A Figura 2.29 demonstra fielmente esta ideia, onde os dispositivos inteligentes, ou clientes da comunicação (à esquerda na figura), podem ser objetos incorporados a sensores, softwares ou algum tipo de tecnologia, variando dentre simples componentes, até dispositivos domésticos e equipamentos industriais mais complexos (IDEALI, 2023).

A IoT é uma ramificação da Indústria 4.0 e está se tornando uma das mais importantes áreas de tecnologia do futuro, principalmente porque a demanda por informações é vista pelo mercado como algo inquestionável e essencial para estratégias de negócios, sendo que o valor da IoT para as empresas tem sentido quando dispositivos interconectados são capazes de prover dados para ajudar a gerenciar os negócios e aumentar a eficiência das operações (EFFEBI, 2019).

Além disso, a partir de 2013, a IoT começou a se envolver em sistemas que utilizam múltiplas tecnologias, não sendo mais um requisito ter conexão com a Internet. Algumas redes sem fio como *WI-FI*, *ZigBee*, MQTT, GSM, GPS entregam muitas vantagens em soluções IoT voltados para automação residencial, agrícola, industrial e para equipamentos móveis. Para tanto, esta realidade foi alcançada quando surgiram os sistemas embarcados com seus módulos integrados, tornando o custo das soluções compatível com os dispositivos a serem conectados (OLIVEIRA, 2017).

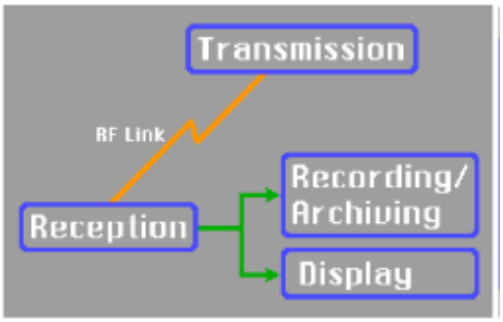
2.9.1 Telemetria

Segundo a definição de Mattos (2004), telemetria vem da junção de duas palavras *Tele* e *meter* que significam, respectivamente, longe e medir. O conceito surgiu em cenários onde existia a necessidade de mensurar variáveis em situações de difícil acesso, sendo possível por meio dos módulos de telemetria. Estes módulos representam os equipamentos ou subsistemas que têm a responsabilidade de medir os parâmetros físicos e condicionar os sinais para uma central receptora.

Portanto, Mattos (2004) comenta que um sistema de telemetria completo precisa ter quatro subsistemas essenciais, conforme a Figura 3.33, sendo eles:

- transmissão, composto basicamente por no mínimo um sensor para medição e uma antena transmissora. Nesta etapa implementa-se vários protocolos de empacotamento de dados, permitindo que sejam enviados no formato de *stream* de medidas via radiofrequência;
- recepção, ou estação de terra, inclui uma antena de rastreamento que recebe os sinais do subsistema de transmissão. Além disso, é comum que execute tarefas de condicionamento dentro de estações remotas, onde posteriormente encaminha os dados de telemetria para diversas centrais por meio de outras redes de comunicação;
- gravação, processo em que os dados são armazenados tanto durante a recepção quanto após todos os processamentos associados. Isso permite que o sinal seja preservado assim que chega no subsistema de recepção ou arquivado para ser reproduzido em subsistemas de visualização;
- visualização, ou mostrador, é utilizado para apresentar os dados do subsistema de gravação ao usuário para análises e tomadas de decisões. O mostrador pode funcionar tanto em tempo real de operação quanto em certo tempo após uma coleta.

Figura 2.33 - Subsistemas de um sistema de telemetria completo.



Fonte: Mattos (2004).

Em geral, a ideia de telemetria ganhou espaço juntamente às atividades aeroespaciais, onde o processo de medição era inacessível ao homem. Com isso, o conceito de remoticidade foi se expandindo, buscando compreender parâmetros de diferentes tipos de ambientes. Outro exemplo são os projetos baseados em IoT, nos quais têm a telemetria como essência, onde os dados são gerados remotamente por meio de um sistema embarcado e os subsistemas de recepção e transmissão se comunicam via rede móvel ou cabeada, até o momento que a

comunicação chega na Internet e popula os bancos de dados, podendo ser posteriormente acessados por serviços web para visualização.

2.9.2 Plataformas de visualização de dados

As plataformas de visualização são softwares que utilizam tecnologias de comunicação para o monitoramento de dados, controle e gestão de informações. Assim, conforme pode ser observado na Figura 2.29, as plataformas, ou supervisórios (localizados abaixo na figura), operam como clientes que requisitam os dados de um servidor na nuvem, além de serem o intermediário entre o serviço que fornece o conteúdo e o usuário analista. As plataformas de visualização de dados têm sua vantagem na tentativa de representar visualmente padrões e tendências que são críticos para uma boa gestão das informações (OLIVEIRA et al., 2014).

No mundo empresarial, há diversos softwares e bibliotecas disponíveis para o monitoramento de dados, como o *Microsoft Power BI*, *Google Charts*, *Tableau*, entre outros. Todas elas compartilham de algumas características: elas utilizam de APIs para a importação dos dados, apresentam gráficos interativos que tornam a visualização mais atraente e, além disso, sua finalidade é oferecer recursos de *business intelligence* para os usuários criarem relatórios e *dashboards*.

Com a popularização da IoT, é possível encontrar inúmeras aplicações web com o objetivo de entregar funcionalidades para a visualização de dados gerados pelos sistemas embarcados. Por se tratarem de ferramentas *open-sources*, há a possibilidade de aproveitar seus recursos para criar novas aplicações com as interfaces do usuário, sendo comum implementar por meio das tecnologias voltadas à *web* como o *React.JS*, JS, HTML e CSS (OLIVEIRA et al., 2014).

2.10 ICONIX

A metodologia adotada para o desenvolvimento da lógica de programação é baseada no processo *Iconic Notation for Interfaces and Exchanges* (ICONIX), no qual é uma ferramenta de modelagem de software ágil e elaborada a partir da *Unified Modeling Language* (UML). É utilizada para especificar capacidades, documentar artefatos e guiar a análise de sistemas de software orientados a objetos antes mesmo de iniciar com o código (ROSENBERG; STEPHENS, 2005). Uma variação da metodologia ICONIX é utilizada neste

trabalho, baseando-se nas seguintes etapas que devem ser realizadas sequencialmente, conforme citam ROSENBERG e STEPHENS (2005):

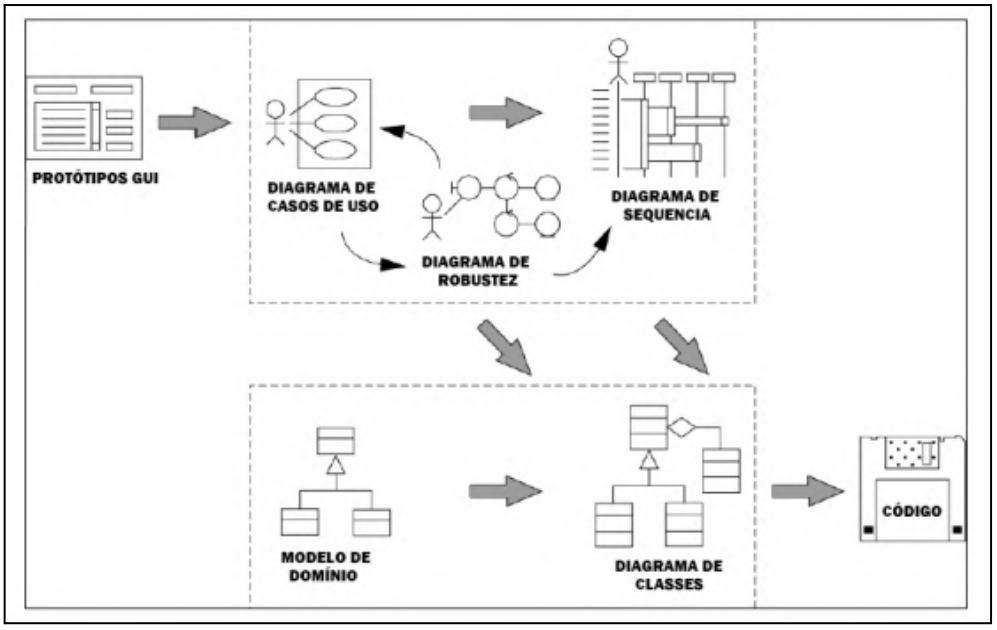
- Análise de Requisitos: nesta etapa são avaliados as instruções que o software deve executar em alto nível, assim como demonstrar os cenários onde um usuário pode interagir com o sistema e os relacionamentos dos fluxos de dados. Assim, o diagrama de casos de uso é o artefato utilizado para a análise de requisitos;
- Análise de Projeto Preliminar: basicamente é o refinamento dos requisitos identificados na etapa anterior, sendo geralmente caracterizado por detalhar as funcionalidades associadas a cada requisito do software, o que permite identificar lacunas e mensurar esforço. O artefato proposto para uso é o diagrama de sequência;
- Projeto de Software: nesta etapa é identificado como realmente será construído o sistema de software na prática, envolvendo uma análise de baixo nível e seu design. O diagrama de classes é adotado como artefato pois ajuda a identificar as relações entre os objetos;
- Implementação do código: por fim, o software é implementado a partir de alguma linguagem de programação. As análises anteriores facilitam para o desenvolvedor não se afastar dos requisitos do sistema durante a fase de implementação.

A Figura 3.34 ilustra os artefatos que podem ser utilizados durante a modelagem do software pelo ICONIX. ROSENBERG e STEPHEN (2005) ressaltam a importância da modelagem visual como ferramenta poderosa de comunicação e documentação. Logo, os diagramas UML são muito utilizados para representar os diferentes aspectos do sistema e suas etapas.

Dessa forma, o processo ICONIX se fundamenta fortemente em metodologias ágeis, visto que sua abordagem é feita de forma incremental e iterativa. Ou seja, o desenvolvimento de software fica dividido em pequenas iterações, com entregas incrementais, de fácil entendimento e em curtos prazos. Assim, a equipe de desenvolvimento consegue trabalhar com feedbacks rápidos e realizar ajustes conforme necessário.

Como resultado, os softwares desenvolvidos atendem com maior precisão aos requisitos dos clientes, dentro de um nível de qualidade esperado, além de serem facilmente entendidos, mantidos e testados por diferentes desenvolvedores. Isto acontece pois o design e o código não divergem com o tempo, tornando-se mais sincronizados.

Figura 2.34 - Visão dos artefatos no processo ICONIX.

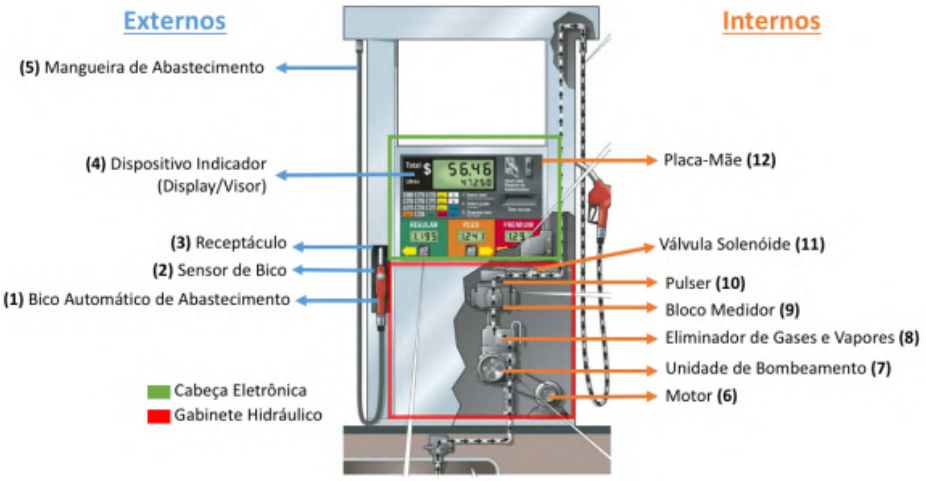


Fonte: Rosenberg (2005).

2.11 Bombas medidoras de combustível

A Figura 2.35 apresenta os principais elementos que compõem uma bomba medidora de combustível onde, no geral, ela consiste em uma caixa, mecanismo de bombeamento, flange de montagem, cabeçote e uma válvula de descompressão, no qual são responsáveis por bombear combustível à pressões uniformes e podem sofrer diversas variações conforme a finalidade da aplicação. Quanto ao modo de operação as unidades podem ser classificadas com mecânicas ou eletrônicas, no qual a diferença entre elas é o indicador e o sistema de medição que são controlados por circuitos eletrônicos, e quanto ao destino são caracterizadas em industrial ou comercial, onde o abastecimento de combustível é, respectivamente, de uso interno do detentor da instalação (sem revenda) ou de uso comercial, indicando o preço por litro (WAYNE, 2023)

Figura 2.35 - Ilustração de uma bomba medidora de combustível e seus elementos internos e externos.

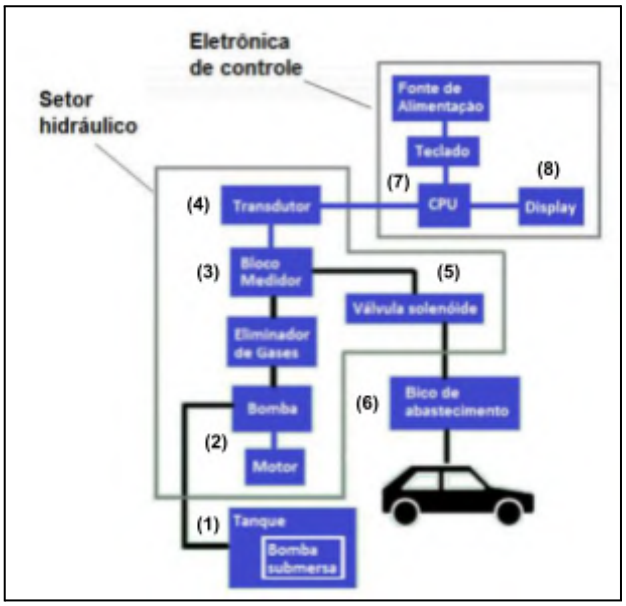


Fonte: Beteto (2019).

Além disso, a estrutura das bombas de combustível acompanha instrumentos de medição de vazão que realizam a interface com o usuário final, indicando assim o quanto foi o investimento para consumir o combustível. Os medidores são indispensáveis na projeção do equipamento devido a sua finalidade de garantir segurança durante o abastecimento e evitar fraudes (ABREU, 2017).

Sendo assim, a Figura 2.36 simplifica o processo realizado entre a medição da vazão até a indicação da quantidade abastecida no *display* da bomba, no qual se inicia com o combustível no tanque (1) que é bombeado (2) e atravessa um bloco medidor volumétrico (3), promovendo a rotação de um eixo transdutor (4). Em seguida, a válvula solenóide (5) permite a passagem do líquido para a mangueira de abastecimento até o bico injetor (6). O transdutor opera como um *pulser* que converte o movimento mecânico em pulsos elétricos, no qual são sinais que circulam pela placa eletrônica e são processados na CPU (7). Logo, uma sequência de pulsos indica o volume até o momento fornecido e o valor proporcional a pagar, cabendo a responsabilidade à CPU em converter esta sequência em informação de quantidade abastecida e indicar no *display* (8) ao usuário (BETETO, 2019).

Figura 2.36 - Etapas de funcionamento simplificadas de uma bomba medidora de combustível.



Fonte: Adaptado de Abreu (2017).

3 MATERIAIS E MÉTODOS

No decorrer deste capítulo será caracterizado, inicialmente, em qual modelo de pesquisa o trabalho se enquadra em respeito à natureza, à abordagem, aos objetivos e aos procedimentos. Em seguida, serão apresentados os materiais e a metodologia adotada para a montagem do projeto tanto em nível de hardware quanto em software.

3.1 Tipo de pesquisa

A investigação proposta na introdução deste trabalho foi conduzida segundo métodos científicos, sendo que, conforme o material apresentado por Nascimento (2016), esta pesquisa pode ser classificada quanto à sua natureza como aplicada, pelo fato que o objetivo geral é desenvolver uma solução baseada em sistemas embarcados para a automação dos postos internos. Assim, a pesquisa envolverá o projeto, desenvolvimento e implementação do sistema, visando aprimorar um processo prático e atender as necessidades do setor de abastecimento de combustíveis nos postos internos.

Quanto à abordagem, a pesquisa se enquadra dentro da categoria quantitativa, visto que serão utilizados métodos e técnicas durante a coleta de dados numéricos e no desenvolvimento tanto do hardware quanto do software, dentro de uma visão sistemática. Ou seja, o foco da abordagem quantitativa será para validar o funcionamento do sistema mensurando o desempenho e eficiência das suas funcionalidades na prática, nos quais serão utilizados para avaliar o impacto da automação.

Além disso, quanto aos procedimentos a pesquisa é caracterizada como experimental, pois serão manipulados algumas variáveis independentes no software, como os comandos dos algoritmos no programa principal, e no hardware, como a quantidade de pulsos coletados pelo ESP32 da bomba de combustível. Partindo disso, serão observadas suas influências nas variáveis dependentes como o fluxo de combustível, precisão do abastecimento, comportamento dos módulos e o tráfego de dados na aplicação web.

Por fim, no que diz respeito aos objetivos, a pesquisa é considerada descritiva, pois visa analisar as características e compreender o funcionamento do sistema de automação durante os testes após seu desenvolvimento completo. Dessa forma, os resultados são baseados na coleta detalhada de informações sobre o desempenho e as vantagens do uso do sistema, auxiliando principalmente no entendimento do seu comportamento.

3.2 Delimitação do tema

O presente trabalho tem como objetivo o desenvolvimento de um sistema automatizado para bombas de combustível em postos internos, visando oferecer uma solução de baixo custo para o gerenciamento de abastecimentos. Para alcançar os objetivos estabelecidos, este projeto concentra-se no estudo e na implementação de um protótipo baseado na placa DevKitC-ESP32, juntamente com uma aplicação web para monitorar os dados.

Sendo assim, o protótipo será desenvolvido em um laboratório na cidade de Lavras, Minas Gerais, onde será utilizado o gerador de pulsos modelo FBCGQ-3 para simular os abastecimentos durante os testes. Além disso, a linguagem C++ será empregada para implementar a lógica do sistema na placa ESP32, e a biblioteca *React.JS* para desenvolver a aplicação web. Os dados de abastecimento serão armazenados em uma instância do *PostGreSQL* e acessados em um servidor hospedado na nuvem por meio de APIs REST.

3.3 Topologia do projeto

A Figura 3.1 apresenta a topologia utilizada no projeto. Essencialmente, essa topologia é baseada na análise de quatro elementos ou camadas, que são a bomba de combustível, o sistema com a placa ESP32, o servidor web e a aplicação web.

De acordo com a Figura 3.1, a primeira camada é composta pela bomba de combustível. A partir dela, é possível obter dados sobre a quantidade de combustível abastecido de duas maneiras: por meio de pulsos eletromecânicos gerados pelos sensores presentes no equipamento, ou por meio dos terminais da placa-mãe (BETETO, 2019). Assim, uma GPIO do ESP32 deve ser dimensionada para conseguir capturar estes pulsos elétricos.

Em seguida, a Figura 3.2 apresenta o sistema com a placa ESP32, no qual é a segunda camada do sistema de automação e onde devem ser conectados os módulos de comunicação, sensores e atuadores. Durante esta etapa, tanto o hardware quanto o software que será implementado no ESP32 deve ser responsável por controlar 3 processos, sendo eles:

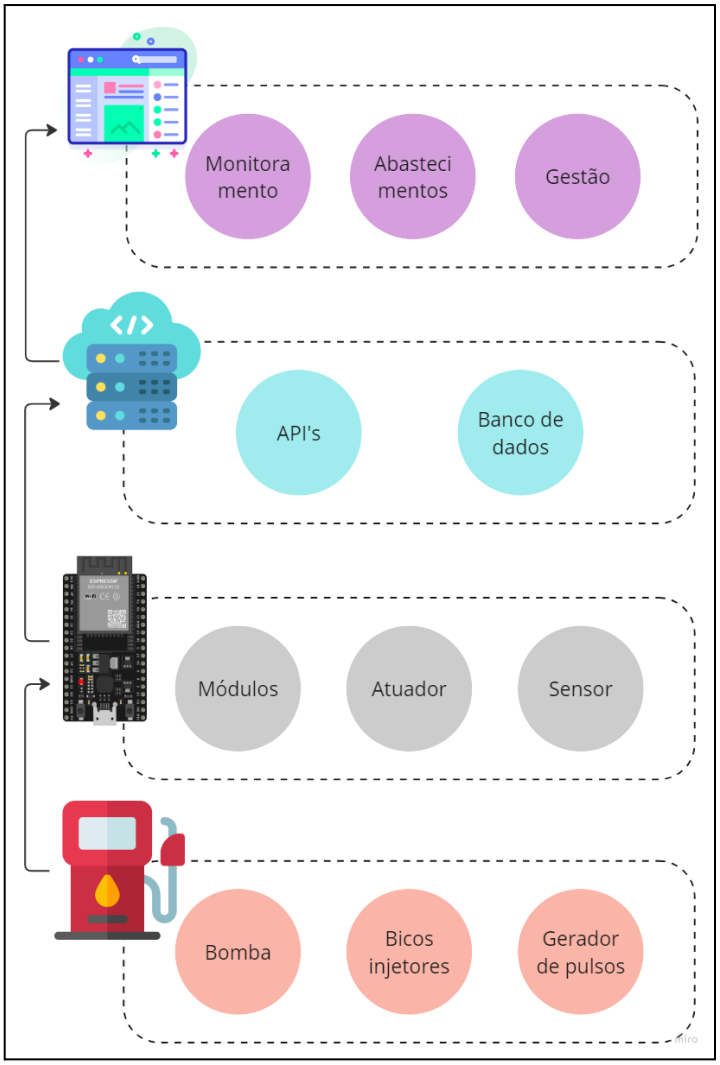
- Controle de acesso: primeiramente, o usuário deve se identificar no leitor RFID Dréxia com seu cartão de identificação antes de utilizar a bomba de combustível, na seguinte sequência: frentista, veículo e motorista. Após a identificação, o ESP32 realizará o controle de acesso, validando se os cartões identificados estão cadastrados no sistema;

- Controle do abastecimento: após a conclusão do primeiro controle, o ESP32 acionará o relé que está bloqueando o uso da bomba de combustível e permitir com que o usuário frentista utilize o bico injetor. Dessa forma, os pulsos elétricos gerados durante o uso do bico injetor serão coletados pelo ESP32 e transformados em quantidade de litros abastecidos.
- Controle dos dados: por fim, assim que o abastecimento é finalizado, todos os dados gerados na etapa anterior são armazenados internamente no sistema e, posteriormente, enviados ao servidor web que hospeda a instância do banco de dados remota. O módulo SIM800L ficará responsável por esta tarefa.

Por sua vez, a terceira camada representa o servidor web que manterá uma instância do *PostGreSQL* remotamente para armazenar os dados enviados pelo sistema com a placa ESP32, sendo possível escrever e ler no banco de dados a partir de APIs REST.

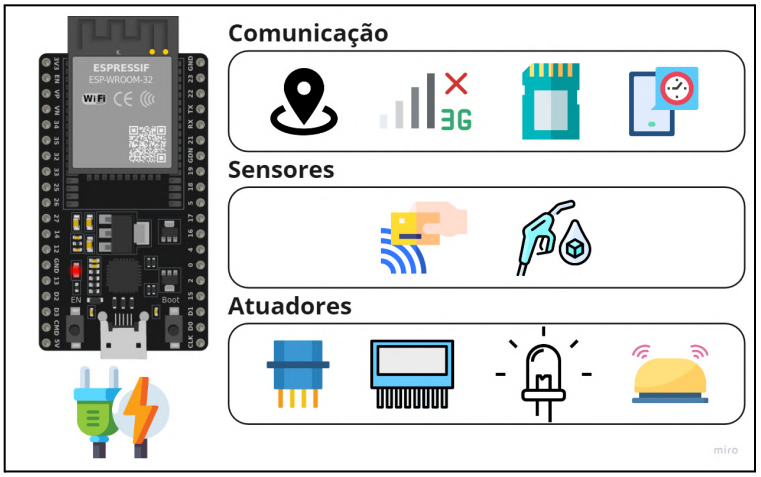
Em último lugar, a quarta camada representa a aplicação web, no qual ficará responsável por solicitar ao servidor web os dados dos abastecimentos reutilizando as mesmas APIs REST. O acesso à aplicação deve ser possível dentro dos navegadores web e disponível a todo momento para uso dos gestores, analistas ou qualquer usuário com as mesmas funções.

Figura 3.1 - Topologia adotada para o projeto.



Fonte: Autor (2023).

Figura 3.2 - Topologia do sistema com a placa ESP32.



Fonte: Autor (2023).

3.3.1 Bomba de combustível

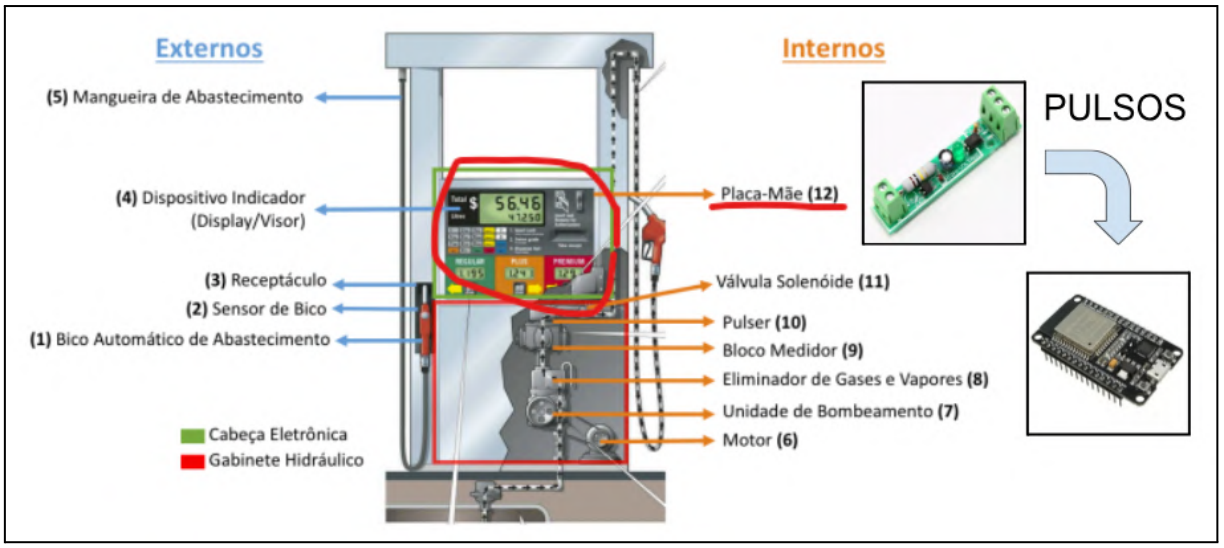
A Figura 3.3 apresenta o modelo de bomba de combustível que será utilizado para a pesquisa. Essencialmente, a bomba é um modelo simples da marca Wayne (Wayne Fueling Systems, Texas, Estados Unidos), sendo uma das fornecedoras mais presentes no mercado atualmente e com uma gama de produtos de automação que podem ser integrados de forma a facilitar o monitoramento de combustível (WAYNE, 2023). Dessa forma, seus modelos de bombas de combustível apresentam uma tendência maior quanto a flexibilidade e facilidade na instalação de dispositivos terceirizados.

Figura 3.3 - Modelo de bomba de combustível adotado na pesquisa.

Fonte: Autor (2023).

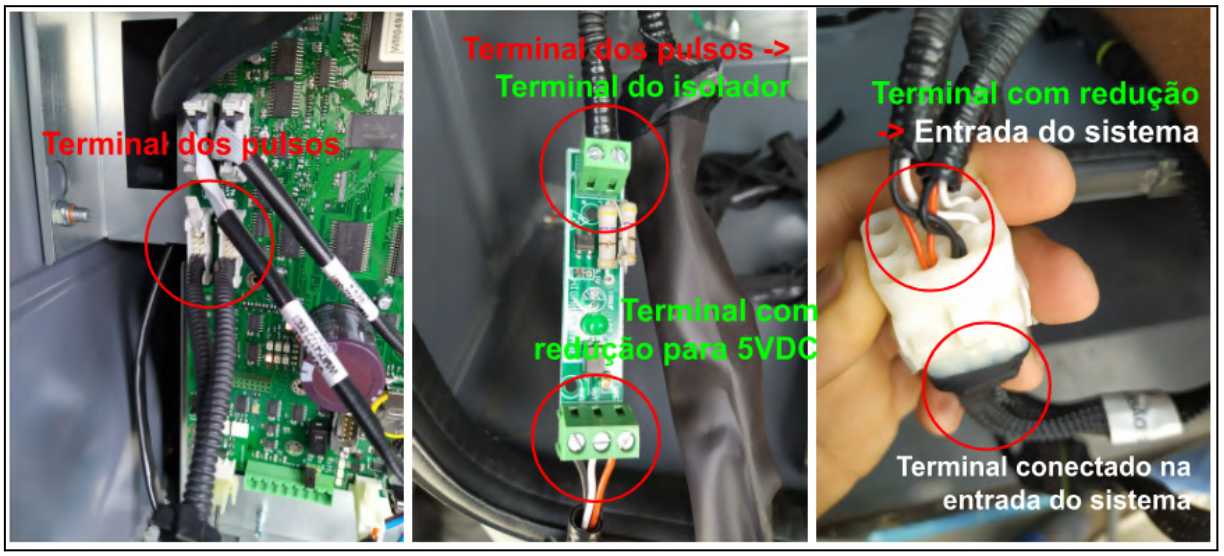
Além disso, este modelo de bomba é elétrico, logo é possível interceptar os pulsos elétricos que contabilizam a quantidade de combustível passando pelo bloco medidor com o apoio de um isolador óptico, conforme apresentado pela Figura 3.4. Em campo, a Figura 3.5 ilustra como será instalado o isolador óptico, no qual tem a função de transformar os pulsos em níveis lógicos suportados pelo ESP32 e enviá-los para o próprio sistema embarcado.

Figura 3.4 - Disposição do isolador óptico, utilizado para interceptar os pulsos elétricos.



Fonte: Adaptado de Beteto (2019).

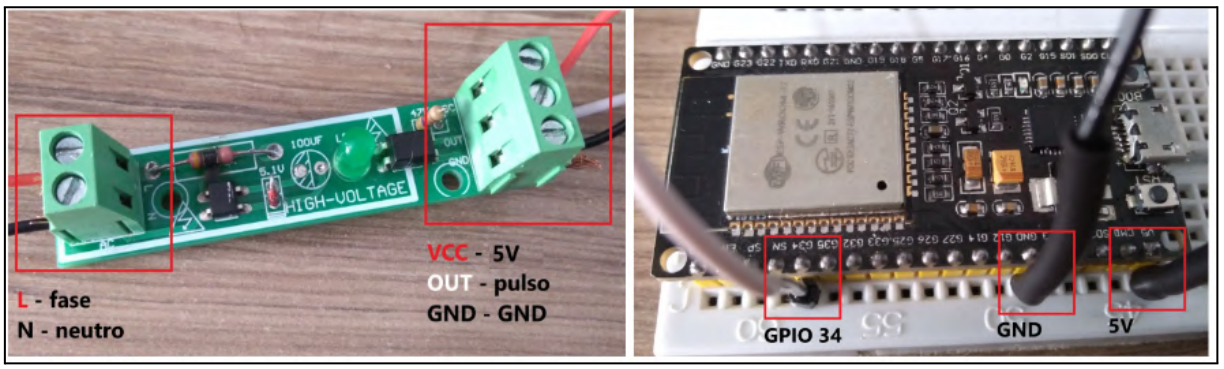
Figura 3.5 - Instalação do isolador óptico.



Fonte: Autor (2023).

Portanto, conforme indicado pela Figura 3.6, o sistema desenvolvido com o kit de desenvolvimento ESP32 deve associar uma GPIO que receba os pulsos interceptados, de forma a ter conhecimento da quantidade de combustível já abastecida, calculada em software, e atuar posteriormente nos bicos injetores para bloquear ou liberar o bombeamento de combustível.

Figura 3.6 - Conexão entre o isolador óptico e o ESP32 em sua GPIO 34, recebendo os pulsos da placa-mãe da bomba.

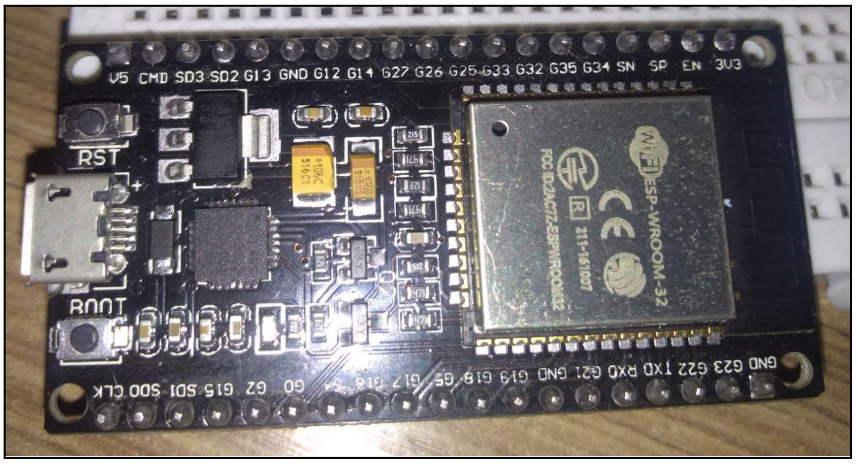


Fonte: Autor (2023).

3.3.2 Sistema com a placa ESP32

A metodologia de montagem do hardware se fundamenta principalmente na placa de desenvolvimento *ESP32-DevKitC* apresentado na Figura 3.7, no qual segundo informações da Espressif Systems (2023), este kit de desenvolvimento tem um conjunto de periféricos mais rico e otimizado para prototipagem.

Figura 3.7 - Modelo *ESP32-DevKitC* com o microcontrolador ESP32-WROOM-32.

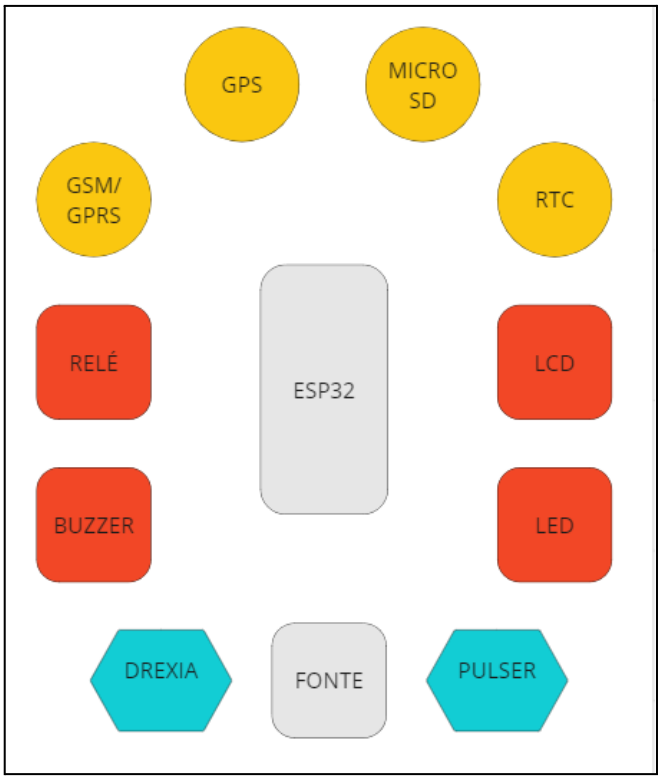


Fonte: Autor (2023).

A Figura 3.8 ilustra um esquema simples com os módulos de comunicação, em amarelo, os sensores, em azul, e os atuadores, em vermelho, que serão conectados ao *ESP32-DevKitC*. Além disso, um módulo regulador de tensão, indicado pelo bloco “Fonte”

em cinza, é inserido para ajustar e manter adequados os níveis de tensão de alimentação do sistema.

Figura 3.8 - Esquema simples dos módulos de comunicação (em amarelo), sensores (em azul), atuadores (em vermelho) e fonte de alimentação (em cinza).

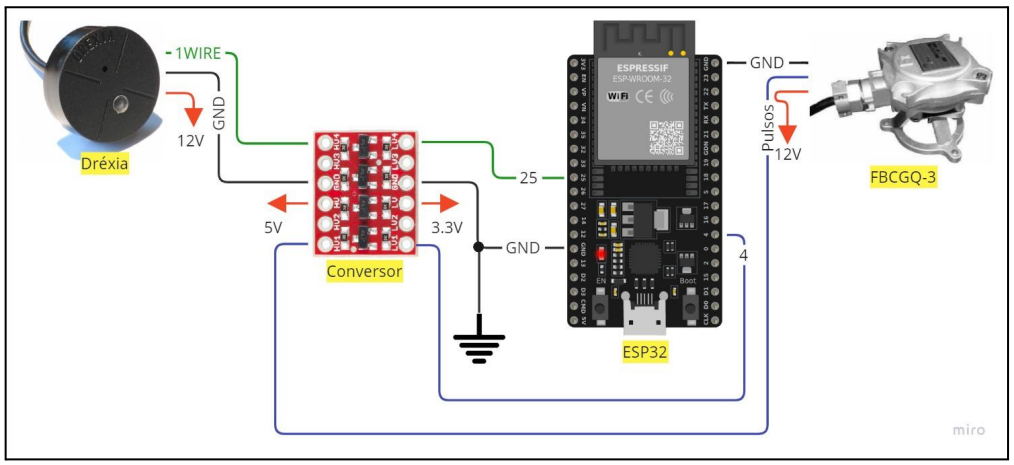


Fonte: Autor (2023).

3.3.2.1 Sensores

Seguindo o esquema apresentado na Figura 3.8, o leitor RFID modelo 1W-H3-05 será dimensionado para a identificação dos usuários. A Figura 3.9 demonstra a conexão proposta para o sensor, sendo que por padrão eles trabalham com nível lógico alto em 5V, o que não é adequado para as GPIOs da placa ESP32. Dessa forma, o conversor de nível lógico de 5V para 3,3V será dimensionado no projeto para que seja possível utilizar o protocolo 1-WIRE sem danificar o kit de desenvolvimento. Além disso, uma GPIO do ESP32 será reservada para conectar o terminal de nível lógico do gerador de pulsos eletromecânico FBCGQ-3.

Figura 3.9 - Esquema de conexão proposto para os sensores de identificação e para o gerador de pulsos da bomba de combustível.



Fonte: Autor (2023).

3.3.2.2 Atuadores

Os atuadores necessários para compor o sistema serão um relé DNI0129 para bloqueio, um emissor de sinais sonoros buzzer, três LEDs com as cores vermelha, azul e verde, e por fim um display LCD de 16 colunas por 2 linhas, resumidamente indicados pela Figura 3.8.

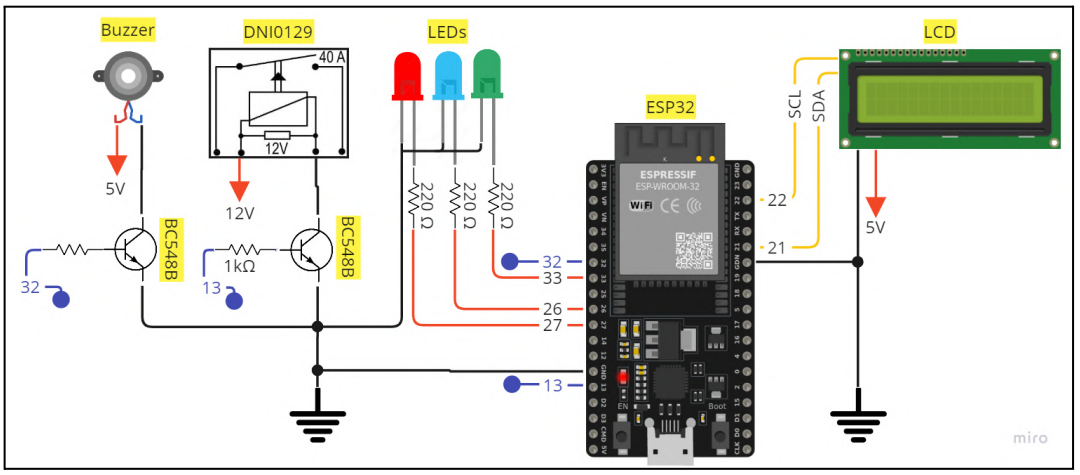
Dessa forma, a Figura 3.10 apresenta o esquema de conexão para os atuadores. Primeiramente, o relé DNI0129 tem a finalidade de bloquear o bico injetor da bomba de combustível até que o ESP32 envie um sinal de liberação conforme a lógica de controle de acesso. Portanto, é extremamente necessário dimensionar um circuito de controle por meio de um transistor para isolar as portas lógicas do ESP32 de receberem os níveis de tensões que operam na bomba medidora de combustível.

Seguindo, conforme apresentado pelo datasheet do sensor RFID Dréxia, o dispositivo possui um buzzer integrado, logo o componente será aproveitado do sensor para a emissão dos sinais sonoros.

Por fim, os LEDs vermelho, azul e verde indicarão, respectivamente, que algum erro na comunicação com qualquer módulo ocorreu, o módulo SIM800L está conectado na rede e o sistema está em operação. O display LCD complementa as informações dos LEDs indicando os erros do sistema em formato de registros ou o diagnóstico de determinada etapa do algoritmo, como, por exemplo, na identificação de algum usuário pelo sensor RFID

Dréxia, o display LCD deve emitir uma resposta visual do identificador hexadecimal do usuário.

Figura 3.10 - Esquema de conexão proposto para os atuadores do sistema.



Fonte: Autor (2023).

3.3.2.3 Módulos de comunicação

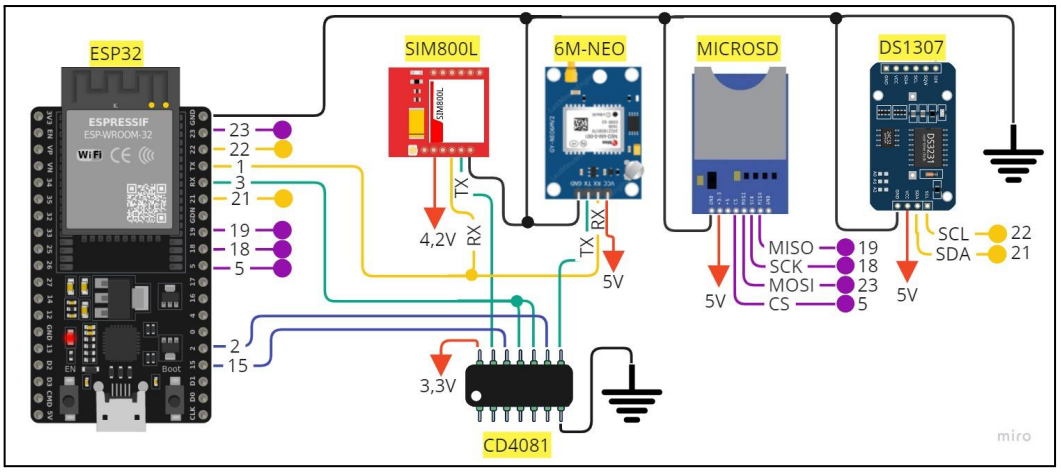
A Figura 3.11 ilustra o esquema desenvolvido para os dispositivos de comunicação, sendo eles um adaptador para cartão de memória microSD, um módulo SIM800L para trabalhar na rede GSM/GPRS, um módulo Ublox NEO-6M para capturar sinal GPS e um relógio de tempo real do modelo DS1307.

O módulo adaptador de cartão de memória microSD tem a finalidade de persistir os dados coletados pelo ESP32. Dessa forma é possível implementar um algoritmo para ler ou gravar no cartão de memória e posteriormente resgatar as informações em uma estrutura de dados.

Em segundo lugar, o módulo SIM800L é responsável por compartilhar os dados na nuvem via rede GPRS. Dessa forma, quaisquer informações geradas pelo sistema embarcado são enviadas ao servidor no qual está hospedado o banco de dados.

Por fim, os módulos RTC DS1307 e Ublox NEO-6M são dimensionados para complementar as informações dos abastecimentos gerados. Eles garantem, respectivamente, informações sobre os momentos e o local, em latitude e longitude, do abastecimento. Observa-se que é inserido um circuito integrado (CI) CD4081 no circuito para que seja possível chavear entre o módulo GPS Ublox NEO-6M e o módulo SIM800L dentro do barramento utilizando o UART, além também de evitar ruídos nos sinais.

Figura 3.11 - Esquema de conexão proposto para os módulos de comunicação do sistema.



Fonte: Autor (2023).

3.3.2.4 Alimentação

Segundo a documentação disponibilizada pela Espressif Systems (ESPRESSIF SYSTEMS, 2023), o microcontrolador do *ESP32-DevKitC* pode fornecer dois terminais de alimentação com níveis de tensão distintos de 3.3V e 5V, onde a corrente elétrica máxima de saída recomendada é de 500mA e não pode acumular, juntamente com todas as GPIOs, mais que 1200mA de saída, sabendo que cada GPIO pode fornecer no máximo 28mA. Portanto, considerando uma situação de maior consumo energético do sistema, os dispositivos serão dimensionados de forma que se respeite os limites de tensão e corrente elétrica apresentados pela fabricante tanto em conjunto quanto de forma individual.

Ou seja, conforme os dados indicados pela Tabela 3.1, nos quais estão presentes nas folhas de dados de cada dispositivo (apresentados nos tópicos Sensores, Atuadores e Módulos externos), o consumo de corrente elétrica total do sistema é de, aproximadamente, 2,4A, calculado na equação 3.1.

Tabela 3.1 - Consumo de cada dispositivo do projeto.

Módulos	Consumo [mA]
ESP32	500
SIM800L	2000
NEO-6M	45
DS1307	0.3
MicroSD	80
Dréxia	160
Pulser	20
Display LCD	16
LED vermelho	18
LED azul	18
LED verde	18
Buzzer	12
DNI0129	170
Total	**3057.3**

Fonte: Autor (2023).

$$Corrente\ total\ (mA) = 500 + 2000 + 45 + 0,3 + 80 + 160 + \\ + 20 + 16 + 18 + 18 + 18 + 12 + 170 \quad (3.1)$$

$$Corrente\ total\ (mA) = 3057.3$$

Sendo assim, a Tabela 3.2 apresenta resumidamente todos os módulos externos que serão utilizados no projeto, assim como quais GPIOs do ESP32 serão reservadas para eles, atentando principalmente nos valores de tensão e nível lógico de operação para cada dispositivo.

Tabela 3.2 - Agrupamento de todos os sensores, atuadores e módulos de comunicação.

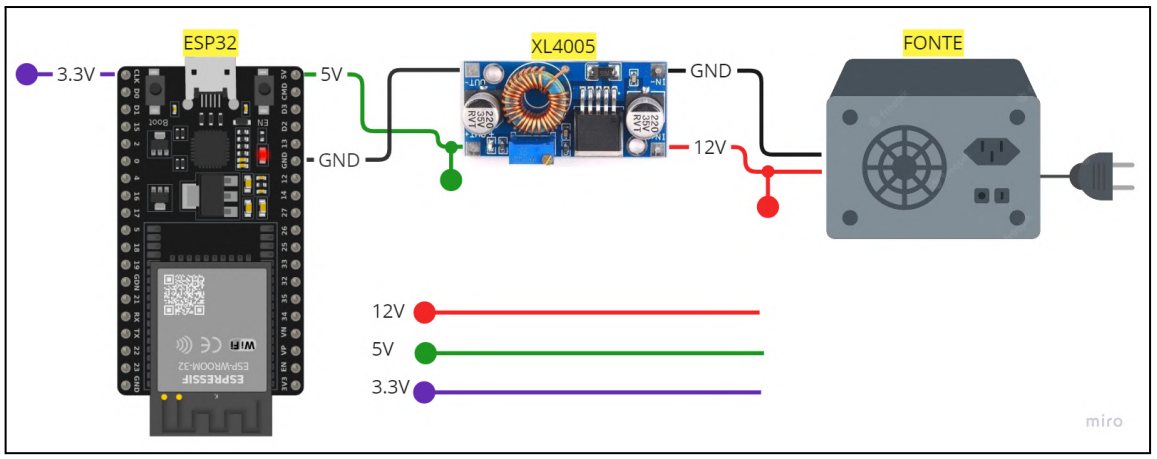
Módulos	Descrição	GPIO		Alimentação	Nível lógico
Embarcado	ESP32	-		5V/500mA	3.3V/28mA
Comunicação	Sim800L	3A 1A	UART	4.2V/2A	3.3V
	Ublox NEO-6M	3B 1B	UART	5V/45mA	3.3V
	DS1307	21A 22A	I2C	5V/0.3mA	3.3V
	Módulo cartão MicroSD	5 18 19 23	SPI	5V/80mA	3.3V
Sensor	Leitor RFID Dréxia	25	1-WIRE	12V/160mA	5V
	Pulser (Bomba)	4	INPUT	12V/20mA	5V
Atuador	Display LCD 16x2	21B 22B	I2C	5V/16mA	3.3V
	LED vermelho	27	OUTPUT	3.3V/18mA	3.3V
	LED azul	26	OUTPUT	3.3V/18mA	3.3V
	LED verde	33	OUTPUT	3.3V/18mA	3.3V
	Buzzer	13	OUTPUT	5V/12mA	5V
	DNI0129	14	OUTPUT	12V/170mA	-

Fonte: Autor (2023).

Logo, conforme demonstrado pela Figura 3.12, será dimensionado uma fonte de alimentação externa de 12V/5A associada em série a um regulador de tensão do modelo XL4005, pensando em suprir a demanda de todos os dispositivos do sistema. Os módulos que consomem 3.3V serão conectados diretamente no pino de saída de tensão de 3.3V do ESP32, aqueles que operam a 5V serão conectados na saída do regulador de tensão e os que operam a 12V serão conectados na saída da fonte de alimentação.

Complementando, vale ressaltar o módulo SIM800L que deve ser conectado na saída do regulador de tensão, pois ele pode chegar a consumir até 2A de corrente, ou seja, este módulo pode danificar o ESP32 se conectado diretamente na placa.

Figura 3.12 - Esquema de conexão proposto para os módulos de alimentação do sistema.



Fonte: Autor (2023).

3.3.3 Servidores web

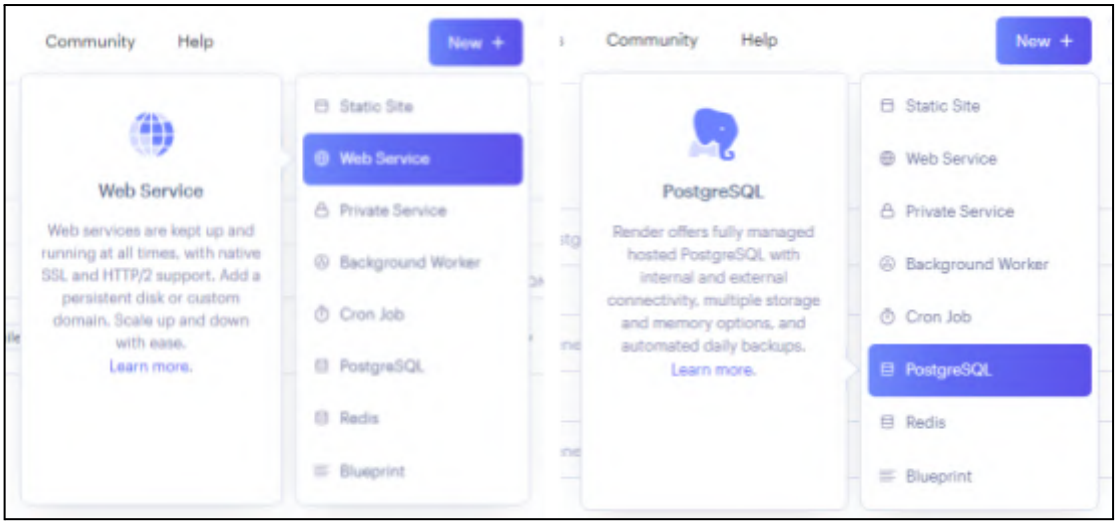
Para a hospedagem de todos os serviços web que serão implementados neste projeto serão utilizados dois servidores de hospedagem, sendo eles o *Render* [4] para o banco de dados e as rotas das APIs REST, e a *Vercel* [5] para a aplicação web.

A Figura 3.13 apresenta os serviços oferecidos no painel de controle do *Render*. Este provedor de hospedagem web é intuitivo, com funcionalidades semelhantes a um provedor de larga escala e plataformas como serviço. Sendo assim, o serviço com as rotas das APIs deve ser publicado em um repositório do GitHub e conectado no painel de controle do *Render* como um projeto *Web Service*, possibilitando com que qualquer atualização do código dentro do repositório seja também atualizado no serviço hospedado, e o banco de dados conectado em uma instância do *PostgreSQL* no mesmo painel de controle apresentado pela Figura 3.13.

[4] Render. **Dashboard**. Disponível em: https://dashboard.render.com/. Acesso em: 28 jun 2023.

[5] Vercel. **Dashboard**. Disponível em: https://vercel.com/dashboard. Acesso em: 28 jun 2023.

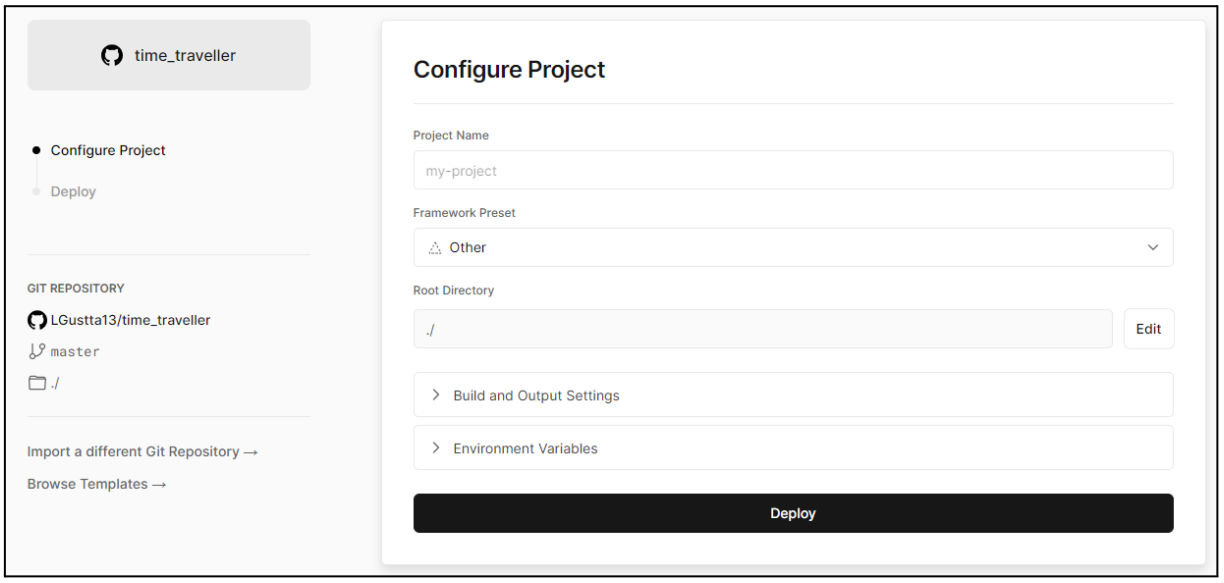
Figura 3.13 - Serviços do painel de controle do Render.



Fonte: Render (2023).

Em relação à aplicação web, a Figura 3.14 apresenta o painel de configuração dentro da *Vercel* para subir uma aplicação no serviço de hospedagem. Com isso, será necessário publicar o código da aplicação web em um repositório do GitHub e importar o mesmo no painel adotando as configurações padrões.

Figura 3.14 - Painel de configuração da *Vercel.*



Fonte: Vercel (2023).

3.3.4 Aplicação web

A aplicação web será desenvolvida de forma a entregar ao usuário a possibilidade de acessar os dados de abastecimento a partir de duas formas: em uma tela de Dashboard ou por meio de uma tela de Listagem. Dessa forma, com o intuito de simplificar a implementação do código das páginas, os layouts foram desenhados anteriormente dentro da plataforma de edição gráfica *Figma*, no qual pode ser acessada no próprio navegador do desenvolvedor.

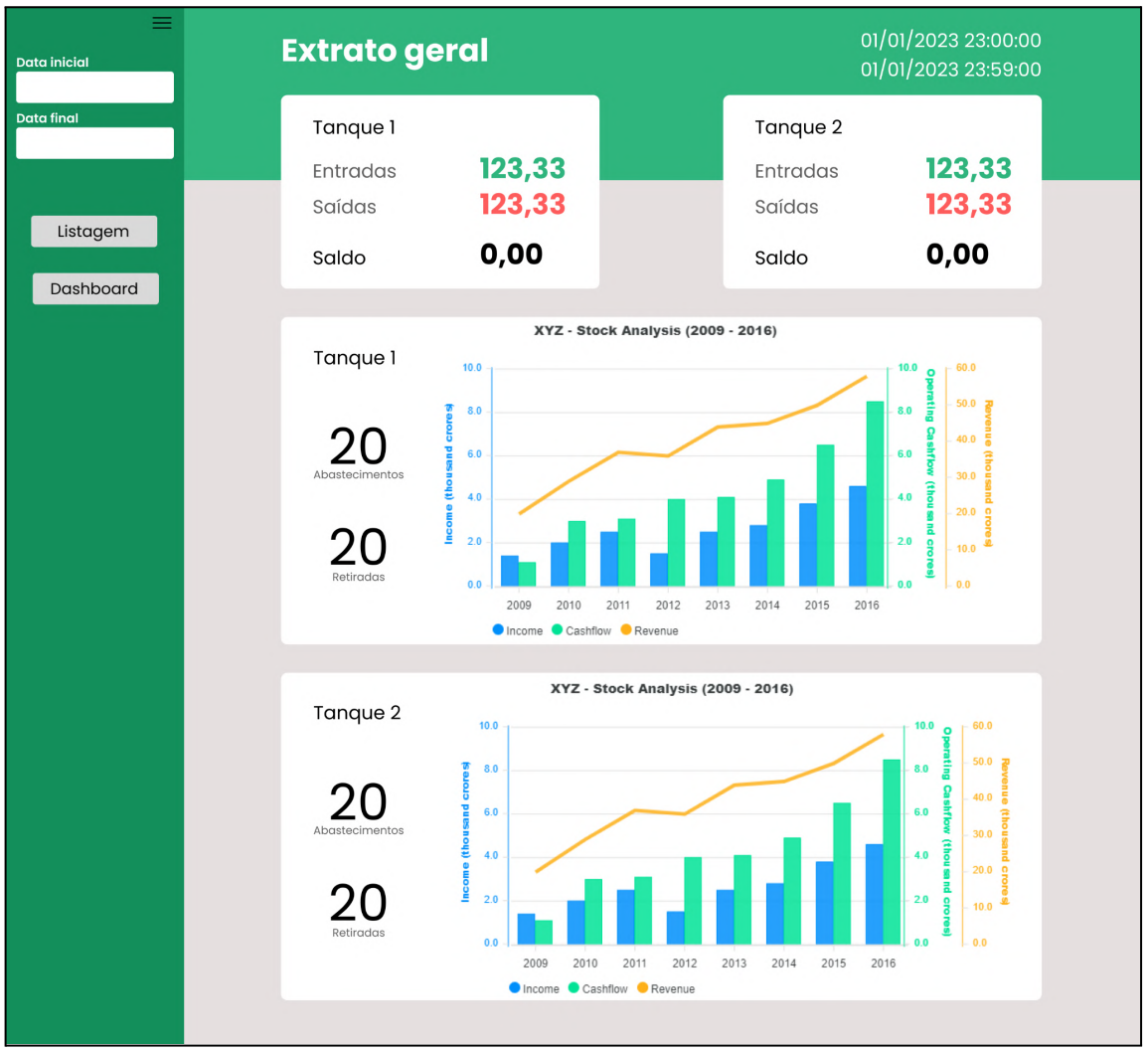
Assim, a tela principal, ou de Listagem, baseia-se no layout desenhado da Figura 3.15, no qual a lógica tem como função listar todos os abastecimentos dentro de um período de tempo definido pelo usuário.

Figura 3.15 - Layout desenhado para a página de Listagem.

Fonte: Autor (2023).

A tela de estatísticas, ou de Dashboard, pode ser visualizada no desenho da Figura 3.16, onde devem ser apresentados gráficos, dados informativos e indicadores visuais com o objetivo de ter uma visão sobre o extrato nas bombas e realizar a gestão de combustível. Dessa forma, para a implementação dos gráficos será utilizada a biblioteca *Apexcharts*, no qual entrega muitos recursos para organizar e visualizar dados em gráficos.

Figura 3.16 - Layout desenhado para a página de Dashboard com dois gráficos ilustrativos do *ApexCharts*.



Fonte: Autor (2023).

Por fim, para o desenvolvimento da lógica de programação, foi utilizado o VSCode como o ambiente de desenvolvimento, além da biblioteca React.Js e *ApexCharts*, respectivamente utilizadas na implementação do código e criação dos gráficos interativos. A execução do código pode ser realizada de duas formas:

- Durante a etapa de desenvolvimento, a lógica será executada em um servidor de desenvolvimento hospedado dentro da própria máquina do desenvolvedor, cabendo à própria API da biblioteca React.Js configurar este ambiente. Esta configuração acontece logo após a inicialização do projeto pelo comando *npx create-react-app* [6];

[6] FACEBOOK. Create React App. [*S. l.: s. n.*], 2023. Disponível em: https://create-react-app.dev/. Acesso em: 28 jun. 2023.

- Durante a etapa de produção, o código será hospedado manualmente dentro da plataforma de nuvem da *Vercel*, no qual é possível construir a aplicação e disponibilizar na Internet. Para isso, o projeto deve ser publicado em um repositório do GitHub e em seguida importá-lo dentro da plataforma.

3.4 Lógica de funcionamento

A metodologia adotada para o desenvolvimento do software, incluindo tanto o sistema com a placa ESP32 quanto a aplicação web, será baseada no processo ICONIX, no qual uma de suas variações é utilizada neste trabalho baseando-se nas seguinte etapas que devem ser realizadas sequencialmente:

- Análise de Requisitos, onde é utilizado o diagrama de casos de uso como artefato para descrever as interações entre o usuário e o sistema com a placa ESP32, além também do analista/gestor com a aplicação web;
- Análise de Projeto Preliminar, no qual é implementado o diagrama de sequência como solução para modelar o processo de cada caso de uso, compreendendo como cada componente funciona, sendo eles o software que controla o sistema, os controladores dos barramentos seriais e portas lógicas e os módulos externos (sensor, atuador e de comunicação);
- Projeto de Software, no qual envolve uma análise de nível mais baixo com diagramas de classes para mapear a estrutura de cada objeto por meio de atributos, métodos e relações. Dessa forma é possível fornecer informações para a etapa de implementação do código;
- Implementação do código, sendo o momento em que o software é desenvolvido a partir da linguagem de programação C++ na plataforma Arduino IDE, ou pelo React.Js no VSCode para a aplicação web.

3.4.1 Análise de requisitos

Os requisitos funcionais que o sistema deve atender se encontram na Tabela 3.3, onde as tarefas são definidas de forma básica para a construção posterior dos modelos de diagramas de caso de uso, sequência e classes. O levantamento de cada requisito partiu de uma pesquisa e análise subjetiva dos principais dados que a bomba de combustível pode gerar durante um abastecimento, considerando também as funcionalidades que uma plataforma de

monitoramento deve possuir para apresentar os mesmos dados. Sabendo disso, a metodologia ICONIX se justifica pelo fato de facilitar a manutenção do software, principalmente se existir a necessidade de inserir novas funcionalidades ao projeto.

Tabela 3.3 - Requisitos funcionais do sistema.

RF	Descrição
RF-01	O sistema deve ser capaz de configurar as bibliotecas necessárias para o funcionamento dos módulos de comunicação
RF-02	O sistema deve ser capaz de identificar um usuário
RF-03	O sistema deve ser capaz de iniciar um processo de abastecimento
RF-04	O sistema deve ser capaz de coletar dados de quantidade de combustível abastecido
RF-05	O sistema deve ser capaz de pegar dados de localização
RF-06	O sistema deve ser capaz de pegar dados de data e horário
RF-07	O sistema deve ser capaz de finalizar um abastecimento
RF-08	O sistema deve ser capaz de armazenar um abastecimento
RF-09	O sistema deve ser capaz de enviar um abastecimento para um servidor web
RF-10	O sistema deve ser capaz de resgatar os dados de abastecimento para a aplicação web
RF-11	O sistema deve permitir pesquisar abastecimentos em um determinado período na aplicação web
RF-12	O sistema deve ser capaz de apresentar dados de abastecimento em forma de gráficos e estatísticas

Fonte: Autor (2023).

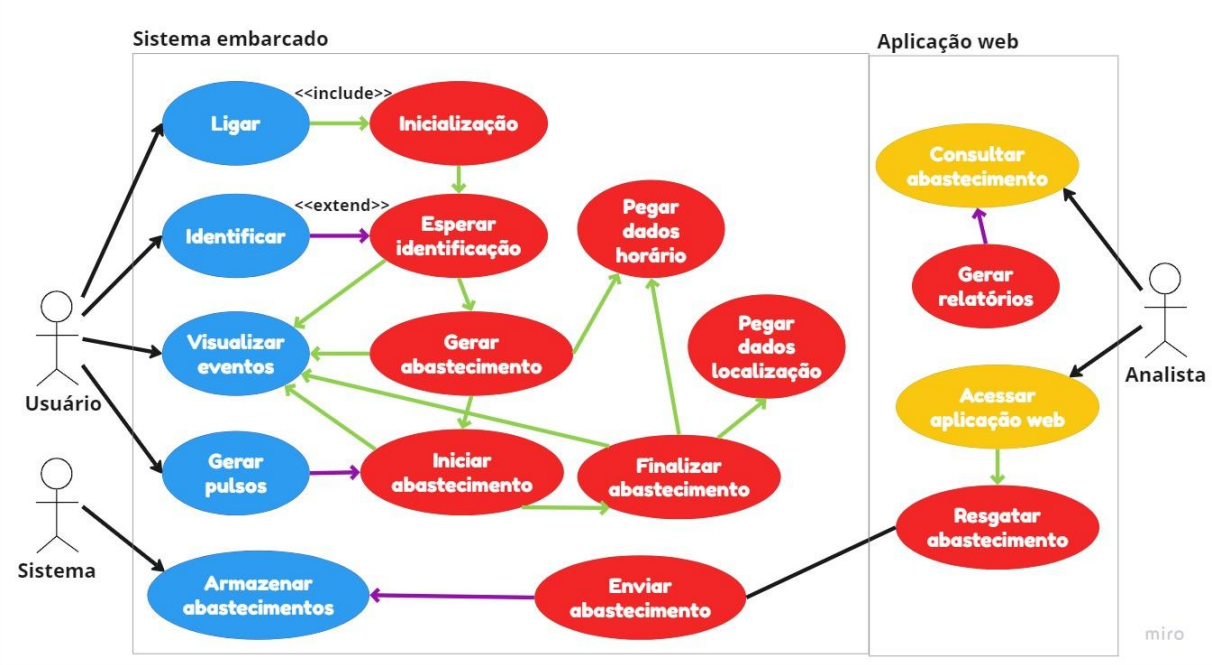
O diagrama geral de casos de uso do sistema pode ser observado a partir da Figura 3.17. Durante o levantamento dos requisitos, detectou-se três atores, sendo eles o usuário que se identifica no sistema na bomba indicado por Usuário, a própria automação do sistema indicada por Sistema e o analista que acessa a aplicação web representado por Analista. Os casos de uso indicados pelas elipses são os requisitos funcionais do sistema de forma textual, indicados pelas cores:

- Azul explica os papéis que os atores Usuário ou Sistema executam no bloco Sistema Embarcado;
- Vermelho detalha as ações e capacidades do próprio sistema e suas associações;
- Amarelo explica os papéis do ator Analista dentro do bloco Aplicação Web.

Todos os casos de uso apresentam relações entre si, sendo elas de inclusão ou extensão. Dessa forma, na Figura 3.17, as setas em verde demonstram uma relação de inclusão (*include*), onde se um caso de uso for executado o próximo relacionado deverá automaticamente ser executado. Em contrapartida, as setas em roxo indicam uma relação de extensão (*extend*), onde o caso de uso pode ou não pode ser executado se o próximo for executado. Por exemplo:

- Se o caso de uso Ligar for executado (papel do ator Usuário), o caso de uso Inicialização é automaticamente executado;
- Se o caso de uso Esperar Identificação for executado, nem sempre o caso de uso Identificar será executado, pois depende da ação do ator Usuário.

Figura 3.17 - Diagrama de casos de uso do sistema contemplando o sistema embarcado e a aplicação web.



Fonte: Autor (2023).

3.4.2 Projeto Preliminar do Software

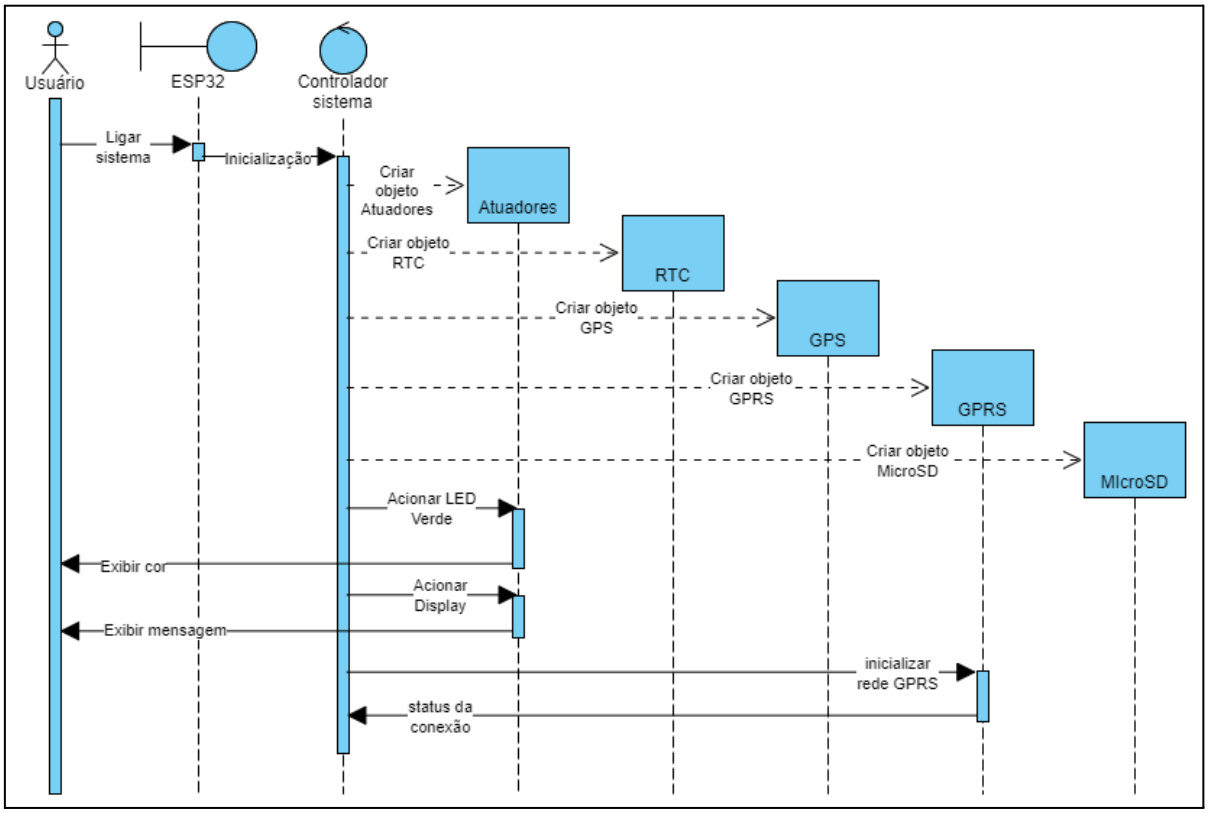
Nesta etapa, todos os casos de uso serão descritos segundo suas principais operações com o objetivo de identificar os objetos que devem fazer parte do sistema e suas relações de comunicação.

3.4.2.1 Caso de uso Inicialização

A Figura 3.18 apresenta o diagrama de sequência do caso de uso Inicialização, no qual é consequência da ação de Ligar do ator Usuário. Durante a inicialização do sistema, o algoritmo principal no ESP32, atribuído no diagrama como o Controlador Sistema, deverá ser responsável por gerenciar a lógica para a criação dos objetos:

- Atuadores, responsável por controlar os dispositivos de atuação do sistema, sendo eles o relé de bloqueio, *buzzer*, display LCD e LEDs;
- RTC, responsável por acessar o módulo DS1307 e resgatar o momento exato em que os abastecimentos são iniciados e finalizados;
- GPS, responsável por acessar o módulo Ublox NEO-6M e resgatar a localização do sistema com o ESP32 no momento em que foi criado o abastecimento;
- GPRS, responsável por acessar o módulo SIM800L e enviar os dados dos abastecimentos finalizados para o banco de dados na nuvem;
- MicroSD, responsável por acessar o módulo cartão microSD e armazenar os dados dos abastecimentos finalizados.

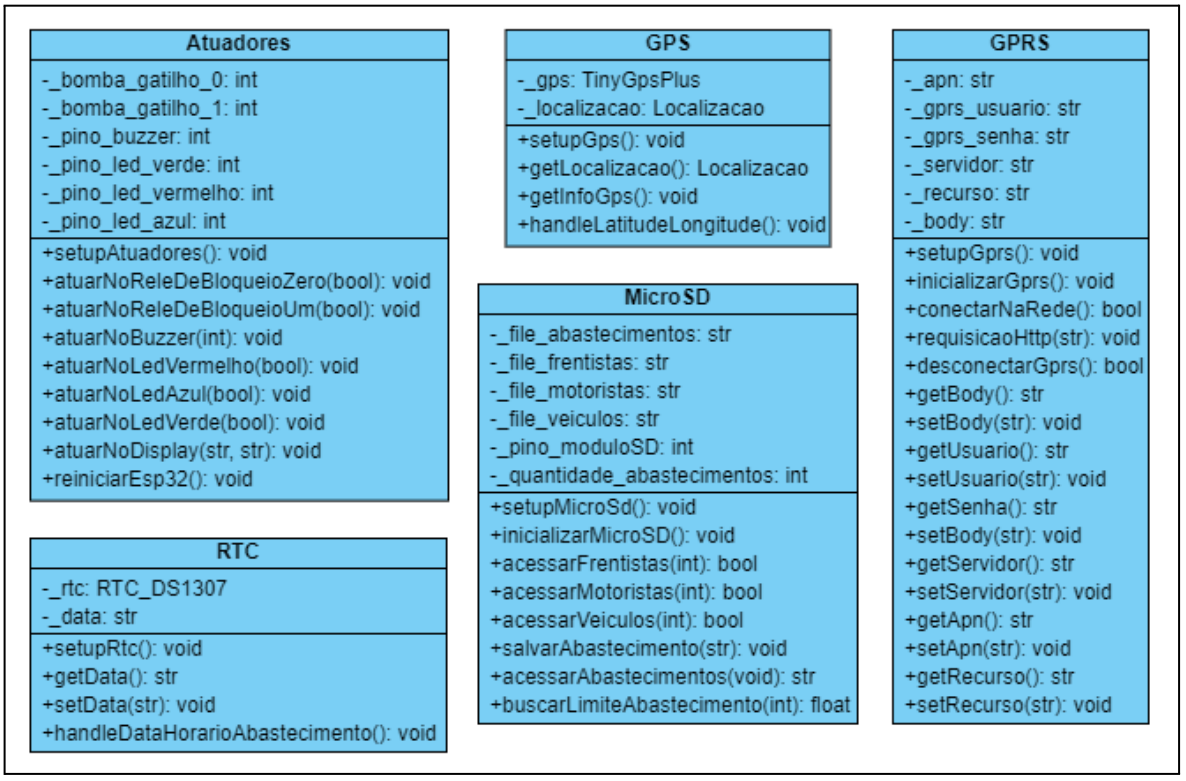
Figura 3.18 - Diagrama de sequência para o caso de uso Inicialização.



Fonte: Autor (2023).

Portanto, estes objetos serão modelados segundo o diagrama de classes da Figura 3.19, no qual é demonstrado os métodos e atributos que serão implementados.

Figura 3.19 - Diagrama de classes para o caso de uso Inicialização.

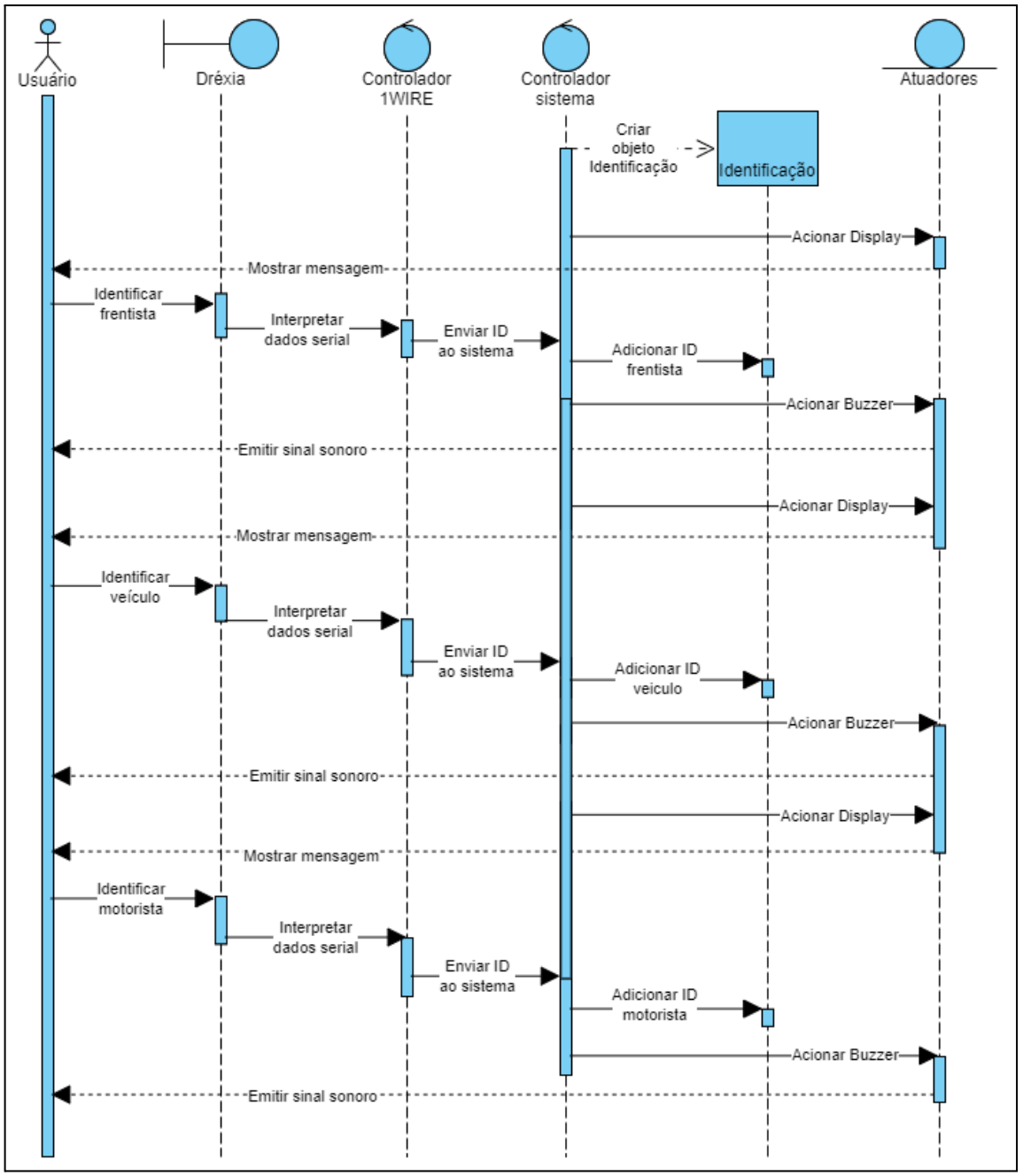


Fonte: Autor (2023).

3.4.2.2 Caso de uso Esperar Identificação

A Figura 3.20 demonstra as operações que o sistema realizará para o caso de uso Esperar Identificação. Primeiramente, o algoritmo do sistema deve ser capaz de iniciar a primeira instrução de identificação. Esta instrução consiste em um algoritmo que instancia um objeto da classe Identificação. Em seguida, o algoritmo mantém o sistema em *stand by* esperando pela identificação do primeiro dos três indivíduos na seguinte ordem: frentista, veículo e motorista. Ou seja, o sistema espera pelo usuário identificar o cartão RFID do frentista, em seguida do veículo e por fim do motorista, registrando cada um dos identificadores como atributos do objeto instanciado.

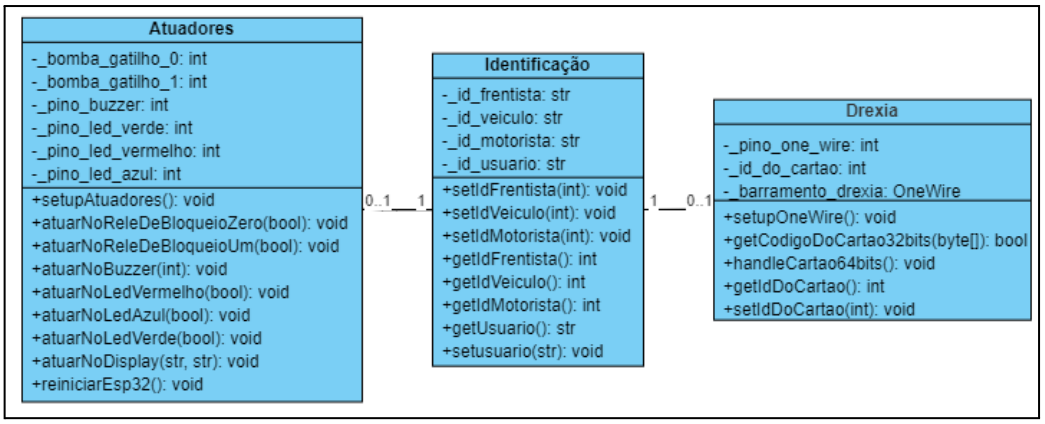
Figura 3.20 - Diagrama de sequência para o caso de uso Esperar Identificação.



Fonte: Autor (2023).

A Figura 3.21 apresenta o diagrama de classes com as funcionalidades básicas que deverão ser implementadas para cada objeto instanciado neste caso de uso.

Figura 3.21 - Diagrama de classes para o caso de uso Esperar Identificação.

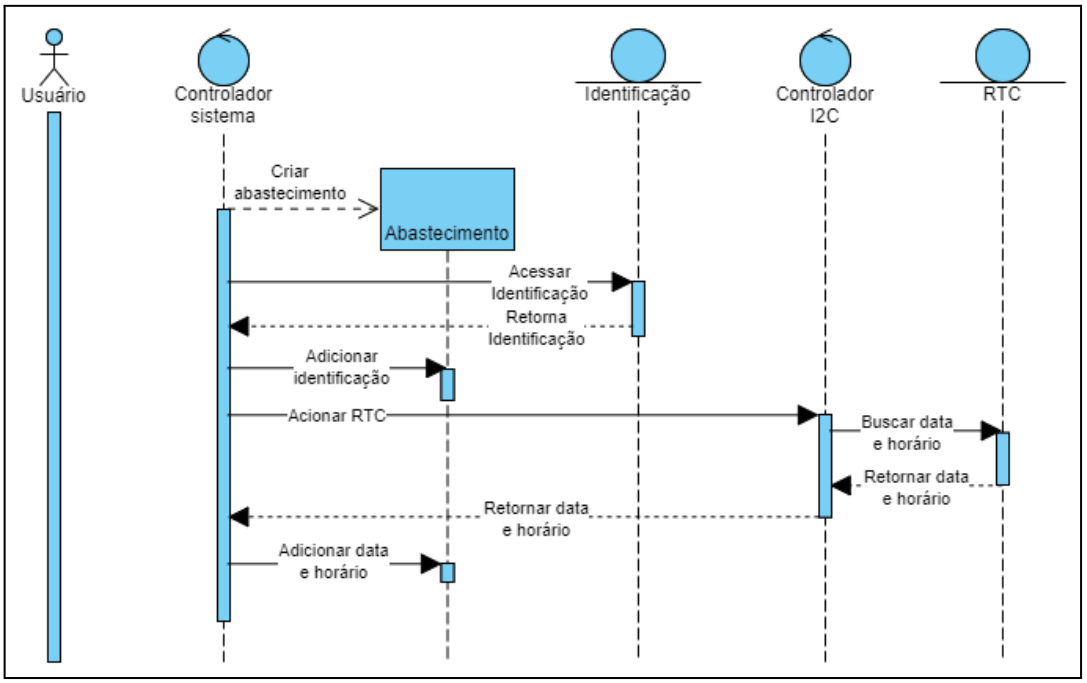


Fonte: Autor (2023).

3.4.2.3 Caso de uso Gerar Abastecimento

Após o processo de Esperar Identificação, o controlador do sistema automaticamente deverá instanciar um objeto Abastecimento que salva as configurações iniciais do processo de abastecimento. Dessa forma, o diagrama de sequência na Figura 3.22 demonstra que o objeto Identificação estará relacionado com o objeto Abastecimento, pois é necessário ter informação do usuário que está utilizando a bomba de combustível naquele instante.

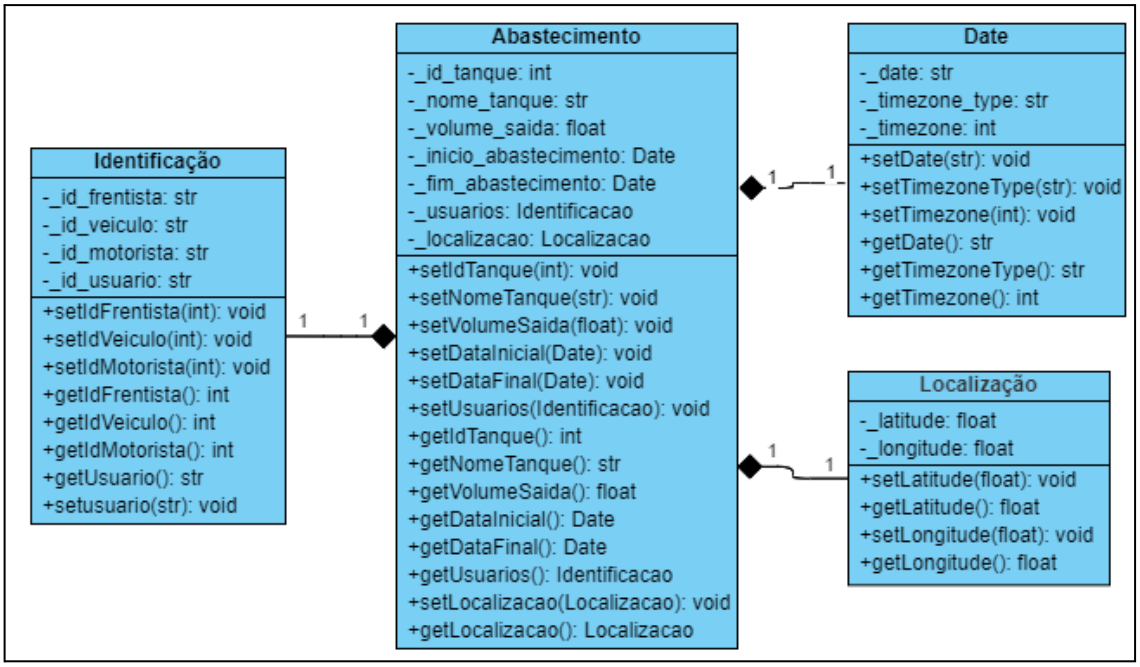
Figura 3.22 - Diagrama de sequência para o caso de uso Gerar Abastecimento.



Fonte: Autor (2023).

O diagrama de classes da Figura 3.23 apresenta todas as características que a entidade/objeto Abastecimento deverá apresentar, com destaque para as entidades/objetos Date e Localização que deverão compor o Abastecimento.

Figura 3.23 - Diagrama de classes para o caso de uso Gerar Abastecimento.

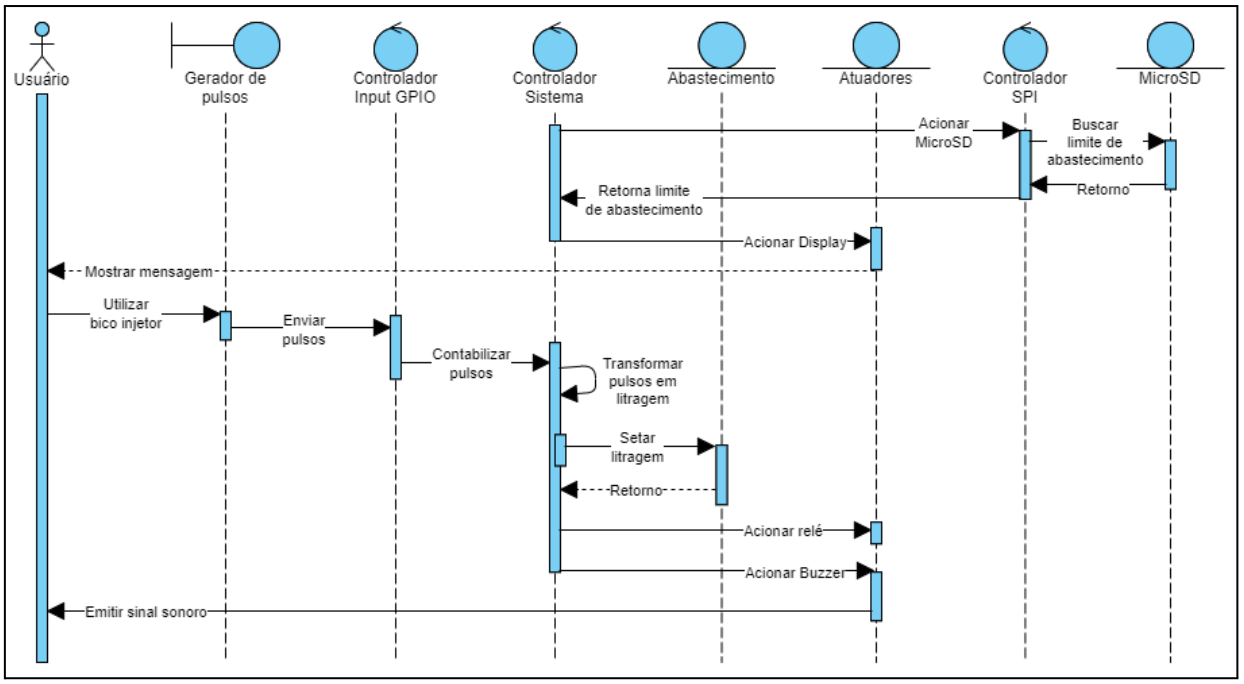


Fonte: Autor (2023).

3.4.2.4 Caso de uso Iniciar Abastecimento

A Figura 3.24 apresenta o diagrama de sequência para o caso de uso Iniciar Abastecimento, no qual procede a etapa de criação do objeto Abastecimento. Este caso de uso representa o início de fato do abastecimento, pois o volume de fluido que sai da bomba de combustível deverá ser mensurado em tempo real pelo controlador do sistema (algoritmo principal).

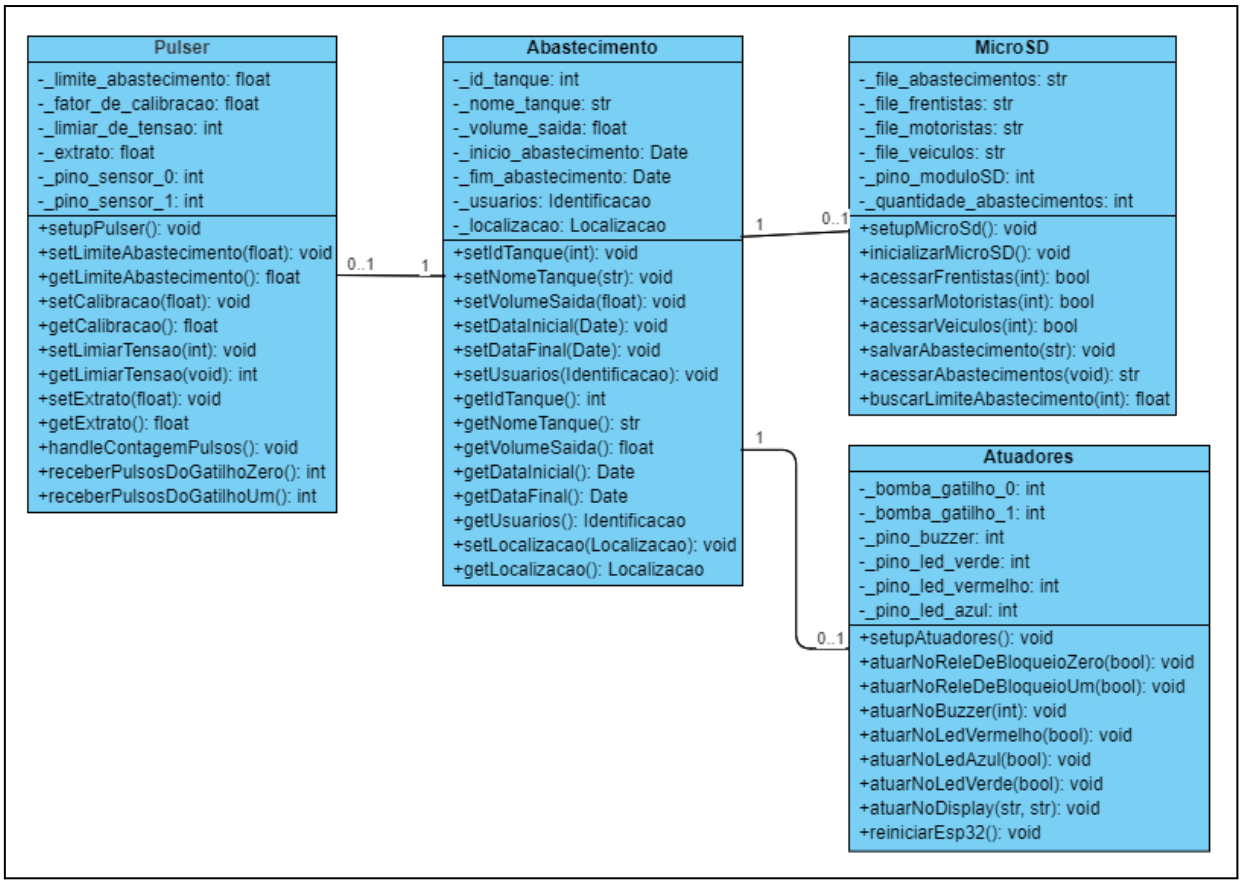
Figura 3.24 - Diagrama de sequência para o caso de uso Iniciar Abastecimento.



Fonte: Autor (2023).

Sendo assim, a Figura 3.25 apresenta a relação da entidade/objeto Abastecimento com os objetos criados no caso de uso Inicialização. Dessa forma, o controlador do sistema precisará acessar o cartão de memória onde fica salvo os dados de limite de abastecimento para o veículo identificado e direcionar a informação para a nova entidade Pulser, que controlará a contagem da quantidade dos pulsos gerados pelo sensor FBCGQ-3 e sua conversão para litros abastecidos.

Figura 3.25 - Diagrama de classes para o caso de uso Iniciar Abastecimento.



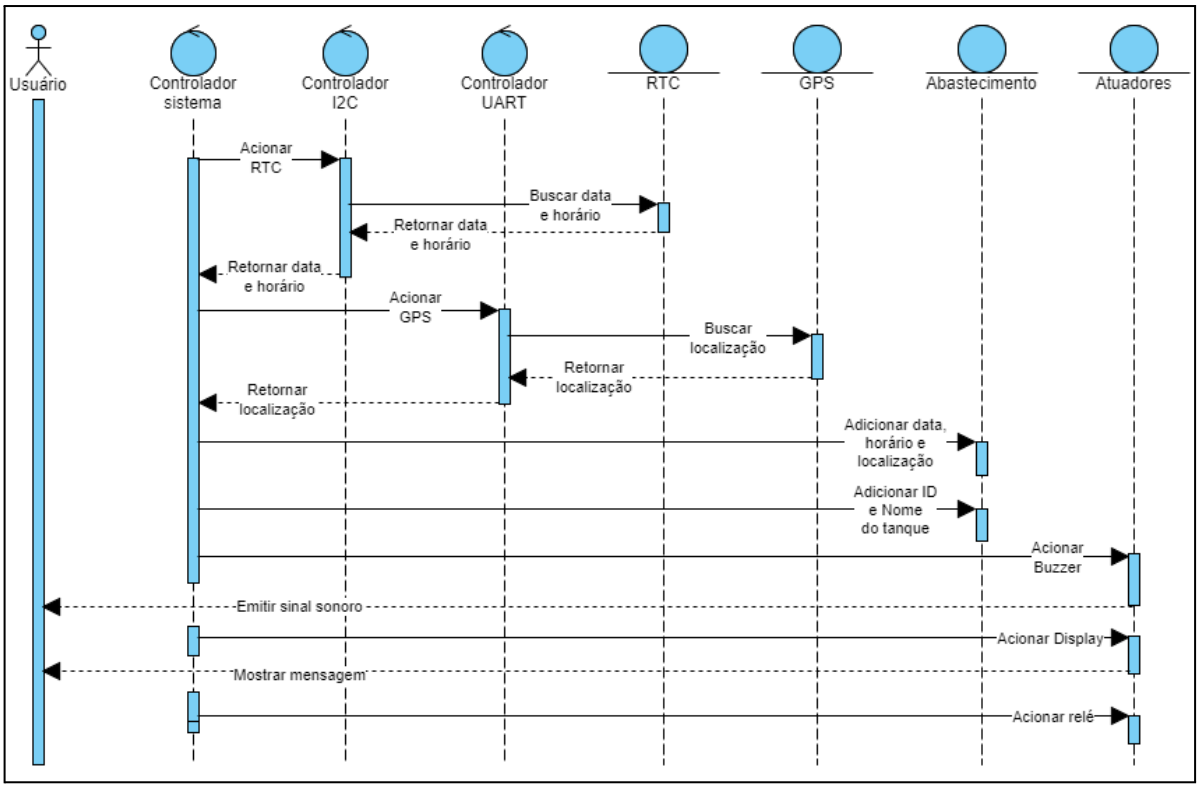
Fonte: Autor (2023).

3.4.2.5 Caso de uso Finalizar Abastecimento

Assim que um abastecimento é realizado considerando que o limite determinado para abastecer o veículo é atingido, o sistema verificará estas informações salvas no objeto Abastecimento e deve finalizar o processo conforme indicado pelo diagrama de sequência da Figura 3.26. Durante esta etapa, o controlador do sistema tem a responsabilidade de acessar os módulos de comunicação representados pelas entidades RTC, GPS e Atuadores.

Primeiramente, o Controlador I2C fica responsável por acessar o barramento I2C para resgatar os dados de data e horário, nos quais são transformados em informações de momento final de abastecimento. Em seguida, o Controlador UART tem a função de acionar a entidade GPS no sistema e requisitar dados de latitude e longitude para transformá-los em informações de localização. Por fim, a entidade Atuadores fica responsável por fechar o relé de bloqueio, impedindo que o usuário utilize o bico injetor e gere pulsos elétricos, além de notificá-lo sobre este evento.

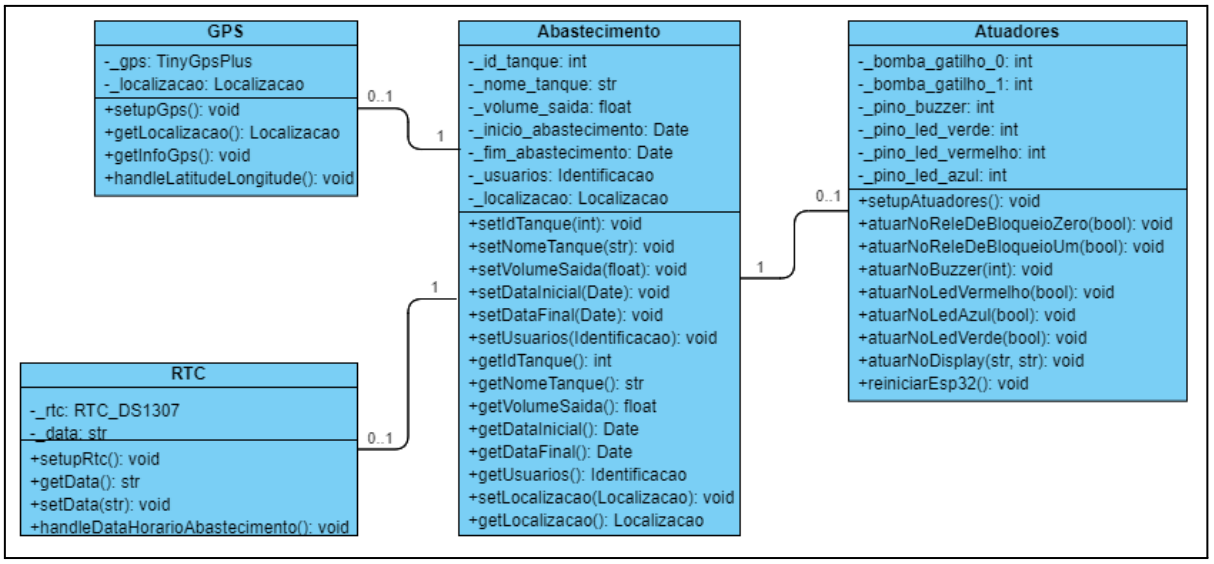
Figura 3.26 - Diagrama de sequência para o caso de uso Finalizar Abastecimento.



Fonte: Autor (2023).

A Figura 3.27 demonstra as relações entre a entidade/objeto Abastecimento e os outros objetos criados no caso de uso Inicialização.

Figura 3.27 - Diagrama de classes para o caso de uso Finalizar Abastecimento.



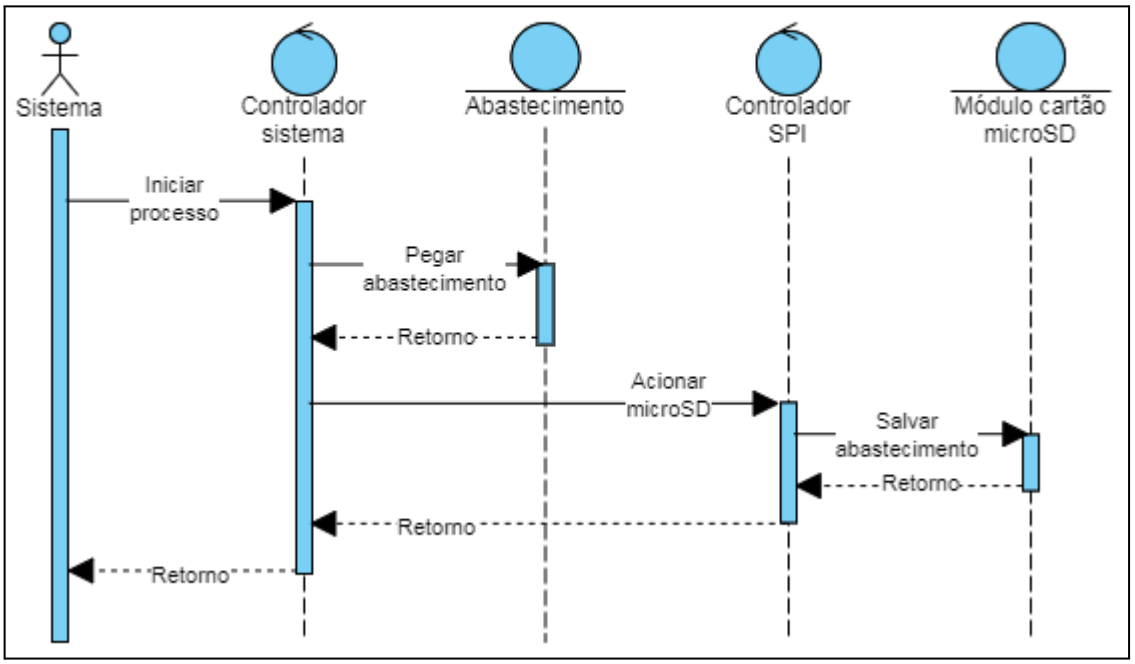
Fonte: Autor (2023).

3.4.2.6 Caso de uso Armazenar Abastecimento

A Figura 3.28 apresenta o diagrama de sequência para o caso de uso Armazenar Abastecimento, sendo que a entidade controladora do sistema tem a responsabilidade de acessar a entidade/objeto Abastecimento e persistir seus dados no módulo de cartão microSD.

Assim, o sistema será programado para realizar esta tarefa periodicamente, onde cada abastecimento realizado é salvo no cartão de memória microSD. Para isso, define-se então um novo ator Sistema que representa a automação da operação. O controlador do sistema tem acesso ao abastecimento realizado e, por meio da entidade Controlador SPI, suas informações serão salvas em um arquivo *.txt* por meio do mesmo protocolo.

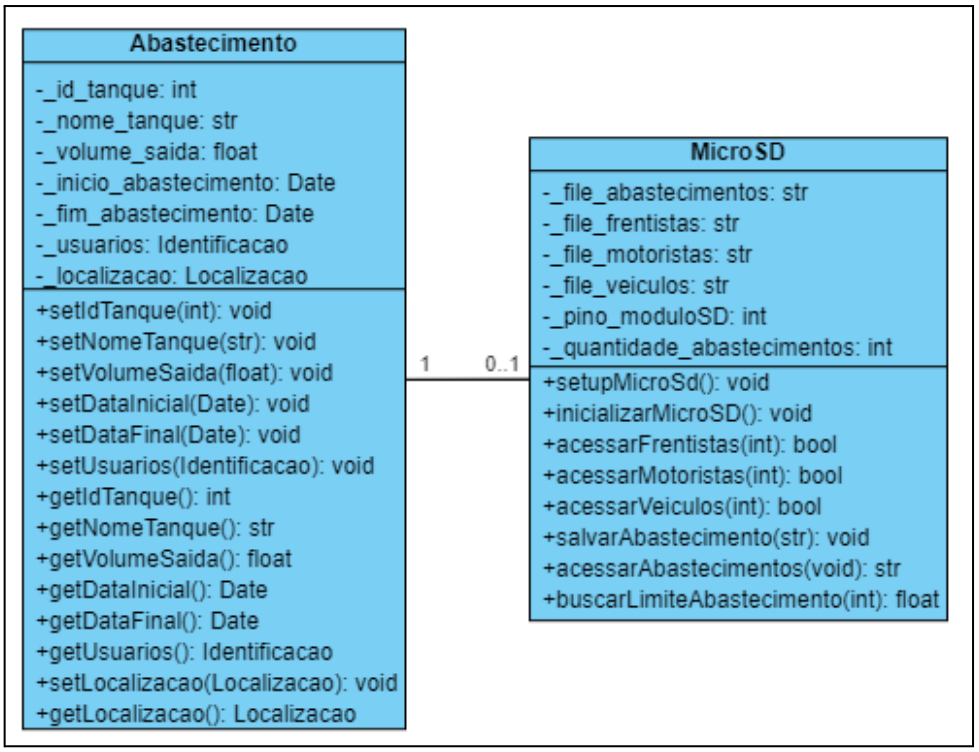
Figura 3.28 - Diagrama de sequência para o caso de uso Armazenar Abastecimento.



Fonte: Autor (2023).

A Figura 3.29 demonstra o diagrama de classes que relaciona a entidade/objeto abastecimento com a entidade MicroSD inicializada durante o caso de uso Inicialização.

Figura 3.29 - Diagrama de classes para o caso de uso Armazenar Abastecimento.



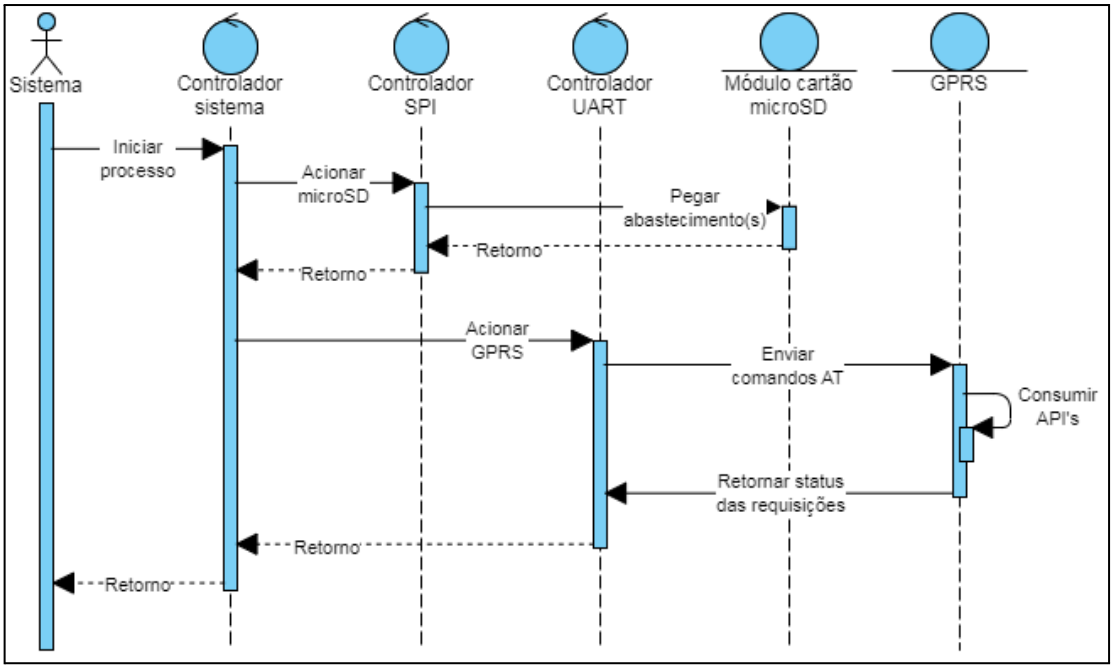
Fonte: Autor (2023).

3.4.2.7 Caso de uso Enviar Abastecimento

A metodologia para enviar um abastecimento ao banco de dados na nuvem pode ser visualizada por meio do caso de uso Enviar Abastecimento da Figura 3.30, onde o controlador do sistema executará a mesma tarefa de acesso ao módulo de cartão microSD para resgatar os dados de abastecimento. O Controlador SPI realiza a interface com o mesmo módulo e entrega ao sistema as informações necessárias para serem consumidas pelas APIs REST.

Dessa forma, o sistema aciona a entidade Controlador UART para acionar o módulo SIM800L. Sua função será utilizar a rede GPRS para consumir as APIs REST de escrita no banco de dados.

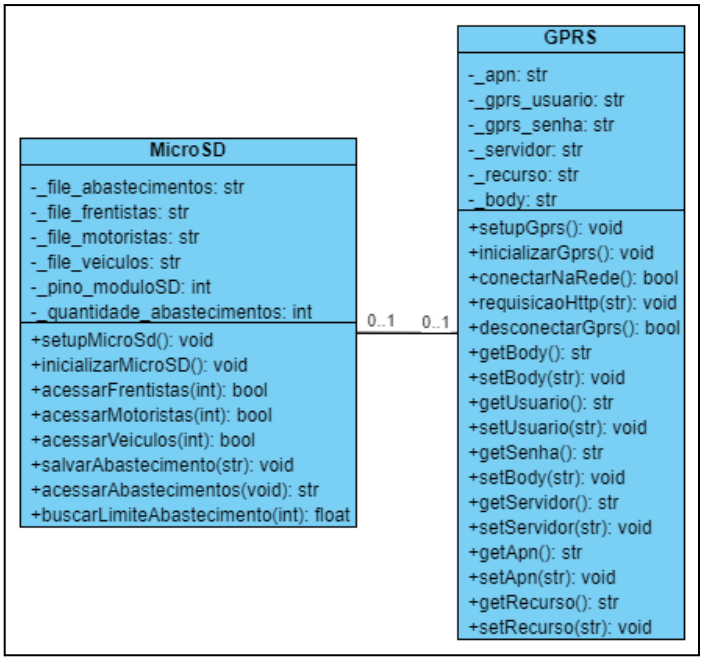
Figura 3.30 - Diagrama de sequência para o caso de uso Enviar Abastecimento.



Fonte: Autor (2023).

A Figura 3.31 apresenta a relação entre as entidades MicroSd e GPRS, onde durante o caso de uso Enviar Abastecimento, elas deverão se associar para ser possível transmitir os dados de abastecimento entre o cartão de memória e o módulo de comunicação GPRS.

Figura 3.31 - Diagrama de classes para o caso de uso Enviar Abastecimento.



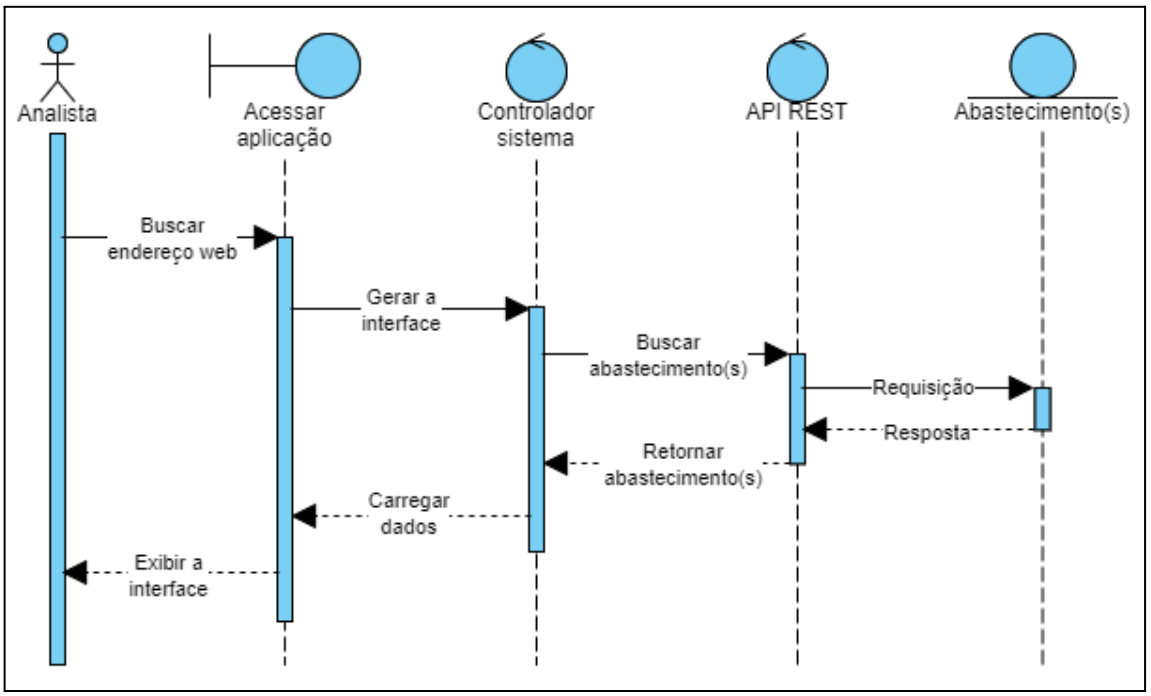
Fonte: Autor (2023).

3.4.2.8 Caso de uso Acessar Aplicação Web

O caso de uso Acessar Aplicação Web é representado por meio da Figura 3.32, no qual deste ponto em diante o ator Analista é a entidade que interage com o sistema. Esta interação se inicia no momento em que o usuário acessa a aplicação por meio de um endereço de domínio no seu browser e, em seguida, uma interface conforme o modelo apresentado na Figura 3.15 é apresentada ao usuário.

A partir do momento que a aplicação inicia o processo de renderização da página principal, por padrão o sistema realiza uma requisição utilizando a API que busca os abastecimentos no banco de dados que foram realizados no mesmo dia em que se está acessando a aplicação. A ideia é justamente ter um conjunto de informações preliminares que orientem de imediato o usuário na interface.

Figura 3.32 - Diagrama de sequência para o caso de uso Acessar Aplicação Web.



Fonte: Autor (2023).

3.4.2.9 Caso de uso Consultar Abastecimento

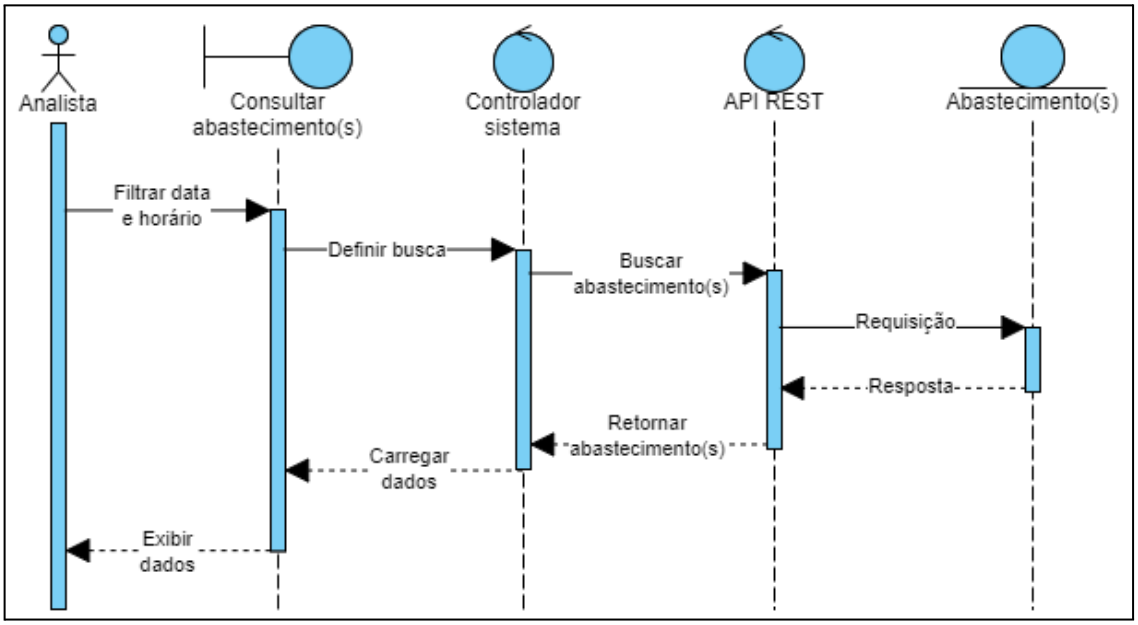
O modelo de interface é adotado inicialmente para que seja possível ao usuário buscar todos os abastecimentos realizados segundo os critérios de busca indicados no bloco *Sidebar*. Para isso, o fluxo se inicia quando é informado no bloco qual será o intervalo de tempo

desejado para a consulta, inserido por meio dos campos *data inicial* e *data final*, conforme o fluxo indicado pela Figura 3.33.

Após a verificação dos campos e o pressionamento do botão Consultar, o sistema age enviando uma requisição ao servidor que hospeda o banco de dados, retornando uma resposta para a consulta.

Por fim, o sistema assume a responsabilidade de apresentar os dados retornados no bloco *Home* da aplicação ao usuário. Este bloco, assim como os outros implementados, estão presentes na página *Listagem*. Além disso, os dois blocos complementares *Header* e *Footer* apresentam ao usuário, respectivamente, informações de quem está logado na aplicação e dicas de uso, indicados pelo *layout* ilustrado na Figura 3.15.

Figura 3.33 - Diagrama de sequência para o caso de uso Consultar Abastecimento.



Fonte: Autor (2023).

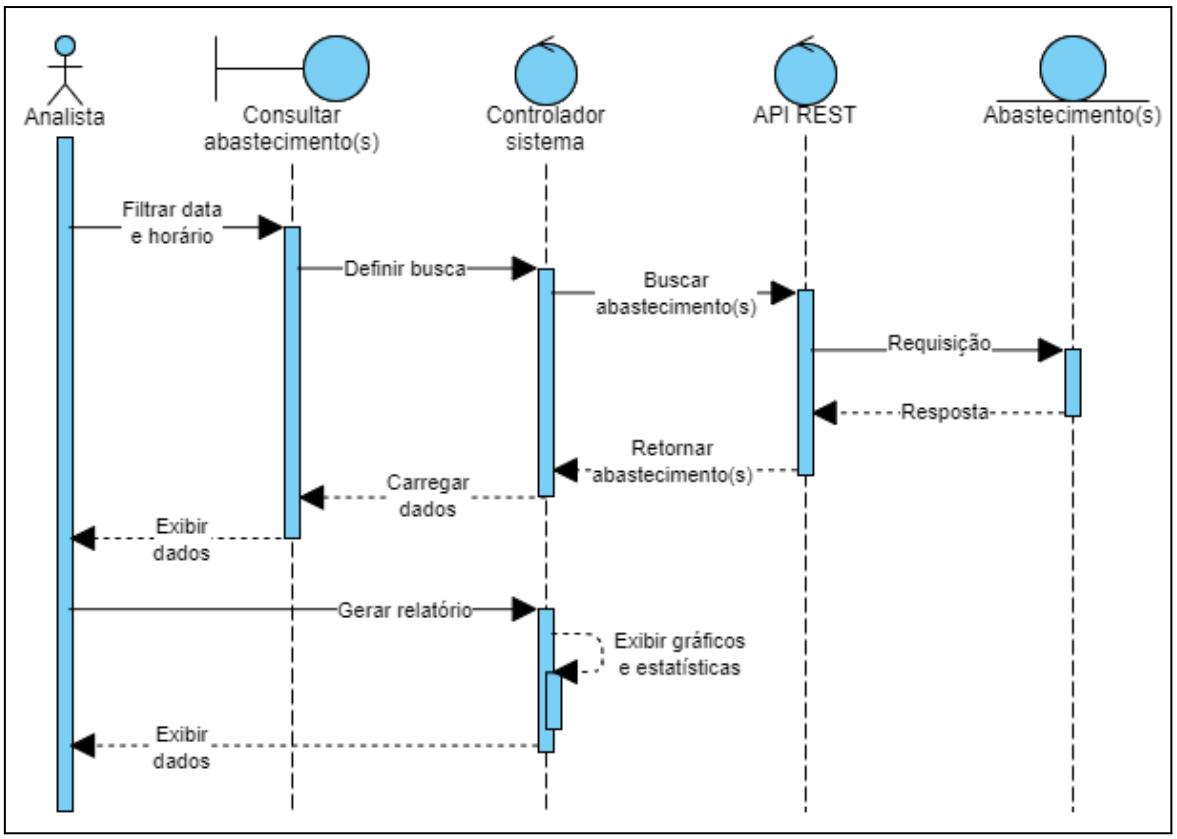
3.4.2.10 Caso de uso Gerar Relatórios

Conforme pode ser visualizado na Figura 3.34, o caso de uso Gerar Relatórios é um complemento do caso de uso Consultar Abastecimento, visto que a diferença se fundamenta na chamada a uma nova página *Dashboard*, no qual são disponibilizados as mesmas informações dos abastecimentos, porém trabalhadas de forma estatística. O objetivo é entregar ao ator Analista um relatório formado por dados gráficos e estatísticos para tomadas de decisão, dentro do intervalo de tempo de pesquisa.

A Figura 3.16 apresenta uma ilustração do layout que deve ser apresentado ao usuário assim que uma consulta for realizada, onde são apresentados 2 modelos diferentes de blocos:

- O primeiro modelo destina-se a apresentar dados estatísticos relacionados aos tanques, como por exemplo:
    a) quantidade de combustível que entrou no tanque, em litros;
    b) quantidade de combustível retirados do tanque, em litros;
    c) saldo final, em litros.
- O segundo modelo complementa o anterior com a ideia de trabalhar os dados dos tanques de combustível juntamente aos dados de abastecimentos retornados das APIs, como por exemplo:
    a) a quantidade de eventos de retiradas por período de tempo;
    b) a quantidade de eventos de entrada de combustível por período de tempo;
    c) saldo do extrato de combustível por período de tempo;
    d) gráfico com a quantidade de combustível retirado por dia, em litros;
    e) gráfico com a quantidade de combustível que entrou no tanque por dia, em litros;
    f) gráfico com a quantidade total de abastecimentos realizados, por dia;
    g) quantidade de abastecimentos por veículo.

Figura 3.34 - Diagrama de sequência para o caso de uso Gerar Relatórios.



Fonte: Autor (2023).

4 RESULTADOS E DISCUSSÃO

Para avaliar o funcionamento da automação do sistema foram realizados a montagem do equipamento e alguns testes finais por meio de uma simulação de abastecimento utilizando o sensor gerador de pulsos FBCGQ-3. Assim que cada módulo no sistema começa a operar, alguns dados são coletados para validar os objetivos propostos.

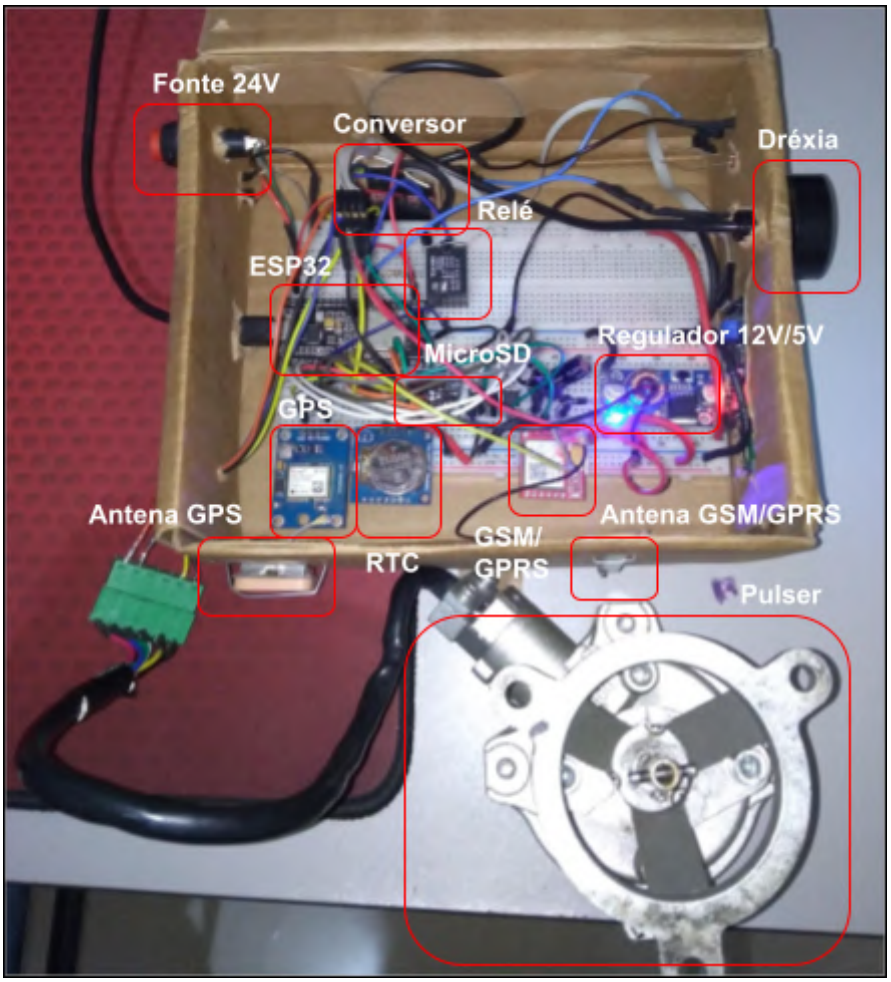
Assim, para complementar os resultados nos testes são apresentadas informações sobre a resposta de cada módulo no sistema composto pela placa ESP32 por meio da Arduino IDE, como também o comportamento das classes implementadas para o funcionamento do software. Por fim, estas informações são diagnosticadas dentro da aplicação web por meio de seus recursos de listagem de abastecimentos e geração de relatórios gráficos.

4.1 Montagem do Hardware

A montagem do sistema resultou em um primeiro protótipo com todos os módulos de alimentação, de comunicação, sensores e atuadores apresentados na Tabela 3.1 conectados na placa ESP32, conforme apresentado pela Figura 4.1. As conexões dos terminais foram realizadas em uma protoboard grande e alguns jumpers com combinação de cores específicas como, por exemplo:

- Vermelho para qualquer terminal relacionado a fonte de alimentação de 12V e também para as tensões reguladas de 5V e 3,3V (VCC);
- Preto para os terminais de aterramento (GND) ou ao negativo da fonte de alimentação (-VCC);
- Azul para o terminal do barramento *One Wire* (1-WIRE) ou para recebimento de dados de leitura ao ESP32 (INPUT);
- Branco para envio de dados de escrita do ESP32 (OUTPUT);
- Amarelo e verde para os respectivos terminais RX e TX ou para os terminais SDA e SCL (UART e I2C);
- Laranja para os terminais MISO, MOSI e CLK (SPI).

Figura 4.1 - Montagem do primeiro protótipo em uma protoboard dentro de uma caixa, indicando os principais módulos do sistema.

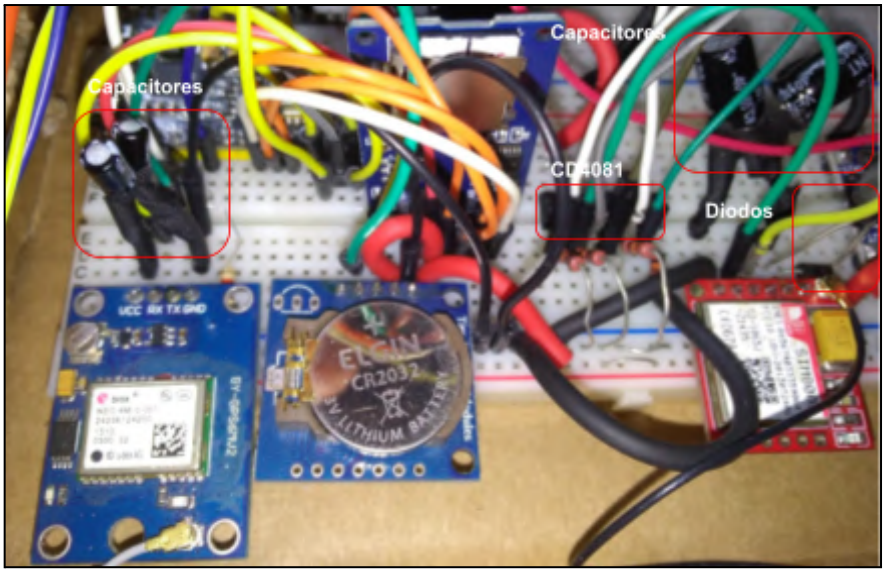


Fonte: Autor (2023).

Observa-se pela Figura 4.2 que os módulos de comunicação foram conectados próximos entre si, porém nenhum ruído ou interferência de sinal foi observado, além de que cada dispositivo possui sua antena transmissora/receptora de sinais, reforçando assim a comunicação. Além disso, seguiu-se com o padrão de cores definido.

É possível observar pela mesma figura que foi adicionado um conjunto de dois diodos 1N4007 ligados em paralelo, conectado em série com o terminal VCC do módulo GSM/GPRS, além de uma associação de capacitores eletrolíticos totalizando 940µF, conectados em paralelo ao terminal VCC tanto do módulo SIM800L quanto do módulo Ublox NEO-6M. Segundo SHYMANSKYY (2023), este processo pode ser feito principalmente no módulo SIM800L para que o mesmo seja energizado com uma tensão de 4,3V e passivo de qualquer tentativa de reinicialização no seu momento de uso mais extremo.

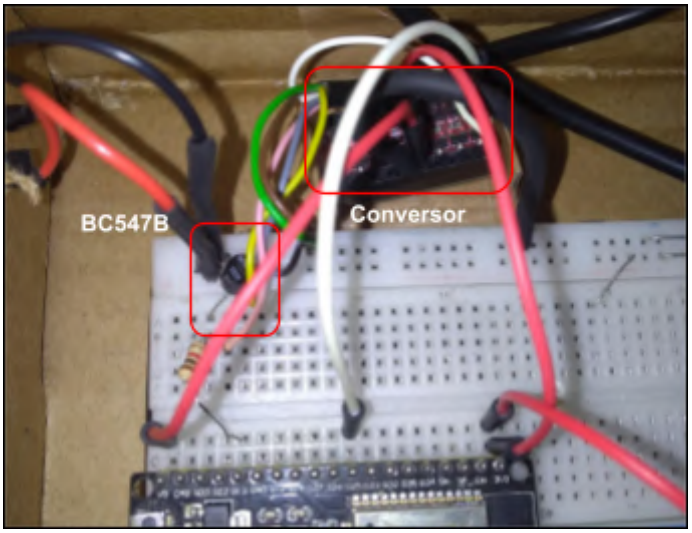
Figura 4.2 - Módulos de comunicação conectados na placa ESP32.



Fonte: Autor (2023).

A Figura 4.3 apresenta a conexão realizada para o sensor de identificação RFID Dréxia, no qual o barramento 1-WIRE é identificado por meio da cor branca. O conversor de nível lógico intercepta o barramento para converter a tensão de 5V para 3.3V com o objetivo de não danificar a GPIO25 do ESP32. O transistor BC548B indicado foi utilizado para chavear o *buzzer*.

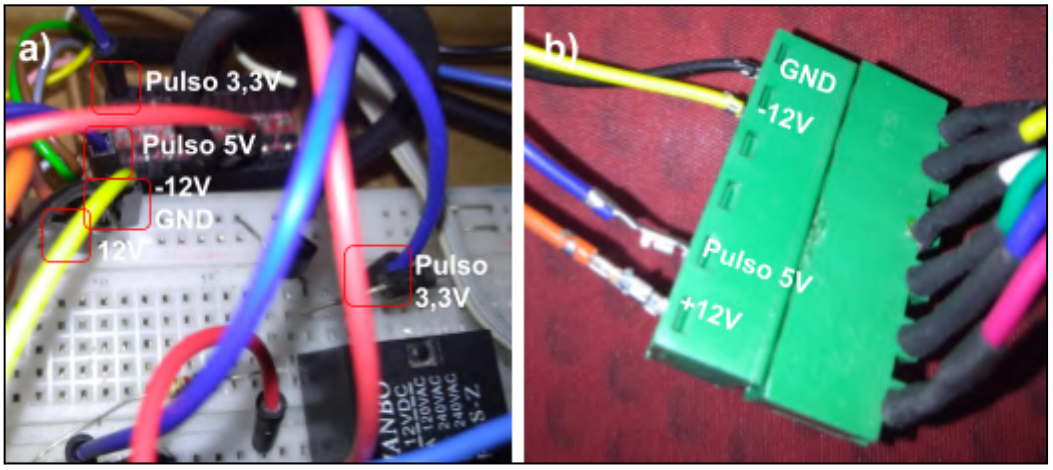
Figura 4.3 - Sensor Dréxia conectado ao conversor de nível lógico e na placa ESP32.



Fonte: Autor (2023).

A Figura 4.4 demonstra a conexão do terminal de dados do sensor gerador de pulsos FBCGQ-3 na placa ESP32. Para isso, um conector de borda do modelo 2EDG6C (com 6 vias de 90°) foi adicionado para facilitar a conexão entre todos os terminais do sensor (alimentação, aterramento e dados) com o ESP32.

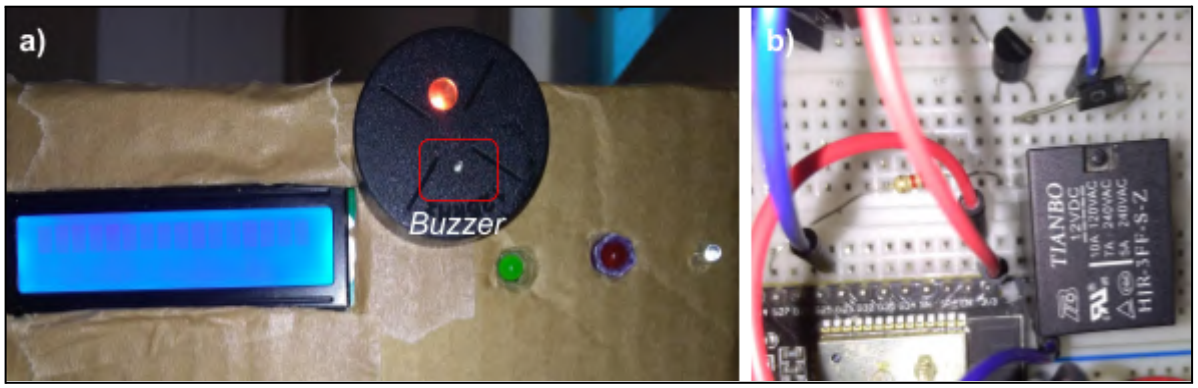
Figura 4.4 - Sensor gerador de pulsos conectado à placa ESP32, onde a) conexões na protoboard e b) conexão no terminal do sensor.



Fonte: Autor (2023).

Por fim, as conexões dos atuadores na placa ESP32 pode ser vista por meio da Figura 4.5, onde se encontra o relé de bloqueio, *buzzer*, *display* LCD e LEDs. Sendo assim, tem-se um sistema que identifica usuários, realiza controle de acesso, gera dados de abastecimento e notifica o usuário no decorrer do seu uso.

Figura 4.5 - Conexão dos atuadores na placa ESP32, onde a) *buzzer*, *display* LCD e LEDs e b) relé de bloqueio.



Fonte: Autor (2023).

No decorrer da montagem do equipamento, os dois problemas mais enfrentados foram: esperar os módulos SIM800L (GSM/GPRS) e Ublox NEO-6M (GPS) pegar sinal e dar manutenção na *protoboard*. Em alguns testes, era necessário esperar em média 10 minutos para os módulos conseguirem se conectar na rede. Isso não seria um problema em campo, pois, naturalmente, um dispositivo móvel demora alguns minutos para se conectar na rede. Porém, se tornava exaustivo quando precisava testá-los após uma reinicialização do ESP32. Além disso, conforme era necessário realocar algum sensor, atuador ou módulo na *protoboard*, existia muita dificuldade devido a quantidade de *jumpers*, o que poderia causar interferência no hardware se desconectado sem atenção.

4.2 Implementação do Software no ESP32

O resultado do desenvolvimento do software para a placa ESP32, segundo a metodologia ICONIX adotada na metodologia do projeto, são apresentados no Apêndice A por meio de seus arquivos em *.cpp* e *.h* já em produção. Estes arquivos apresentam as instruções de cada funcionalidade dos seguintes elementos:

- entidades do software Identificação, Abastecimento, Data, Localização, GPS, GPRS, MicroSD, RTC e Atuadores;
- elementos de borda Drexia e Pulser;
- controladores das APIs REST, SPI, UART, I2C e GPIOs.

Primeiramente, a lógica de programação implementada em C++ para o ESP32 é inserida em seu microcontrolador por meio de um cabo USB na sua porta Serial. Assim que a placa é alimentada pelo mesmo cabo, observa-se que o ESP32 é energizado. Neste momento, o microcontrolador processa o algoritmo principal, representado pelo controlador do sistema nos diagramas de classe de uso, iniciando com as instruções de importar todas as classes desenvolvidas, inclusive as bibliotecas ArduinoJSON, TinyGSM, HttpClient, OneWire, RTClib e TinyGPSPlus, nos quais são necessárias para o funcionamento de cada classe implementada. Dessa forma, o Requisito Funcional RF-01, conforme apresentado pela Tabela 3.2, é atendido garantindo que os protocolos de comunicação, ou controladores apresentados nos casos de uso, funcionem corretamente.

A entidade Identificação (Apêndice A.1), garante que o Requisito Funcional RF-02 seja atendido. Isso acontece pois o objeto criado registra o número que representa a identificação do frentista, motorista e veículo, para que posteriormente as mesmas informações sejam associadas ao objeto Abastecimento. Além disso, a entidade Dréxia

(Apêndice A.5) possibilita resgatar os identificadores por meio do leitor RFID pela funcionalidade *getIdDoCartao64bits()*.

Em seguida, o sistema inicializa o abastecimento criando a entidade/objeto Abastecimento (Apêndice A.2), o que cumpre o Requisito Funcional RF-03. Nesta etapa, o sistema associa a Identificação anterior ao objeto Abastecimento e registra o momento inicial por meio da funcionalidade *handleDataHorarioAbastecimento()* pela entidade RTC (Apêndice A.11), no qual implementa o protocolo I2C pela biblioteca RTClib e armazena a data com o horário do momento que foi chamada dentro da entidade/objeto Data (Apêndice A.3), atendendo assim o Requisito Funcional RF-06.

Gerado o abastecimento, o sistema tem acesso a quantidade de volume definida para o veículo, segundo o seu identificador RFID, acessando o arquivo *veiculos.txt* dentro da entidade MicroSD (Apêndice A.10) pela funcionalidade *buscarLimiteAbastecimento()*. Assim tem-se uma referência e controle da quantidade de combustível a abastecer para qualquer veículo, sendo que este arquivo pode ser atualizado manualmente ou automaticamente por meio da API que acessa os veículos registrados na aplicação web.

Complementando, a entidade Pulser (Apêndice A.6) garante a validação do quarto Requisito Funcional RF-04, pois a funcionalidade *handleContagemPulsos()* faz com que o sistema verifique repetidamente a entrada analógica associada (GPIO4), calculando os pulsos recebidos pelo sensor FBCGQ-3 em litros abastecidos até que o volume de saída da bomba seja o mesmo definido para o limite de abastecimento do veículo. Além disso, a entidade Atuadores (Apêndice A.7) tem sua função em bloquear a bomba de combustível assim que o abastecimento é finalizado, notificando o usuário sobre o evento ocorrido.

A entidade GPS (Apêndice A.8) tem a função de implementar o protocolo UART e entregar ao sistema os dados de localização em latitude e longitude, nos quais são salvas na entidade/objeto Localização (Apêndice A.4) e posteriormente compondo a entidade Abastecimento no momento que o abastecimento é finalizado. Logo, os Requisitos Funcionais RF-05 e RF-07 também são cumpridos, pelo fato de que é possível localizar onde ocorreu o abastecimento, adicionando informações do identificador e nome da bomba de combustível utilizada.

A entidade MicroSD (Apêndice A.10) fica responsável por salvar o abastecimento finalizado dentro do arquivo *abastecimentos.txt*, atendendo assim o Requisito Funcional RF-08. Dessa forma, o sistema persiste dados até o momento que são enviados para o servidor web por meio da entidade GPRS (Apêndice A.9) pela funcionalidade *requisicaoHttp()* e

disponibilizados na aplicação de monitoramento, garantindo também o Requisito Funcional RF-09.

A aplicação web do sistema foi desenvolvida utilizando a tecnologia React.JS para possibilitar o acesso aos dados dos abastecimentos realizados. Um domínio de teste foi utilizado para que seja demonstrado a opção de consulta segundo um período inicial e final de tempo definido pelo usuário na aplicação, atendendo assim aos Requisitos Funcionais RF-10 e RF-11.

Por fim, a aplicação web disponibiliza em sua barra lateral duas opções de visualização, sendo elas uma página de Listagem e outra página de *Dashboard*. Nesta segunda opção, é possível visualizar os abastecimentos no formato de gráficos e relatórios trabalhados de forma estatística, conforme a necessidade do Requisito Funcional RF-12.

Vale ressaltar que algumas dificuldades foram enfrentadas durante o desenvolvimento do software do sistema, principalmente para o ESP32. A princípio foi pensado em criar funcionalidades que pudessem interpretar todos os protocolos de comunicação e barramentos seriais do ESP32, porém o projeto ficou estagnado por muito tempo nesta etapa. Assim, a solução foi utilizar bibliotecas externas. Dessa forma, muito tempo foi investido para entender como elas funcionavam, seguido pela necessidade de utilizar a linguagem C++ para associá-las com as classes criadas na fase de implementação da metodologia ICONIX adotada. Portanto, foi necessário corrigir um número cansativo de erros de compilação.

4.4 Teste de integração

O teste de integração foi feito com o intuito de avaliar o comportamento do sistema assim que um usuário se identifica no sensor RFID Dréxia e realiza um abastecimento. Dessa forma, o que se buscou neste teste foi observar o abastecimento na aplicação web, juntamente com os dados do usuário (frentista, motorista e veículo) identificado e do abastecimento feito, considerando os dados de identificação do tanque, quantidade retirada da bomba, data e localização.

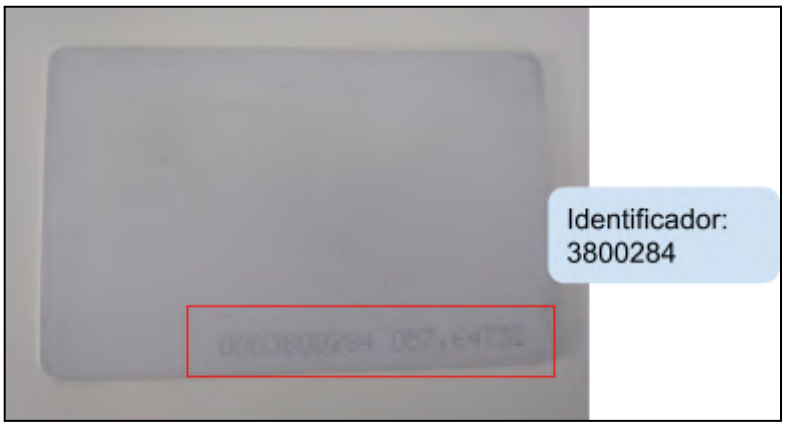
Assim que o sistema é iniciado, uma mensagem é apresentada no display LCD exigindo a identificação do frentista, conforme a Figura 4.6.

Figura 4.6 - Mensagem exigindo a identificação do frentista.

Fonte: Autor (2023)

Para isso, estes indivíduos são associados a um cartão de identificação RFID com código único de 5 a 8 dígitos, conforme indicado pela Figura 4.7.

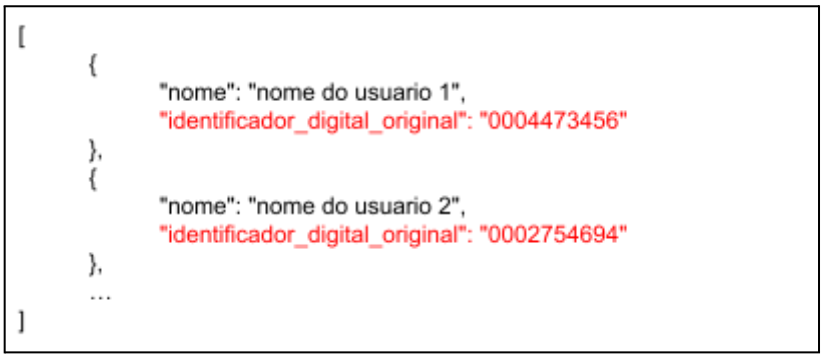
Figura 4.7 - Cartão de identificação RFID do frentista e seu identificador associado.



Fonte: Autor (2023).

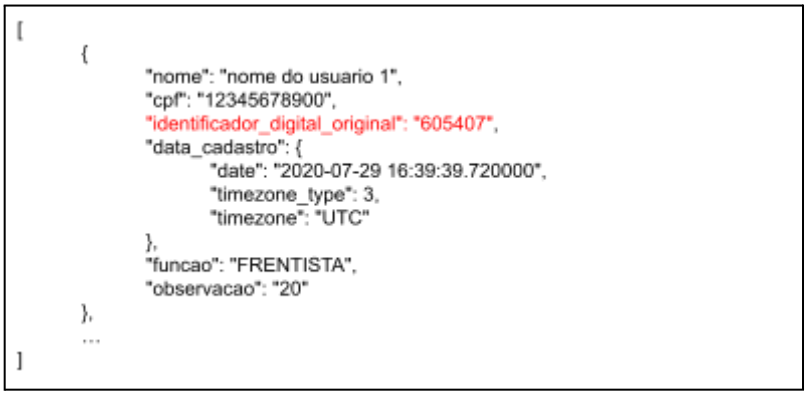
A associação é feita anteriormente dentro do banco de dados, dessa forma o sistema resgata os dados dos usuários via API REST e os utiliza como condição para realizar o controle de acesso e liberar o abastecimento. Os dados são retornados pelo servidor no formato JSON seguindo a estrutura da Figura 4.8 para motoristas, Figura 4.9 para frentistas e Figura 4.10 para veículos. Logo, é necessário que os mesmos dados persistam no sistema, sendo possível armazená-los dentro do cartão de memória microSD.

Figura 4.8 - Estrutura de dados retornada em formato JSON identificando dois motoristas, em vermelho.



Fonte: Autor (2023).

Figura 4.9 - Estrutura de dados retornada em formato JSON identificando um frentista, em vermelho.



Fonte: Autor (2023).

Figura 4.10 - Estrutura de dados retornada em formato JSON identificando um veículo, seu nome/placa, volume máximo de abastecimento (em litros) e identificador, em vermelho.


```
[
    {
        "placa": "FRT4123",
        "data_cadastro": {
            "date": "2020-05-22 09:48:46.857000",
            "timezone_type": 3,
            "timezone": "UTC"
        },
        "data_alteracao": {
            "date": "2022-01-11 14:36:49.720000",
            "timezone_type": 3,
            "timezone": "UTC"
        },
        "limite_abastecimento": 2000,
        "prazo_proximo_abastecimento": 1,
        "tipo_veiculo": "CAVALO",
        "data_desativacao": null,
        "identificador_digital_original": "0003903928"
    },
    ...
]
```


Fonte: Autor (2023).

A entidade Dréxia indicada pela Figura 3.18 é responsável por ler o identificador do cartão RFID e enviar a informação lida através do barramento 1-WIRE até o ESP32. Sendo assim, um processo de identificação no barramento tem o seguinte resultado apresentado na Figura 4.11.

Figura 4.11 - Exemplo de leitura do cartão com identificador 3800284, realizada pelo Dréxia e enviada pelo barramento 1-WIRE até o ESP32.

```
Encontrado dispositivo:
0x01, 0xDC, 0xFC, 0x39, 0x00, 0x08, 0x00, 0xAE

CHECKSUM: 174
CONST VALUE: 0
MANUFACTURER: 8
CARD DATA: 3800284
CODE DS1990A: 1
```

Fonte: Autor (2023).

O dispositivo apresenta uma tag de 64 bits agregando em sua estrutura a seguinte ordem: o *Checksum* ou *Control sum* dos dados em 1 byte, uma constante fixa (*Constant value*)

de 1 byte, o código do fabricante (*Producer code*) em 1 byte, o código do dispositivo de emulação (*Code DS1990A*) em 1 byte e os dados de identificação (*Serial number*) em 4 bytes, sendo esta última a informação de interesse, conforme a Figura 4.12. Ou seja, o valor de interesse foi o hexadecimal 0x39FCDC, ou 3800284 em decimal.

Figura 4.12 - Formato dos dados no barramento 1-WIRE, onde o *Serial number*, sublinhado em vermelho, é o dado de interesse.

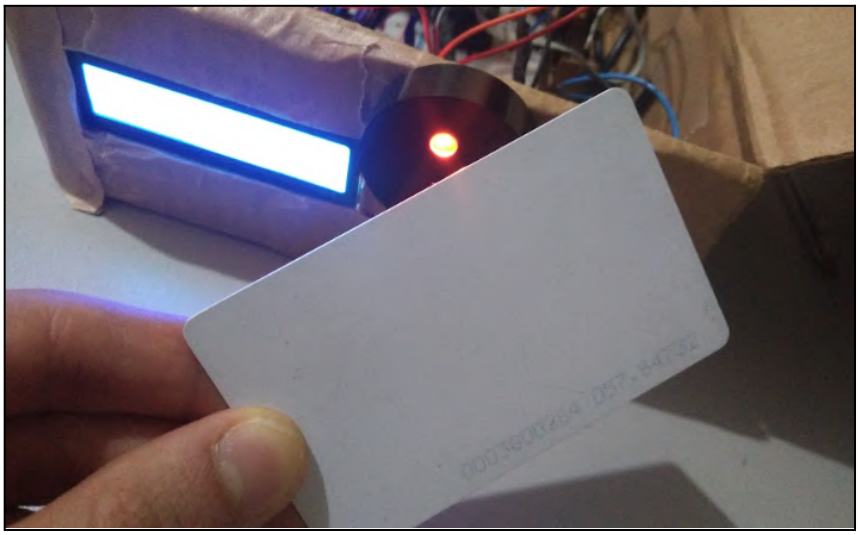
Control sum	Constant value	Producer code	Serial no.	Code DS1990A
CRC	0x00	1 byte	4 bytes	0x01
MSB				LSB

Fonte: Adaptado de Drexia (2022).

O controlador do sistema exigiu que o usuário se identificasse três vezes, sendo que a cada momento de identificação os dados do cartão (*Serial number*) foram salvos no sistema, acionado o *buzzer* para informar o usuário sobre o evento.

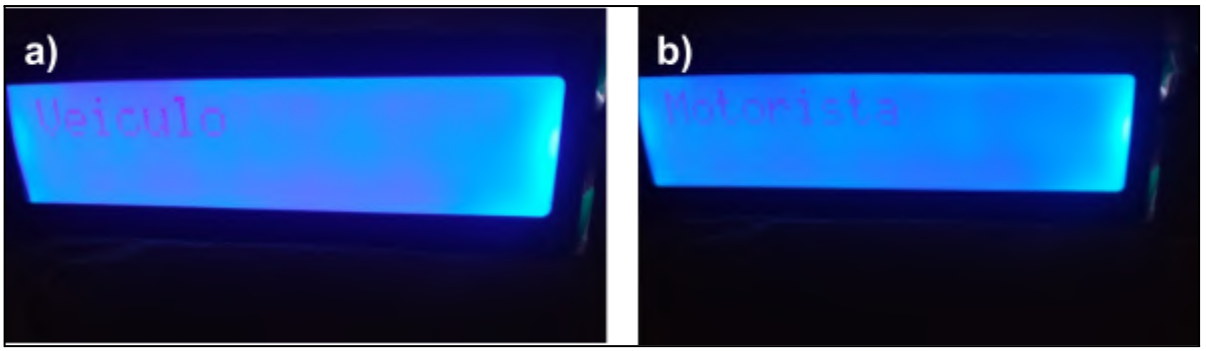
Dessa forma, o frentista se identificou pelo leitor RFID com seu cartão de identificação já cadastrado no sistema, conforme apresentado na Figura 4.13, seguido pela identificação do veículo e do motorista, respectivamente, indicado pela Figura 4.14.

Figura 4.13 - Processo realizado para a identificação do usuário.



Fonte: Autor (2023).

Figura 4.14 - Mensagem apresentada para a identificação: a) do veículo e b) do motorista.



Fonte: Autor (2023).

Por exemplo, como resposta na identificação do frentista, o sistema emitiu um sinal sonoro pelo *buzzer* e diagnosticou o usuário com a mensagem “Identificado! 3800284” no Display LCD, como pode ser visto na Figura 4.15.

Figura 4.15 - Mensagem de usuário identificado.

Fonte: Autor (2023).

Em seguida, o sistema acessou o módulo DS1307 (RTC) para buscar dados de horário e data, nos quais foram transformados em informações do momento em que se gerou o abastecimento. A Figura 4.16 ilustra o retorno da funcionalidade que acessa o barramento I2C e busca no módulo DS1307 o horário e a data nos instantes em que foi requisitado.

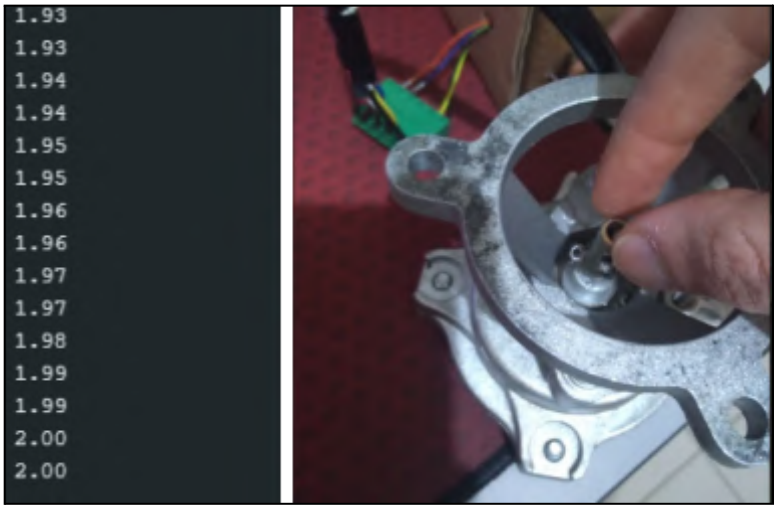
Figura 4.16 - Retorno da funcionalidade que acessa o módulo RTC DS1307.

```
Imprimindo Horário e Data
Hora: 10:25:47
Data: 20-3-2023
Imprimindo Horário e Data
Hora: 10:25:48
Data: 20-3-2023
Imprimindo Horário e Data
Hora: 10:25:49
Data: 20-3-2023
```

Fonte: Autor (2023).

Por fim, o controlador do sistema abre o relé de bloqueio liberando os gatilhos da bomba de combustível. Assim, ele acionou o *buzzer* emitindo 3 *beeps* com a mensagem "Realize o abastecimento", permitindo ao usuário utilizar o bico injetor e acionando o relé de bloqueio. É possível observar pela Figura 4.17 um diagnóstico do sistema contabilizando a quantidade de combustível em tempo real, calculada com o suporte do gerador de pulsos, até diagnosticar a mensagem "Extrato", juntamente com o total abastecido, conforme a Figura 4.18, no qual simulou-se um abastecimento de 2 litros. Como o fim do abastecimento, o sistema bloqueou o uso do bico injetor de combustível fechando os contatos do relé de bloqueio, diagnosticando a mensagem "Bomba bloqueada". Consequentemente, o sistema não permite mais utilizar o bico injetor até que um novo processo de identificação seja realizado.

Figura 4.17 - Diagnóstico em tempo real, pela plataforma Arduino IDE, da quantidade de combustível calculada pelo sistema a partir do sensor gerador de pulsos.



Fonte: Autor (2023).

Figura 4.18 - Simulando um abastecimento de 2 litros.

Fonte: Autor (2023).

O terminal do sensor que envia os pulsos elétricos foi conectado na GPIO4 da placa ESP32, no qual foi possível observar a contagem de pulsos elétricos recebidos. Em paralelo, o sistema transformou os pulsos contabilizados em uma quantidade de volume abastecido segundo o fator de calibração de 0,005 litros por pulso.

O fator de calibração foi calculado a partir da relação matemática entre o fluxo de saída de combustível no medidor de volume da bomba, em litros por revolução, e a quantidade total de pulsos gerados pelo sensor em uma volta completa, em pulsos por revolução. Segundo a documentação disponibilizada pela Exzotron Technology (2023), para o modelo FBCGQ-3 a relação é feita a partir da equação 4.1, no qual é determinado o fator de calibração de 0,005 litros por pulso elétrico. Isso significa que o eixo de rotação do sensor FBCGQ-3 precisa realizar duas voltas completas para contabilizar um litro de combustível que saiu da bomba.

$$Fator\ de\ calibração\ = 0.5\ [litros/revolução]\ x\ 100\ [pulsos/revolução] \quad (4.1)$$
$$Fator\ de\ calibração\ = 0.005\ [litros/pulso]$$

Em paralelo ao abastecimento, o controlador do sistema verificou constantemente se a quantidade de volume abastecido chegou no valor limite definido de 2 litros. Esta informação é encontrada no campo “limite_abastecimento” dentro do arquivo que contém a identificação de cada veículo cadastrado, conforme apresentado na Figura 4.10.

Assim que o abastecimento foi finalizado, o sistema acessou o módulo Ublox NEO-6M para registrar a latitude e longitude de onde foi realizado o mesmo. Os dados gerados no barramento podem ser vistos a partir da Figura 4.19.

Figura 4.19 - Dados retornados no barramento assim que o sistema acessa o módulo Ublox NEO-6M.

```
Location of the device:
Lat: -21.359653, Lng: -45.516589
Lat: -21.359653, Lng: -45.516589
Lat: -21.359653, Lng: -45.516589
Lat: -21.359653, Lng: -45.516589
Lat: -21.359653, Lng: -45.516589
Lat: -21.359654, Lng: -45.516591
```

Fonte: Autor (2023).

Sendo assim, a Figura 4.20 indica a estrutura de dados resultante de um processo de abastecimento, no qual as informações foram salvas linha por linha no arquivo *abastecimentos.txt* dentro do cartão de memória microSD.

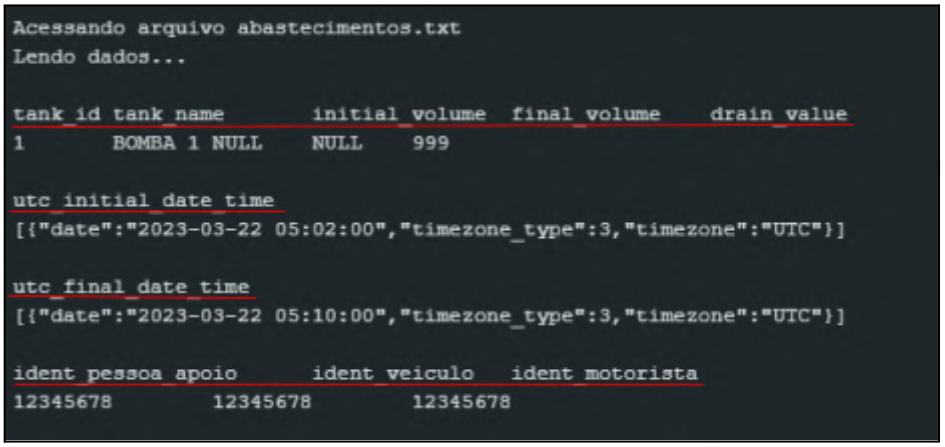
Figura 4.20 - Estrutura de dados formada após um processo de abastecimento.


```
[
  {
    "tank_id":1,
    "tank_name":"BOMBA 1",
    "initial_volume":"NULL",
    "final_volume":"NULL",
    "drain_value":999,
    "utc_initial_date_time":[{
      "date":"2023-03-22 00:02:00",
      "timezone_type":3,
      "timezone":"UTC"
    }],
    "utc_final_date_time":[{
      "date":"2023-03-22 00:10:00",
      "timezone_type":3,
      "timezone":"UTC"
    }],
    "ident_pessoa_apoio":12345678,
    "ident_veiculo":12345678,
    "ident_motorista":12345678
  }
]
```


Fonte: Autor (2023).

A Figura 4.21 apresenta um abastecimento já salvo dentro do cartão de memória microSD, no qual foi possível acessar suas informações pelo barramento SPI, onde o módulo microSD está conectado.

Figura 4.21 - Exemplo de um abastecimento salvo no arquivo *abastecimentos.txt,* o cabeçalho está sublinhado em vermelho.



Fonte: Autor (2023).

Durante o envio do abastecimento para o servidor com o banco de dados, o sistema realiza uma requisição na rota *postAbastecimentos*, passando os dados serializados como parâmetros da URL. Foram implementadas 5 rotas no projeto, sendo elas:

- *getMotoristas* consulta os motoristas cadastrados e seus identificadores. Seu *endpoint* é acessado por um método GET e não necessita de parâmetros adicionais.
  Método: GET
  URL: https:// .br/Motoristas

- *getVeiculos* consulta os veículos cadastrados e seus identificadores, assim como a volumetria liberada para o abastecimento do veículo. Seu *endpoint* é acessado por um método GET e não necessita de parâmetros adicionais.
  Método: GET
  URL: https:// .br/Veiculos

- *getFrentistas* consulta os frentistas cadastrados e seus identificadores. Seu *endpoint* é acessado por um método GET e não necessita de parâmetros adicionais.
  Método: GET
  URL: https:// .br/Frentistas

- *getAbastecimentos* consulta os abastecimentos cadastrados e seus identificadores. Seu *endpoint* é acessado por um método GET e necessita dos parâmetros dataInicial e dataFinal.
  Método: GET
  URL:https:// .br/Abastecimentos
  Parâmetros: dataInicial: string, dataFinal: string

- *postAbastecimentos* cadastra o(s) abastecimento(s) realizado(s) no banco de dados. Seu *endpoint* é acessado por um método POST e tem como parâmetros o(s) abastecimento(s) no formato JSON, conforme a Figura 4.22.
  Método: POST
  URL: https:// .br/Abastecimentos
  Parâmetros: abastecimentos: JSON string

Figura 4.22. Formato da URL que acessa a rota *postAbastecimentos*.

https:// .br/Abastecimentos?abastecimentos=[{[{"tank_id":1,"tank_name":"BOMBA_1","initial_volume":"NULL","final_volume":"NULL","drain_value":999,"utc_initial_date_time":[{"date":"2023-03-22%2000:02:00","timezone_type":3,"timezone":"UTC"}],utc_final_date_time":[{"date":"2023-03-22%2000:10:00","timezone_type":3,"timezone":"UTC"}],"ident_pessoa_apoio":12345678,"ident_veiculo":12345678,"ident_motorista":12345678}]

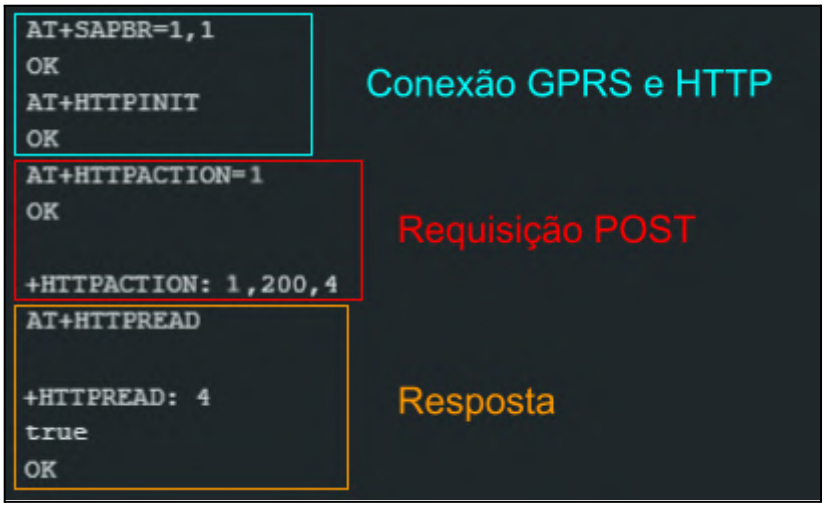
Fonte: Autor (2023).

A Figura 4.23 ilustra um exemplo de requisição HTTP realizada por meio do módulo SIM800L, no qual é enviado um abastecimento para o banco de dados por meio da rota *postAbastecimentos*. É possível visualizar a cadeia de comandos AT enviados do ESP32 para o módulo dentro do barramento UART.

Neste exemplo, o sistema abriu duas conexões, sendo elas uma GPRS (AT+SAPBR=1,1) e outra HTTP (AT+HTTPINIT), onde cada comando recebeu como resposta um “OK”, indicando que a solicitação foi executada com sucesso. Em seguida, a requisição HTTP foi realizada pelo comando AT+HTTPACTION=?, onde 0 indica ao módulo que será utilizado o método GET e 1 o método POST. O módulo SIM800L tem um *timeout* de 120 segundos até que ele aborte a requisição. Como a operação foi realizada com sucesso, o módulo respondeu com o método que foi utilizado (1 para POST), o status da requisição (200

como sucedida) e a quantidade de *bytes* da resposta, sendo neste caso um *true*, informando que o abastecimento foi salvo no banco de dados.

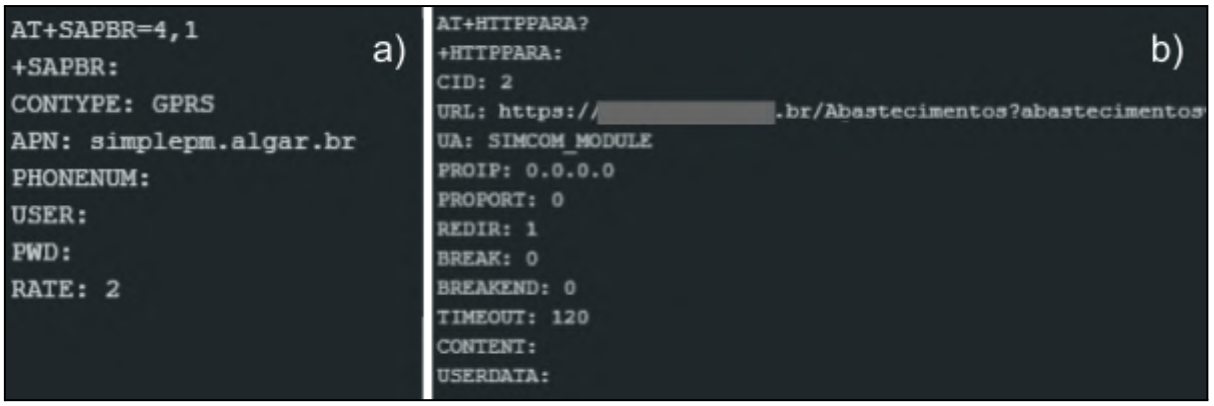
Figura 4.23 - Requisição realizada no sistema com a placa ESP32.



Fonte: Autor (2023).

A Figura 4.24 apresenta as configurações dos parâmetros das conexões GPRS e HTTP, nos quais podem ser facilmente configurados segundo os comandos AT da Tabela 4.1 (SIMCOM, 2015). A atenção deve ser voltada principalmente para a APN que garante o acesso do módulo na rede por meio do SIMCard, o tipo de conexão como GPRS e a URL da API REST.

Figura 4.24 - Parâmetros configurados para a conexão: a) GPRS e b) HTTP.



Fonte: Autor (2023).

Tabela 4.1 - Conjunto de comandos AT utilizados no módulo SIM800L.

Conexão	Comando	Descrição
GPRS	AT+SAPBR=3,1,"Contype","GPRS"	Define o tipo de conexão no módulo
	AT+SAPBR=3,1,APN, <APN>	Determina o nome do ponto de acesso (APN)
	AT+SAPBR=3,1,USER,""	Escreve o nome de usuário da APN
	AT+SAPBR=3,1,PWD,""	Escreve a senha da APN
	AT+SAPBR=1,1	Abre uma conexão GPRS
	AT+SAPBR=2,1	Status da conexão GPRS
	AT+SAPBR=4,1	Resgata os parâmetros da conexão GPRS
	AT+SAPBR=0,1	Finaliza uma conexão GPRS
HTTP	AT+HTTPINIT	Abre uma conexão HTTP
	AT+HTTPPARA="CID", 1	Parâmetros da configuração 1 utilizados
	AT+HTTPPARA="URL", <API>	Define a URL a ser chamada
	AT+HTTPSCONT	Salva as configurações definidas
	AT+HTTPACTION=0	Executa um método GET na URL definida
	AT+HTTPREAD	Recupera a resposta do método GET
	AT+HTTPTERM	Finaliza uma conexão HTTP

Fonte: Adaptado de SIMCom (2015).

Exemplificando com o teste, a Figura 4.25 demonstra um abastecimento que foi enviado ao banco de dados na nuvem utilizando os mesmos comandos AT e as configurações definidas para as conexões GPRS e HTTP.

Figura 4.25 - Representação JSON de um abastecimento de teste.

Fonte: Autor (2023).

É importante ressaltar que o módulo de comunicação SIM800L utiliza a rede móvel 2G e precisa receber os dados que serão enviados via barramento serial do ESP32. Logo, após vários testes, foi observado que o sistema demandava no máximo dois minutos desde o momento de armazenamento interno dos dados do abastecimento no cartão de memória até apresentá-los na aplicação web, para somente um abastecimento. Entretanto, os testes não foram realizados para o envio de dois ou mais abastecimentos ao mesmo tempo, o que poderia demandar mais tempo.

Após a requisição, o abastecimento de 2 litros foi visualizado pela aplicação web dentro do período de consulta definido, como pode ser observado na Figura 4.26.

Figura 4.26 - Utilizando a aplicação web para acessar o abastecimento realizado.

Fonte: Autor (2023).

Além disso, gerou-se os relatórios estatísticos e gráficos, indicados pela Figura 4.27. Neste ponto, a aplicação web tratou os dados dos abastecimentos para apresentar ao usuário dentro do período de 06 de julho até o dia 13 de julho (indicado no canto superior direito da interface), gerando assim as seguintes informações:

- *Card* Tanque 1: apresentou a entrada de combustível no tanque do posto interno (1000 litros, em verde) e a saída de combustível no período de consulta (2 litros, em vermelho), onde a diferença foi apresentada no campo Extrato (998 litros, em negrito).
- *Card* Tanque 2: ilustrou o segundo tanque instalado no posto interno, porém em ambiente de testes ele foi desconsiderado.

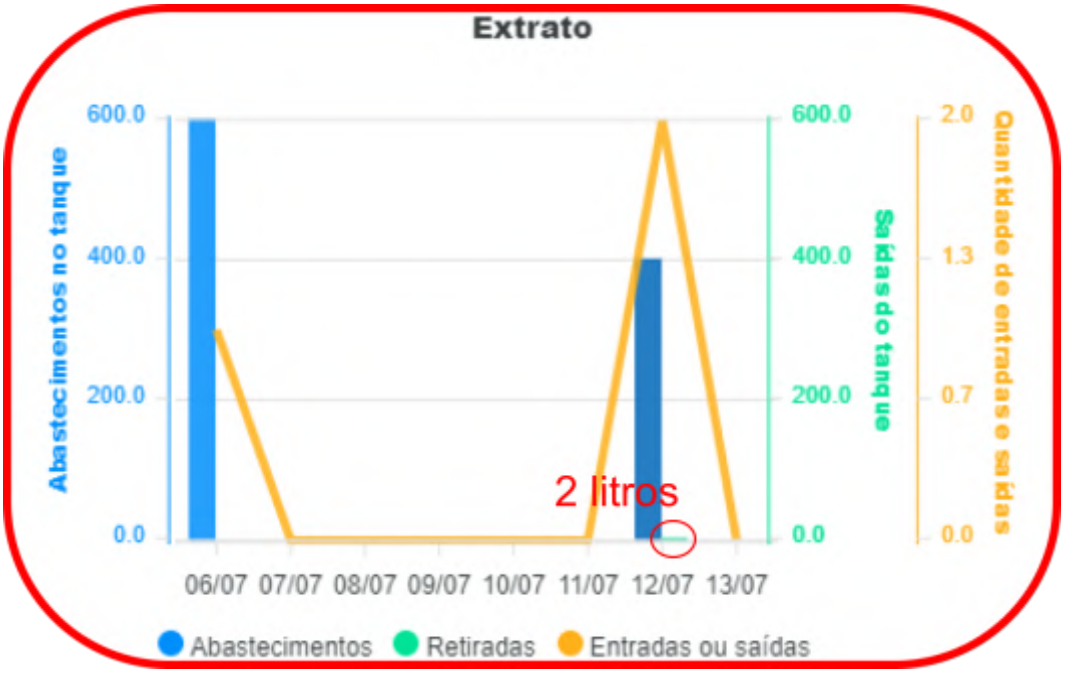
- *Card* Extrato: indicou as informações do Tanque 1, no qual foram realizadas duas entradas de combustível no tanque do posto interno (2 abastecimentos) e um abastecimento em um veículo (1 retirada). O gráfico indicado na Figura 4.28 ilustra a quantidade total de eventos de entradas e saídas de combustível na bomba, por dia (linha em amarelo), quantidade de entradas de combustível no Tanque 1, em litros (barra em azul) e a quantidade de combustível retirada, em litros (barra em verde).

Dessa forma, foi computado o abastecimento de 2 litros (pequena barra em verde no gráfico de Extrato na Figura 4.28), no dia 12 de julho de 2023 (indicado no eixo horizontal do gráfico), no mesmo dia em que foi inserido 400 litros de combustível dentro do Tanque 1 do posto interno (barra em azul).

Figura 4.27 - Utilizando a aplicação web para visualizar o extrato de combustível nas bombas.

Fonte: Autor (2023).

Figura 4.28 - Zoom no gráfico Extrato



Fonte: Autor (2023).

Por fim, a Figura 4.29 apresenta o abastecimento realizado, porém dentro do banco de dados. Dessa forma, o sistema garantiu que os dados serão persistidos e acessados por qualquer aplicação que consiga consumir as APIs REST implementadas, inclusive foi possível buscá-los novamente pelo ESP32.

Figura 4.29 - Resultado da requisição no lado do banco de dados. O abastecimento de 2 litros está indicado em vermelho.

id #	t... #	inicio	d... # ▲	latitude #	longitude #	frenti... #	motorist... #	veicul... #
8	3	2023-07-12T03:01:38.25…	2	-21.359653	-45.516589	1	1	1
2	1	2023-06-28T03:01:38.25…	200.3	310182	310182	2	2	3
4	1	2023-06-27T03:01:38.25…	760.2	1231241	1434113	1	2	3

Fonte: Autor (2023).

5 CONCLUSÃO

Neste capítulo será apresentada uma síntese de todo o conteúdo discutido, enfatizando quais objetivos foram alcançados com a pesquisa. Dessa forma, serão destacadas as contribuições desta pesquisa na comunidade, as implicações práticas encontradas e propostas de continuidade.

5.1 Contribuições

Não é recente que as empresas de logística tenham alocado seus investimentos para a instalação de postos internos de combustível. Com o avanço tecnológico, surgem muitas oportunidades para implementar soluções de automação que possam tornar os processos nos postos internos automáticos, eficientes e seguros. Dessa forma, o objetivo deste trabalho foi apresentar e desenvolver um protótipo com o ESP32 capaz de realizar esta função nas bombas de combustível, utilizando dos conceitos de IoT e telemetria. Como resultado, o sistema implementado foi capaz de coletar os dados gerados durante os abastecimentos, enviar para o serviço de banco de dados hospedado na nuvem e disponibilizá-los na aplicação web.

Seu desenvolvimento foi possível devido ao embasamento teórico pesquisado que garantiu conhecimento suficiente para elaborar o projeto de hardware com todos os sensores, atuadores e módulos de comunicação, além da implementação do software que instruiu o sistema a operar de forma adequada e esperada na maior parte dos casos de uso apresentados.

Os dispositivos utilizados no protótipo foram testados individualmente e em conjunto com o ESP32, verificando seu funcionamento. Logo, o ESP32, os sensores Dréxia e gerador de pulsos, assim como os atuadores *buzzer*, display LCD, LEDs e relé de bloqueio desempenharam efetivamente seus papéis no controle de acesso e dos abastecimentos na bomba de combustível. Os módulos de comunicação RTC DS1307 e cartão de memória microSD funcionaram corretamente, porém deve-se ficar atento aos seus contatos na protoboard. Os módulos SIM800L e ublox NEO-6M tiveram alguns problemas para receber sinal, porém já era um comportamento esperado pelo fato de utilizarem tecnologias de gerações antigas. Entretanto, pode-se concluir que os dispositivos performaram conforme dimensionado e esperado.

A metodologia ICONIX adotada se mostrou efetiva, pelo fato de que o projeto de software se tornou escalável e flexível. Isso significa que a metodologia garante uma fácil manutenção e melhorias que possam ser solicitadas por pessoas interessadas no projeto.

Como a implementação foi orientada a objetos, é possível implementar quaisquer interfaces e funcionalidades, pelo fato de que os artefatos da ferramenta, ou diagramas UML, permitem identificar em que pontos, ou entidades do software, precisam ser modificadas para atender a nova funcionalidade, tornando-se uma ferramenta poderosa para projetos em times e voltados a *feedbacks*.

Em suma, conclui-se que o projeto atingiu todos objetivos propostos para a pesquisa, com potencial de se tornar um complemento para futuras pesquisas na área de automação e telemetria.

5.2 Propostas de continuidade

É importante mencionar que os resultados desta pesquisa abrem muitas oportunidades de continuação e melhorias. Dessa forma, é possível apontar algumas sugestões:

- Montagem de uma placa de circuito impressa (PCB), pois é inevitável que o protótipo possa enfrentar problemas de conexões com *jumpers* e interferências em uma situação de instalação em campo. Sendo assim, a proposta seria montar a PCB e adicionar conectores de eixo, diminuindo a quantidade de *jumpers* e possíveis interferências geradas pelos fios, além de facilitar a conexão dos módulos externos e internos quando for necessário realizar uma manutenção;
- Caixa de proteção com material mais resistente. Ainda pensando nas aplicações em campo, torna-se necessário dimensionar uma caixa de proteção, ou *case*, que possa proteger os componentes eletrônicos do sistema e o ESP32. O material pode ser desde uma simples caixa de plástico até uma fibra de carbono, cabe ao instalador avaliar o grau de risco do local de instalação;
- Utilizar o módulo WIFI já embarcado no ESP32. Mesmo que o módulo SIM800L seja utilizado, é interessante ter uma outra opção de enlace de conexão na rede caso este fique sem sinal, como por exemplo o módulo WIFI. A sugestão é implementar e adicionar esta tecnologia como segunda opção de conexão na Internet.
- Testar os serviços de banco de dados, rotas das APIs REST e aplicação web em um serviço de hospedagem pago. Até o momento foram realizados testes em uma máquina local e em serviços gratuitos de hospedagem, o que não representa uma situação real de operação do projeto, pois o sistema pode perder dados caso o limite de armazenamento seja atingido;

- Aprimorar a aplicação web de forma que possa apresentar mais dados ao usuário. É possível configurar o ESP32 para enviar mais dados relacionados aos usuários frentistas, motoristas, veículos e status do sistema, como alimentação, tempo de uso, ligado ou desligado, entre outros. Logo, a aplicação web pode ser aprimorada para apresentar todos estes dados, além dos abastecimentos realizados. Dessa forma, o serviço entregará mais valor ao usuário;
- Melhorar o software desenvolvido para o ESP32, pois ainda existem alguns gargalos e possibilidades de otimização nas classes, principalmente nas classes MicroSD e GPRS que são mais complexas. Tal característica existe pois elas foram desenvolvidas acima das bibliotecas externas *ArduinoJson* e *TinyGsmClient*, apresentando assim uma dependência muito forte nelas;
- Implementar um componente na aplicação web que possa dar utilidade aos dados de latitude e longitude, como por exemplo chamar uma API de mapas do *Google Maps* ou do *Leaflet*.

REFERÊNCIAS BIBLIOGRÁFICAS

ABREU, B. E. **Aplicação de técnicas de segurança da informação para checagem de software de bombas medidoras de combustíveis líquidos na verificação subsequente.** Orientador: Dr Marcos José Hoffmann de Senna. 2017. 115 p. Dissertação (Mestrado) (Mestre em Metrologia e Qualidade) - Instituto Nacional de Metrologia, Qualidade e Tecnologia - Inmetro, Duque de Caxias, 2017. Disponível em: http://otzsrvbom.otimize.com:8080/jspui/bitstream/2050011876/1080/1/2017_Abreu.pdf. Acesso em: 10 ago. 2022.

AGGARWAL, S. **Modern web-development using ReactJS**. International Journal of Recent Research Aspects, Haryana, Índia, v. 5, p. 133 - 137, 2018. Disponível em: http://ijrra.net/Vol5issue1/IJRRA-05-01-27.pdf. Acesso em: 28 jun. 2023.

ALMEIDA, R. Comunicação Serial. In: _____ . **Programação de Sistemas Embarcados**: Desenvolvendo Software para Microcontroladores em Linguagem C. 1. ed. Rio de Janeiro: Grupo GEN, 2016. cap. 18, p. 262-272. ISBN 9788595156371. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788595156371/>. Acesso em: 2 ago. 2022.

ALMEIDA, R. Saídas digitais. In: _____. **Programação de Sistemas Embarcados**: Desenvolvendo Software para Microcontroladores em Linguagem C. 1. ed. Rio de Janeiro: Grupo GEN, 2016. cap. 14, p. 196-202. ISBN 9788595156371. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788595156371/>. Acesso em: 2 ago. 2022.

ALMEIDA, R. Saídas PWM. In: _____.. **Programação de Sistemas Embarcados**: Desenvolvendo Software para Microcontroladores em Linguagem C. 1. ed. Rio de Janeiro: Grupo GEN, 2016. cap. 20, p. 330-331. ISBN 9788595156371. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788595156371/>. Acesso em: 2 ago. 2022.

ANDRADE, P.; MARQUES, L.; DIAS, J.; DINIZ, M.; SIGNORETTI, G.; SILVA, I.; MELO, W. S.; GALHARDO, C. **Uma Solução Baseada na Internet dos Veículos Inteligentes para a Vigilância Metrológica de Bombas de Combustível**. Sociedade Brasileira de Automática: XV Simpósio Brasileiro de Automação Inteligente, Universidade Federal do Rio Grande do Norte, Natal, p. 2057-2059, 2021. DOI 10.20906/sbai.v1i1.2857. Disponível em: https://www.sba.org.br/open_journal_systems/index.php/sbai/article/view/2857. Acesso em: 15 mar. 2023.

ANP. **Anuário estatístico brasileiro do petróleo, gás natural e biocombustíveis**. 2023. Disponível em: https://www.gov.br/anp/pt-br/centrais-de-conteudo/publicacoes/anuario-estatistico/anuario-estatistico-2023. Acesso em: 18 jul. 2023.

ARDUINO DOCS. **Visão geral do IDE do Arduino**. Torino, 2023. Disponível em: https://docs.arduino.cc/software/ide-v1/tutorials/Environment. Acesso em: 3 jun. 2023.

BALACHANDRAN, S. **General Purpose Input/Output (GPIO)**. [s. l.], 11 ago. 2009. Disponível em: <https://www.egr.msu.edu/classes/ece480/capstone/fall09/group03/AN_balachandran.pdf>. Acesso em: 31 jul. 2022.

BARROS, E; CAVALCANTE, S. **Introdução aos Sistemas Embarcados**. Recife, p. 36, 2018. Disponível em: <https://www.cin.ufpe.br/~vba/periodos/8th/s.e/aulas/STP%20-%20Intro%20Sist%20Embarcados.pdf>. Acesso em: 31 jul. 2022.

BELSHE, M; PEON, R; THOMSON, M. **Hypertext Transfer Protocol Version 2**: HTTP/2. [S. l.], 2015. Disponível em: https://www.rfc-editor.org/rfc/rfc7540. Acesso em: 10 ago. 2022.

BETETO, A. L. **Proposta de automação para controle da vazão de combustíveis líquidos como nova abordagem de fiscalização para o gerenciamento de riscos em postos revendedores**. Orientador: Prof. Dr. Eduardo Mario Dias. 2019. 105 p. Dissertação (Mestrado) (Mestre em Sistemas de Potência) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Energia e Automação Elétricas, São Paulo, 2019. Disponível em: https://www.teses.usp.br/teses/disponiveis/3/3143/tde-19112019-095406/publico/AlinneLopomoBetetoCorr19.pdf. Acesso em: 10 ago. 2022.

CARVALHO, V. B. **Desenvolvimento e Teste de um Monitor de Barramento I2C para Proteção Contra Falhas Transientes**. Orientador: Profª. Drª. Fernanda G. L. Kastensmidt. 2016. 87 p. Dissertação (Mestrado em Ciências da Computação) - Universidade Federal do Rio Grande do Sul, Instituto de Informática, Porto Alegre, 2016. Disponível em: <https://www.lume.ufrgs.br/bitstream/handle/10183/150164/001008274.pdf?sequence=1>. Acesso em: 5 ago. 2022.

CHASE, O. **Sistemas Embarcados**. SBA Jovem, [s. l.], 2007. p. 2. Disponível em: <http://www.maxpezzin.com.br/aulas/6_EAC_Sistemas_Embarcados/1_SE_Introducao.pdf>. Acesso em: 27 jul. 2022.

CNT. **Como baixar a pressão no transporte de cargas**. 2019. Disponível em: https://cnt.org.br/agencia-cnt/cnt-transporte-cargas-preco-diesel. Acesso em: 18 jul 2023.

COMER, D. E. Conjuntos de Protocolos e Modelos em Camadas. In: _____ . **Redes de Computadores e Internet.** 6. ed. Porto Alegre: Bookman, 2016. cap. 1, p. 10-12. ISBN 978-0-13-358793-7. Disponível em: https://books.google.com.br/books?hl=pt-BR&lr=&id=1nwdDAAAQBAJ&oi=fnd&pg=PR1&dq=redes+de+computadores&ots=DskNH0nru7&sig=9XebJISru-aaS27-AYiZGftTjBo#v=onepage&q&f=false. Acesso em: 10 ago. 2022.

DNI. **Catálogo de produtos**: DNI0129. São Paulo: [s. n.], 2020. 204 p. Disponível em: <https://bd-sp.canaldapeca.com.br.s3.sa-east-1.amazonaws.com/DNI/Catalogo-DNI-30-anos.pdf>. Acesso em: 1 ago. 2022.

DREXIA. **1W-H3-05**: Leitor RFID de 125kHz para etiquetas exclusivas. [S. l.], 2022. Disponível em: <https://drexia.pl/en/strona-glowna/30-1w-h3-05-i-1w-h3-05k-125khz-rfid-reader-for-unique-tags.html>. Acesso em: 1 ago. 2022.

EFFEBI. Internet of Things' development. In: _____. **IoT Handbook**. Bovezzo: [s. n.], 2019. cap. 1, p. 2-3. ISBN 2016-1-IT01-KA202-005561. Disponível em: <https://www.iot4smes.eu/en/handbook_toc.aspx>. Acesso em: 10 ago. 2022.

ESPRESSIF SYSTEMS. **ESP32-DevKitC V4 getting started guide**. Shanghai: [s. n.], 2023. Disponível em: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html. Acesso em: 28 jun. 2023.

ESPRESSIF SYSTEMS. **ESP32-WROOM-32**. Shanghai: [s. n.], 2023. 27 p. v. 3.3. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf. Acesso em: 3 jun. 2023.

EXZOTRON TECHNOLOGY. **Gerador de pulsos**: FBCGQ-3. Rostov: [s. n.], 2018. 12 p. Disponível em: https://exzotron.ru/wp-content/uploads/2018/09/fbcgq-3-generator-impulsov-topaz-DSMK.402148.003-RE%60.pdf. Acesso em: 28 jun. 2023.

FAIRCHILD SEMICONDUCTOR. **BSS138**: N-Channel Logic Level Enhancement Mode Field Effect Transistor. Sunnyvale: [s. n.], 2005. 5 p. Disponível em: <http://cdn.sparkfun.com/datasheets/BreakoutBoards/BSS138.pdf>. Acesso em: 2 ago. 2022.

FERREIRA, A. G. Introdução aos APIs. In: ____. **Interface de programação de aplicações (API) e web services**. São Paulo: Platos Soluções Educacionais, 2021. cap. 2, p. 14-15. ISBN 978-65-5356-033-8. Disponível em: https://integrada.minhabiblioteca.com.br/reader/books/9786553560338/pageid/3. Acesso em: 10 ago. 2022.

FOROUZAN, B. A.; MOSHARRAF, F. Introdução: Protocolos em camadas. In: ____. **Redes de Computadores**: uma abordagem top-down. 1. ed. Porto Alegre, RS: Bookman, 2013. cap. 1, p. 8 - 21. ISBN 9788580551693. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788580551693/. Acesso em: 28 jun. 2023.

GABRIEL, M. M.; KURIA, K. P.. **Arduino Uno, Ultrasonic Sensor HC-SR04 Motion Detector with Display of Distance in the LCD**. International Journal of Engineering Research & Technology (IJERT), Quênia, v. 9, p. 938-939, 2020. Disponível em: <https://pdfs.semanticscholar.org/ed65/f45794d2223e83cf0037a807c6f0756ff373.pdf>. Acesso em: 2 ago. 2022.

GERIGAN, C; OGRUTAN, P. **AT commands in project based learning**. Bulletin of the Transilvania University of Brasov, Brasov, v. 4, ed. 2, p. 8, 2011. Disponível em:

http://webbut2.unitbv.ro/bu2011/Series%20I/BULETIN%20I/Gerigan_C.pdf. Acesso em: 3 ago. 2022.

GOOGLE. **Introduction to Google Drive API**. [S. l.]: Google Drive, 2022. Disponível em: https://developers.google.com/drive/api/guides/about-sdk?hl=pt_BR. Acesso em: 5 ago. 2022.

HEATH, S. What is an embedded system?. In: _____. **Embedded Systems Design**. 2. ed. London: Newnes, 2003. cap. 1, p. 1-12. ISBN 0 7506 5546 1.

IDEALI, W. Internet das Coisas: Conceito. In: _____. **Conectividade em Automação e IoT**: Protocolos I2C, SPI, USB, TCP-IP entre outros. Funcionalidade e interligação para automação e ToT. 1. ed. Rio de Janeiro, RJ: Alta Books, 2021. cap. 2, p. 27 - 37. ISBN 9786555202564. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9786555202564/. Acesso em: 28 jun. 2023.

JOSUTTIS, N. M. Introduction to C++ and the Standard Library. In: _____ . **The C++ Standard Library**. 2. ed. Nova Jersey: Addison-Wesley, 2013. cap. 2, p. 7-10. ISBN 978-0-321-62321-8. Disponível em: https://books.google.com.br/books?hl=pt-BR&lr=&id=9DEJKhasp7gC&oi=fnd&pg=PR3&dq=C%2B%2B&ots=_c07R6JyTF&sig=ynPBMWEsMyI9rzqTewIhuNlOkd4#v=onepage&q=C%2B%2B&f=false. Acesso em: 4 ago. 2022.

KAPLAN, E. D; HEGARTY, C. J. GPS Overview. In: ____ . **Understanding GPS**: Principles and Applications. 2. ed. Norwood: Artech House, 2005. cap. 1, p. 3-4. ISBN 9781580538947. Disponível em: https://d1.amobbs.com/bbs_upload782111/files_33/ourdev_ 584835O21W59.pdf. Acesso em: 3 ago. 2022.

MATTOS, A. N. Visão geral da telemetria. In: ____. **Telemetria e conceitos relacionados**: Uma visão geral dos sistemas de telemetria com ênfase em aplicações aeroespaciais. São José dos Campos, SP: [s. n.], 2004. cap. 2, p. 17 - 34. Disponível em: https://archive.org/details/TelemetriaEConceitosRelacionados. Acesso em: 28 jun. 2023.

MAXIM INTEGRATED. **1-Wire Communication Through Software**. Massachusetts, 2002. 15 p. Disponível em: <https://pdfserv.maximintegrated.com/en/an/AN126.pdf>. Acesso em: 1 ago. 2022.

MAXIM INTEGRATED. **DS3231**. 10. ed. rev. Sunnyvale: [s. n.], 2015. 20 p. Disponível em: <https://www.analog.com/media/en/technical-documentation/data-sheets/ds3231.pdf>. Acesso em: 2 ago. 2022.

META OPEN SOURCE. **Learn React**: Quick start. [S. l.], 2023. Disponível em: https://react.dev/learn. Acesso em: 28 jun. 2023.

MICROSOFT. **SQL Server no Azure**. [S. l.]: Microsoft, 2022. Disponível em: https://azure.microsoft.com/pt-br/services/sql-database/campaign/#pricing. Acesso em: 5 ago. 2022.

MICROSOFT. **Virtual Studio Code**: Getting started. [S. l.]: Microsoft, 2022. Disponível em: https://code.visualstudio.com/docs. Acesso em: 28 jun. 2022.

NASCIMENTO, F. P.; SOUSA, F. L. L. Classificação da Pesquisa: Natureza, método ou abordagem metodológica, objetivos e procedimentos. In: _____. **Metodologia da Pesquisa Científica**: Teoria e prática. Brasília: Thesaurus, 2015. cap. 6, ISBN 9788540903999.

NASUTION, T. H. et al. **Electrical appliances control prototype by using GSM module and Arduino**. 4th International Conference on Industrial Engineering and Applications, Nagoya, p. 1-2, 2017. Disponível em: <http://kresttechnology.com/krest-academic-projects/krest-mtech-projects/ECE/M%20Tech-ECE%20EMBEDDED%202017-18/MTECH%20IEEE%20ECE%20EMB%20BASE%20PAPER/43.%20Electrical%20appliances%20control%20prototype%20by%20using%20GSM%20module%20and%20Arduino.pdf>. Acesso em: 2 ago. 2022.

NETO, J. L. A. **O mercado brasileiro de combustíveis**. Boletim de Conjuntura do Setor Energético: FGV Energia, Rio de Janeiro, p. 4-5, 2017. Disponível em: https://bibliotecadigital.fgv.br/dspace/bitstream/handle/10438/19257/Coluna%20Opiniao%20Fevereiro%20Jose%20Lima.pdf. Acesso em: 15 mar. 2023.

OLIVEIRA, A. S.; ANDRADE, F. S. Firmware. In: ____. **Sistemas Embarcados**: Hardware e Firmware na Prática. 2. ed. São Paulo: Saraiva, 2010. cap. 2, p. 88-103. ISBN 9788536520346.

OLIVEIRA, A. S.; ANDRADE, F. S. Hardware: A parte física de um sistema embarcado. In: .____. **Sistemas Embarcados**: Hardware e Firmware na Prática. 2. ed. São Paulo: Saraiva, 2010. cap. 1, p. 21-23, 47-48. ISBN 978 85 365 2034 6.

OLIVEIRA, G. C.; HAHN, J. S.; SILVA, E. S.; SILVA, P. A. **Desenvolvimento e Implantação de um sistema supervisório de energia elétrica no IFSC-SJ**. 4º Seminário de Pesquisa, Extensão e Inovação do IFSC, Florianópolis, p. 2, 2014. Disponível em: https://www.researchgate.net/profile/Tiago-Semprebom/publication/281592374_Desenvolvimento_e_Implantacao_de_um_sistema_supervisorio_de_energia_eletrica_no_IFSC-SJ/links/55ef304a08ae0af8ee1b136e/Desenvolvimento-e-Implantacao-de-um-sistema-supervisorio-de-energia-eletrica-no-IFSC-SJ.pdf. Acesso em: 10 ago. 2022.

OLIVEIRA, S. **Internet das Coisas com ESP8266, Arduino e Raspberry PI**. São Paulo: Novatec, 2017. 257 p. ISBN 978-85-7522-582-0.

PEREIRA, F. Protocolo 1-wire. In: ____. **Microcontroladores PIC**: Programação em C. 7. ed. São Paulo: Saraiva, 2009. cap. 12, p. 283-289. ISBN 9788536519937. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788536519937/>. Acesso em: 1 ago. 2022.

PEREIRA, F. Tópicos Avançados: Apresentação em Display. In: ____. **Microcontroladores PIC**: Programação em C. 7. ed. São Paulo: Saraiva, 2009. cap. 12, p. 310 - 313. ISBN 9788536519937. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788536519937/. Acesso em: 11 jul. 2023.

POTTER, J. **Npm trends**. [S. l.], 2023. Disponível em: https://npmtrends.com/angular-vs-react-vs-vue. Acesso em: 28 jun. 2023.

ROCHA, R. **Programação Paralela e Distribuída**: Fundamentos. Faculdade de Ciências da Universidade do Porto, Porto, p. 1-12; 53-54, 2008. Disponível em:

https://www.dcc.fc.up.pt/~ricroc/aulas/0708/ppd/apontamentos/fundamentos.pdf. Acesso em: 5 ago. 2022.

ROCHOL, J. Redes celulares: GPRS. In: ____. **Sistemas de comunicação sem fio**: conceitos e aplicações. Porto Alegre, RS: Bookman, 2018. cap. 8, p. 365 - 367. ISBN 9788582604564. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/ 9788582604564. Acesso em: 28 jun. 2023.

RODRIGUES, T. N. et al. Geração de serviços com REST: Introdução à arquitetura REST. In: .____. **Integração de aplicações**. Porto Alegre, RS: Sagah, 2020. cap. 9, p. 151 - 155. ISBN 9786556900216. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/ 9786556900216/. Acesso em: 28 jun. 2023.

ROSENBERG, D.; STEPHENS, M. What is Agility? (And why does it matter?). In: ____. **Agile Development with ICONIX Process**: people, process, and pragmatism. 1. ed. New York: Apress, 2005. cap. 1, p. 21 - 23. ISBN 9781590594643. Disponível em: https://books.google.com.br/books?id=pjGzTOYocikC&pg=PA1&hl=pt-BR&source=gbs_toc_r&cad=4#v=onepage&q&f=false. Acesso em: 28 jun. 2023.

SANTOS, F. T.; MARINATO, M. M.; SANTOS, W. C. **Utilização de um sistema Supervisório na usina fotovoltaica**. Rede de Ensino DOCTUM, Espírito Santo, p. 8-10, 2021. Disponível em: https://dspace.doctum.edu.br/bitstream/123456789/3885/1/Sistema%20Supervis%C3%B3rio%20_Artigo_2021_2.pdf. Acesso em: 10 ago. 2022.

SHYMANSKYY, V. **TinyGSM**: A small Arduino library for GSM modules, that just works. GitHub, v 0.11.7, 2023. Disponível em: https://github.com/vshymanskyy/TinyGSM/wiki/Powering-GSM-module. Acesso em: 8 jul. 2023.

SIMCOM. **Module 2G**. Shanghai, 2022. Disponível em: https://www.smartinstec.com.br/blog/sabe-como-funciona-o-sistema-gsm-e-gprs/. Acesso em: 3 ago. 2022.

.

SIMCOM. **SIM800 Series AT Command Manual**. Shanghai: [s. n.], 2015. 380 p. v. 1.09. Disponível em: https://www.elecrow.com/wiki/images/2/20/SIM800_Series_AT_Command_Manual_V1.09.pdf. Acesso em: 3 ago. 2022.

SONI, A.; RANGA, V. **API Features Individualizing of Web Services**: REST and SOAP. International Journal of Innovative Technology and Exploring Engineering (IJITEE), Madhya Pradesh, v. 8, p. 666, 2019. Disponível em: https://www.researchgate.net/profile/Virender-Ranga/publication/335419384_API_Features_Individualizing_of_Web_Services_REST_and_SOAP/links/5d64960ea6fdccc32cd31171/API-Features-Individualizing-of-Web-Services-REST-and-SOAP.pdf. Acesso em: 10 ago. 2022.

SPARKFUN ELETRONICS. **Bi-Directional Logic Level Converter Hookup Guide**. Niwot, 2022. Disponível em: <https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide#>. Acesso em: 2 ago. 2022.

SPARKFUN ELECTRONICS. **Sensores e Atuadores IOT**. São Paulo, 2020. Disponível em: https://embarcados.com.br/sensores-e-atuadores-iot/. Acesso em: 01 ago. 2022.

TANENBAUM, A. S.; BOS, H. Processos e Threads. In: _____. **Sistemas Operacionais Modernos**. 4. ed. São Paulo: Pearson, 2016. cap. 2, p. 83-84. ISBN 978-85-4301-818-8. Disponível em: http://ldemetrio.com.br/Livros/Livros_TI/segunda_unid/Sistemas%20Operacionais%20Modernos%20-%20Tanenbaum%20-%204%20Edi%C3%A7%C3%A3o.pdf. Acesso em: 5 ago. 2022.

TAURION, C. O que é Computação em Nuvem. In: _____. **Computação em Nuvem**: Transformando o mundo da Tecnologia da Informação. Rio de Janeiro: Brasport, 2009. cap. 1, p. 17-26. ISBN 978-85-7452-423-8. Disponível em: https://books.google.com.br/books?hl=pt-BR&lr=&id=mvir2X-A2mcC&oi=fnd&pg=PA1&dq=computa%C3%A7%C3%A3o+em+nuvem&ots=CbKj1CTVYt&sig=Q9dXWkRW6y4DuUl4B8MZeYypuwQ#v=onepage&q=computa%C3%A7%C3%A3o%20em%20nuvem&f=false. Acesso em: 5 ago. 2022.

TEIXEIRA, G. R. **Desenvolvimento de sistema microcontrolado para redução de perdas associadas ao processo de refrigeração do leite em pequenas propriedades**. Orientador: Prof. Dr. Gustavo Lobato Campos. 2018. 69 p. Monografia (Bacharel em Engenharia Elétrica) - Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais, [S. l.], 2018.

Disponível em: https://www.formiga.ifmg.edu.br/documents/2018/Biblioteca/TCCs_e_Artigos/TCC_EE_Gabriel_Ramos_Teixeira.pdf. Acesso em: 3 ago. 2022.

THOMAZINI, D.; ALBUQUERQUE, P. U. B. Introdução. In: _____. **Sensores industriais**: Fundamentos e aplicações. 9. ed. São Paulo: Érica, 2020. cap. 1, p. 1-2. ISBN 9788536533230. Disponível em: <https://books.google.com.br/books?hl=pt-BR&lr=&id=1qgPEAAAQBAJ&oi=fnd&pg=PP15&dq=sensores&ots=RJ7D_KnLWC&sig=PnrM7ow5EkjJZkXIvIau7tS-_FI#v=onepage&q=sensores&f=false>. Acesso em: 1 ago. 2022.

TRINDADE, Derick Horrana de Souza. **Monitoramento de sistemas de transporte e de rodovias com Arduino e Shield**: GSM, GPS e GPRS. Orientador: Marcus Vinicius Batistuta. 2015. 120 p. Monografia (Bacharelado em Engenharia Eletrônica) - Faculdade UnB Gama, Universidade de Brasília, [S. l.], 2014. Disponível em: https://bdm.unb.br/bitstream/10483/20837/1/2015_DerickHorranaDeSouzaDaTrindade_tcc.pdf. Acesso em: 3 ago. 2022.

WAYNE FUELING SYSTEM. **Marcas**. [S. l.], 2023. Disponível em: https://www.dover fuelingsolutions.com/wayne?lang=pt. Acesso em: 28 jun. 2023.

WENDLING, M. **Sensores**. UNESP - Campus de Guaratinguetá, n. 2, p. 4-5, 2010. Disponível em: https://www.feg.unesp.br/Home/PaginasPessoais/ProfMarceloWendling/4---sensores-v2.0.pdf. Acesso em: 27 jul. 2022.

WESTERN DIGITAL. **Sandisk**. [S. l.], 2022. Disponível em: https://www.westerndigital.com/pt-br/brand/sandisk. Acesso em: 2 ago. 2022.

WILMSHURST, T. Tiny computers, hidden control: What is an embedded system?. In: ____. **Designing embedded systems with PIC microcontrollers**: Principles and applications. 1. ed. Londres: Elsevier, 2007. cap. 1, p. 3-13. ISBN 0 7506 6755 9.

XIAMEN AMOTEC DISPLAY. **Specifications of LCD module**. Xiamen, China: [s. n.], 2008. 22 p. Disponível em: https://components101.com/sites/default/files/component datasheet/16x2%20LCD%20Datasheet.pdf. Acesso em: 11 jul. 2023.

XLSEMI. **XL4005**: 5A 300kHz 32V Buck DC to DC Converter. 2.1. ed. rev. [S. l.: s. n.], 2020. 9 p. Disponível em: https://pdf1.alldatasheet.com/datasheet-pdf/view/763181/ETC2/XL4005.html. Acesso em: 2 ago. 2022.

APÊNDICES

A.1 - Arquivo .cpp da Classe Identificacao

```C/C++
#include "Identificacao.h"
Identificacao::Identificacao(){
  _id_frentista = 0;
  _id_veiculo = 0;
  _id_motorista = 0;
  _id_usuario = "frentista";
}

int Identificacao::getIdFrentista(void){return _id_frentista;}
void Identificacao::setIdFrentista(int id){_id_frentista = id;}
int Identificacao::getIdVeiculo(void){return _id_veiculo;}
void Identificacao::setIdVeiculo(int id){_id_veiculo = id;}
int Identificacao::getIdMotorista(void){return _id_motorista;}
void Identificacao::setIdMotorista(int id){_id_motorista = id;}
String Identificacao::getIdUsuario(void){return _id_usuario;}
void Identificacao::setIdUsuario(String nome){_id_usuario = nome;}
```

A.2 - Arquivo .cpp da Classe Abastecimento

```
C/C++
#include "Abastecimento.h"

Abastecimento::Abastecimento(){}
int Abastecimento::getIdTanque(void){return _id_tanque;}
void Abastecimento::setIdTanque(int id){_id_tanque = id;}
String Abastecimento::getNomeTanque(void){return _nome_tanque;}
void Abastecimento::setNomeTanque(String nome){_nome_tanque = nome;}
float Abastecimento::getVolumeSaida(void){return _volume_saida;}
void Abastecimento::setVolumeSaida(float volume){_volume_saida = volume;}
Localizacao Abastecimento::getLocalizacaoBomba(void){return _localizacao;}
void Abastecimento::setLocalizacaoBomba(float latitude, float longitude){
  _localizacao.setLatitude(latitude);
  _localizacao.setLongitude(longitude);}
Data Abastecimento::getDataInicial(void){return _inicio_abastecimento;}
void Abastecimento::setDataInicial(String data_inicial){
  _inicio_abastecimento.setData(data_inicial);}
Data Abastecimento::getDataFinal(void){return _fim_abastecimento;}
void Abastecimento::setDataFinal(String data_final){
  _fim_abastecimento.setData(data_final);}
Identificacao Abastecimento::getUsuarios(void){return _usuarios;}
void   Abastecimento::setUsuarios(int   id_frentista,   int   id_veiculo,   int
id_motorista){
  _usuarios.setIdFrentista(id_frentista);
  _usuarios.setIdMotorista(id_motorista);
  _usuarios.setIdVeiculo(id_veiculo);
}
```

A.3 - Arquivo .cpp da Classe Data

```
C/C++
#include "Data.h"

Data::Data(){
  _timezone_type = "UTC";
  _timezone = 3;
  _data = "2000-01-01T00:00:00.001Z";}
void Data::setData(String data){_data = data;}
String Data::getData(void){return _data;}
void Data::setTimezoneType(String type){_timezone_type = type;}
String Data::getTimezoneType(void){return _timezone_type;}
void Data::setTimezone(int timezone){_timezone = timezone;}
int Data::getTimezone(void){return _timezone;}
```

A.4 - Arquivo .cpp da Classe Localizacao

```
C/C++
#include "Localizacao.h"

Localizacao::Localizacao(void){
  _latitude = 0;
  _longitude = 0;}
void Localizacao::setLatitude(float latitude){_latitude = latitude;}
float Localizacao::getLatitude(void){return _latitude;}
void Localizacao::setLongitude(float longitude){_longitude = longitude;}
float Localizacao::getLongitude(void){return _longitude;}
```

A.5 - Arquivo .cpp da Classe Drexia

```
C/C++
#include "DrexiaServices.h"
#include <OneWire.h>

DrexiaServices::DrexiaServices(int pino_one_wire)
: _pino_one_wire(pino_one_wire), _barramento_drexia(_pino_one_wire){
  _id_do_cartao = 0;
  setupOneWire();}

void DrexiaServices::setupOneWire(void){}
int DrexiaServices::getIdDoCartao(void){return _id_do_cartao;}
void DrexiaServices::setIdDoCartao(int id){_id_do_cartao = id;}

bool DrexiaServices::getCodigoDoCartao32bits(byte _buffer_1wire[]){
  bool crc_valido = true;
  _id_do_cartao = (_id_do_cartao + _buffer_1wire[4]) << 8;
  _id_do_cartao = (_id_do_cartao + _buffer_1wire[3]) << 8;
  _id_do_cartao = (_id_do_cartao + _buffer_1wire[2]) << 8;
  _id_do_cartao = (_id_do_cartao + _buffer_1wire[1]);
  if (OneWire::crc8(_buffer_1wire, 7) != _buffer_1wire[7]){
    Serial.print("CRC não é válido!\n");
    crc_valido = false;}
  return crc_valido;}

void DrexiaServices::getIdDoCartao64bits(void){
  bool codigo_resgatado_com_sucesso = false;
  byte _buffer_1wire[8];
  while (!codigo_resgatado_com_sucesso){
    while (_barramento_drexia.search(_buffer_1wire)){
      codigo_resgatado_com_sucesso = getCodigoDoCartao32bits(_buffer_1wire);
      if (!codigo_resgatado_com_sucesso){return;}
    }
    _barramento_drexia.reset_search();}}
```

A.6 - Arquivo .cpp da Classe Pulser

```
C/C++
#include "PulserServices.h"

PulserServices::PulserServices(){
  _limite_de_abastecimento = 0;
  _fator_de_calibracao = 1;
  _limiar_de_tensao = 400;
  setupPulser();}

PulserServices::PulserServices(float      limite_de_abastecimento,      float
calibracao, int limiar_de_tensao = 400){
  _limite_de_abastecimento = limite_de_abastecimento;
  _fator_de_calibracao = fator_de_calibracao;
  _limiar_de_tensao = limiar_de_tensao;
  setupPulser();}

void PulserServices::setupPulser(void){}
void PulserServices::setLimiteAbastecimento(float limite_de_abastecimento){
  _limite_de_abastecimento = limite_de_abastecimento;}
float PulserServices::getLimiteAbastecimento(void){
  return _limite_de_abastecimento;}
void PulserServices::setCalibracao(float fator_de_calibracao){
  _fator_de_calibracao = fator_de_calibracao;}
float PulserServices::getCalibracao(void){return _fator_de_calibracao;}
void PulserServices::setLimiarTensao(int limiar_de_tensao){
  _limiar_de_tensao = limiar_de_tensao;}
int PulserServices::getLimiarTensao(void){return _limiar_de_tensao;}
void PulserServices::setExtrato(float extrato){_extrato = extrato;}
float PulserServices::getExtrato(void){return _extrato;}
int PulserServices::receberPulsosDoGatilhoZero(void){
  int value = analogRead(_pino_sensor_0);
  return value;}
int PulserServices::receberPulsosDoGatilhoUm(void){
  int value = analogRead(_pino_sensor_1);
  return value;}
```

```
bool PulserServices::handleContagemPulsos(void){
  bool finalizado = false;
  bool flag = false;
  float extrato = 0;
  int sensorValue0 = 0;
  int sensorValue1 = 0;
  int contagem = 0;

  if (_limite_de_abastecimento == 0){return finalizado;}
  while (extrato < _limite_de_abastecimento){
    sensorValue0 = receberPulsosDoGatilhoZero();
    sensorValue1 = receberPulsosDoGatilhoUm();
   if ((sensorValue0 > _limiar_de_tensao or sensorValue1 > _limiar_de_tensao)
and flag == false){
      contagem++;
      extrato = (float)contagem * _fator_de_calibracao;
      Serial.println(extrato);
      flag = true;}
   if ((sensorValue0 == 0 or sensorValue1 == 0) and flag == true){
      flag = false;}}
  finalizado = true;
  setExtrato(extrato);
  return finalizado;}
```

A.7 - Arquivo .cpp da Classe Atuadores

```
C/C++
#include "AtuadorServices.h"
#include <LiquidCrystal_I2C.h>

AtuadorServices::AtuadorServices(int  bomba_gatilho_0,  int  bomba_gatilho_1,
int  pino_sensor_0,  int pino_sensor_1,  int  pino_buzzer,  int  pino_led_verde,
int pino_led_vermelho, int pino_led_azul){
  _bomba_gatilho_0 = bomba_gatilho_0;
  _bomba_gatilho_1 = bomba_gatilho_1;
  _pino_sensor_0 = pino_sensor_0;
  _pino_sensor_1 = pino_sensor_1;
  _pino_buzzer = pino_buzzer;
  _pino_led_verde = pino_led_verde;
  _pino_led_vermelho = pino_led_vermelho;
  _pino_led_azul = pino_led_azul;
  setupAtuadores();}

void AtuadorServices::setupAtuadores(void){
  pinMode(_bomba_gatilho_0, OUTPUT);
  pinMode(_bomba_gatilho_1, OUTPUT);
  pinMode(_pino_buzzer, OUTPUT);
  pinMode(_pino_sensor_0, INPUT_PULLDOWN);
  pinMode(_pino_sensor_1, INPUT_PULLDOWN);
  pinMode(_pino_led_verde, OUTPUT);
  pinMode(_pino_led_vermelho, OUTPUT);
  pinMode(_pino_led_azul, OUTPUT);}
void AtuadorServices::atuarNoReleDeBloqueioZero(bool valor){
  digitalWrite(_bomba_gatilho_0, valor);}
void AtuadorServices::atuarNoReleDeBloqueioUm(bool valor){
  digitalWrite(_bomba_gatilho_1, valor);}
void AtuadorServices::atuarNoLedVermelho(bool valor){
  digitalWrite(_pino_led_vermelho, valor);}
void AtuadorServices::atuarNoLedAzul(bool valor){
  digitalWrite(_pino_led_azul, valor);}
void AtuadorServices::atuarNoLedVerde(bool valor){
```

```
  digitalWrite(_pino_led_verde, valor);}
void AtuadorServices::atuarNoBuzzer(int tempo_delay){
  digitalWrite(_pino_buzzer, HIGH);
  delay(tempo_delay);
  digitalWrite(_pino_buzzer, LOW);
  delay(tempo_delay);
  digitalWrite(_pino_buzzer, HIGH);
  delay(tempo_delay);
  digitalWrite(_pino_buzzer, LOW);}

void AtuadorServices::atuarNoDisplay(String mensagem){
  int lcdColumns = 16;
  int lcdRows = 2;
  LiquidCrystal_I2C lcd(0x27, lcdColumns, lcdRows);
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print(mensagem);}

void AtuadorServices::reiniciarEsp32(void){ESP.restart();}
```

A.8 - Arquivo .cpp da Classe GPS

```
C/C++
#include "GpsServices.h"
#include <TinyGPSPlus.h>

GpsServices::GpsServices(void){Serial2.begin(9600);}
Localizacao GpsServices::getLocalizacao(void){return _localizacao;}

void GpsServices::getInfoGps(void){
  float latitude = (float)(_gps.location.lat(), 6);
  float longitude = (float)(_gps.location.lng(), 6);
  _localizacao.setLatitude(latitude);
  _localizacao.setLongitude(longitude);}

void GpsServices::handleLatitudeLongitude(void){
  bool dados_verificados = false;
  bool dados_da_porta_serial_resgatados = false;
  bool dados_de_localizacao_validos = false;

  while (dados_verificados == false){
    dados_da_porta_serial_resgatados = _gps.encode(Serial2.read());
    if (dados_da_porta_serial_resgatados){
      dados_de_localizacao_validos = _gps.location.isValid();
      if (dados_de_localizacao_validos){
        getInfoGps();
        dados_verificados = true;}
      else{Serial.print(F("INVALIDO"));}}}
}
```

A.9 - Arquivo .cpp da Classe GPRS

```
C/C++
#include "GprsServices.h"
#include <TinyGsmClient.h>
#include <ArduinoHttpClient.h>

GprsServices::GprsServices(String apn, String gprs_usuario, String
gprs_senha, String servidor, String recurso){
  _apn = apn;
  _gprs_usuario = gprs_usuario;
  _gprs_senha = gprs_senha;
  _servidor = servidor;
  _recurso = recurso;
  _body = "";
  setupGprs();}

void GprsServices::setupGprs(void){
#define SerialAT Serial2
#define TINY_GSM_RX_BUFFER 10000
#define GSM_BAUD 9600
#define TINY_GSM_USE_GPRS true
  TinyGsmClient client(modem);
  const int port = 80;
  HttpClient http(client, server, port);
  delay(10);
  SerialAT.begin(9600);
  delay(6000);}

void GprsServices::inicializarGprs(void){
  modem.restart();
  String modemInfo = modem.getModemInfo();
#if TINY_GSM_USE_GPRS
  if (GSM_PIN && modem.getSimStatus() != 3){modem.simUnlock(GSM_PIN);}
#endif}
```

```
bool GprsServices::conectarNaRede(void){
  if (!modem.waitForNetwork()){
    delay(10000);
    return false;}
  if (modem.isNetworkConnected()){Serial.println("Conectado na rede");}
#if TINY_GSM_USE_GPRS
  if (!modem.gprsConnect(_apn, _gprs_usuario, _gprs_senha)){
    delay(10000);
    return false;}
  if (modem.isGprsConnected()){Serial.println("Conexão GPRS sucedida");}
#endif
  return true;}

bool GprsServices::requisicaoHttp(String token, String abastecimentosJson){
  http.connectionKeepAlive();
  int err = http.get(resource);
  if (err != 0){
    delay(10000);
    return;}
  int status = http.responseStatusCode();
  if (!status){
    delay(10000);
    return;}
  setBody(http.responseBody());
  http.stop();
  return true;}

bool GprsServices::desconectarGprs(void){
#if TINY_GSM_USE_GPRS
  modem.gprsDisconnect();
  setBody("");
#endif
  return true;}

String GprsServices::getBody(void){return _body;}
void GprsServices::setBody(String body){_body = body;}
String GprsServices::getUsuario(void){return _usuario;}
```

```
void GprsServices::setUsuario(String usuario){_usuario = usuario;}
String GprsServices::getSenha(void){return _senha;}
void GprsServices::setSenha(String senha){_senha = senha;}
String GprsServices::getServidor(void){return _servidor;}
void GprsServices::setServidor(String servidor){_servidor = servidor;}
String GprsServices::getApn(void){return _apn;}
void GprsServices::setApn(String apn){_apn = apn;}
String GprsServices::getRecurso(void){return _recurso;}
void GprsServices::setRecurso(String recurso){_recurso = recurso;}
```

A.10 - Arquivo .cpp da Classe MicroSD

```
C/C++
#include "MicroSdServices.h"
#include <ArduinoJson.h>
#include <SD.h>
#include <SPI.h>
#include "Data.h"

MicroSdServices::MicroSdServices(int pino_sd, String filename_abastecimentos,
String     filename_frentistas,     String     filename_veiculos,     String
filename_motoristas){
  _file_abastecimentos = filename_abastecimentos;
  _file_frentistas = filename_frentistas;
  _file_motoristas = filename_motoristas;
  _file_veiculos = filename_veiculos;
  _pino_moduloSD = pino_sd;
  _quantidade_abastecimentos = 0;
  inicializarMicroSd();}

void MicroSdServices::inicializarMicroSd(void){
  if (!SD.begin(_pino_moduloSD)){return;}
  uint8_t cardType = SD.cardType();
  if (cardType == CARD_NONE){return;}
  uint64_t cardSize = SD.cardSize() / (1024 * 1024);
  Serial.printf("SD Card Size: %lluMB\n", cardSize);}

String MicroSdServices::acessarAbastecimentos(void){
  String abastecimentos = "";
  if (SD.exists(_file_abastecimentos)){
    File file = SD.open(_file_abastecimentos);
    if (!file){Serial.println(F("Falha ao ler arquivo"));}
    else{
      while (file.available()){
        abastecimentos = file.readString();
        file.close();}
      Serial.println(abastecimentos);}}
```

```
  else{Serial.println("O arquivo não existe!");}
  return abastecimentos;}

String MicroSdServices::acessarFrentistas(void){
  String frentistas = "";
  if (SD.exists(_file_frentistas)){
    File file = SD.open(_file_frentistas);
    if (!file){Serial.println(F("Falha ao ler arquivo"));}
    else{
      while (file.available()){
        frentistas = file.readString();
        file.close();}
      Serial.println(frentistas);}}
  else{Serial.println("O arquivo não existe!");}
  return frentistas;}

String MicroSdServices::acessarMotoristas(void){
  String motoristas = "";
  if (SD.exists(_file_motoristas)){
    File file = SD.open(_file_motoristas);
    if (!file){Serial.println(F("Falha ao ler arquivo"));}
    else{
      while (file.available()){
        motoristas = file.readString();
        file.close();}
      Serial.println(motoristas);}}
  else{Serial.println("O arquivo não existe!");}
  return motoristas;}

String MicroSdServices::acessarVeiculos(void){
  String veiculos = "";
  if (SD.exists(_file_veiculos)){
    File file = SD.open(_file_veiculos);
    if (!file){Serial.println(F("Falha ao ler arquivo"));}
    else{
      while (file.available()){
        veiculos = file.readString();
```

```
        file.close();}
      Serial.println(veiculos);}}
  else{Serial.println("O arquivo não existe!");}
  return veiculos;}

void MicroSdServices::salvarAbastecimento(String abastecimento_serial)
{
  File file = SD.open(_file_abastecimentos);
  StaticJsonDocument<(_quantidade_abastecimentos * 800) + 2> root;
  DeserializationError error = deserializeJson(root, file);
  if (error){Serial.println(F("Nenhum abastecimento inicial no arquivo"));}
  file.close();
  StaticJsonDocument<abastecimento_serial.length()> doc;
  DeserializationError error = deserializeJson(doc, abastecimento_serial);
  if (error){
    Serial.print("Falha ao deserializar o JSON: ");
    Serial.println(error.c_str());
    return false;}

  JsonObject obj = root.createNestedObject();
  obj["tank_id"] = doc["id"];
  obj["tank_name"] = doc["nome_bomba"];
  obj["latitide"] = doc["latitude"];
  obj["longitude"] = doc["longitude"];
  obj["drain_value"] = doc["volume_abastecido"];

  JsonObject utc_initial_date_time =
      obj.createNestedObject("utc_initial_date_time");
  utc_initial_date_time["date"] = doc["data_inicial"]["dia"];
  utc_initial_date_time["timezone_type"] =
      doc["data_inicial"]["timezone_tipo"];
  utc_initial_date_time["timezone"] = doc["data_inicial"]["timezone"];
  JsonObject utc_final_date_time =
      obj.createNestedObject("utc_final_date_time");
  utc_final_date_time["date"] = doc["data_final"]["dia"];
  utc_final_date_time["timezone_type"] = doc["data_final"]["timezone_tipo"];
  utc_final_date_time["timezone"] = doc["data_final"]["timezone"];
```

```
  obj["ident_pessoa_apoio"] = doc["id_frentista"];
  obj["ident_veiculo"] = doc["id_veiculo"];
  obj["ident_motorista"] = doc["id_motorista"];

  file = SD.open(_file_abastecimentos, FILE_WRITE);
  if (serializeJson(root, file) == 0){
    Serial.println(F("Falha ao escrever no arquivo"));
    return false;}
  file.close();
  return true;}

float MicroSdServices::buscarLimiteAbastecimento(int id_veiculo){}
```

A.11 - Arquivo .cpp da Classe RTC

```
C/C++
#include "RtcServices.h"
#include <RTClib.h>

RtcServices::RtcServices(void) : _rtc(){}
String RtcServices::getData(void){return _data;}
void RtcServices::handleDataHorarioAbastecimento(void){
  _rtc.begin();
  DateTime now = _rtc.now();
  char buf1[] = "YYYY-MM-DDThh:mm:ssZ";
  _data = now.toString(buf1);}
```