WILLIAN ALVES DE ALMEIDA

PROJETO E IMPLEMENTAÇÃO DE UMA ARQUITETURA BÁSICA IOT APLICADA A UM ALIMENTADOR DE ANIMAIS

LAVRAS – MG

2023

WILLIAN ALVES DE ALMEIDA

PROJETO E IMPLEMENTAÇÃO DE UMA ARQUITETURA BÁSICA IOT APLICADA A UM ALIMENTADOR DE ANIMAIS

Trabalho de conclusão de curso apresentado 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. Bruno de Abreu Silva

Orientador

LAVRAS – MG

2023

WILLIAN ALVES DE ALMEIDA

PROJETO E IMPLEMENTAÇÃO DE UMA ARQUITETURA BÁSICA IOT APLICADA A UM ALIMENTADOR DE ANIMAIS

Trabalho de conclusão de curso apresentado 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 12 de Dezembro de 2023.

Prof. Dr. Wilian Soares Lacerda	UFLA
Prof. Dr. Paulo Afonso Parreira Júnior	UFLA

Prof. Dr. Bruno de Abreu Silva
Orientador

LAVRAS – MG
2023

Dedico aos meus pais e companheiros de curso.

AGRADECIMENTOS

Agradeço, primeiramente, a Deus por me dar a vitalidade necessária para concluir o curso. Agradeço aos meus pais, por me fornecerem tudo de que precisei do início ao fim. Agradeço especialmente a mulher que foi como a vó que não tive, que estará para sempre em meu coração, Dona Rafaela Leão, assim como agradeço a seus familiares, por todo carinho e irmandade.
Também agradeço ao Professor Dr. Bruno de Abreu Silva pela disponibilidade, paciência e sempre boa vontade para me auxiliar no que precisei.
Por fim, agradeço à Universidade Federal de Lavras e a todos os professores e funcionários por tornar tudo isso possível, para o processo da minha formação profissional e pessoal.

"O senhor é o Deus eterno. Ele fortalece o cansado e dá grande vigor ao que está sem forças. Até os jovens se cansam e ficam exaustos, e os moços tropeçam e caem; mas aqueles que esperam no senhor renovam as suas forças. Voam alto como águias; correm e não ficam exaustos, andam e não se cansam." (Isaías 40:28-31)

"Peça a Deus que abençoe os seus planos, e eles darão certo."(Provérbios 16:3)

RESUMO

O estudante ou profissional de Engenharia de Controle e Automação tem como objetivo acadêmico capacitar-se e integrar diversas áreas, como elétrica, eletrônica, mecânica e computação, buscando um conhecimento mais abrangente, o que se configura como uma vantagem. Porém, surge então o desafio da falta de conhecimento mais específico ou do caminho a ser seguido para obter tal conhecimento em alguns casos. Indivíduos interessados em seguir carreira na área de sistemas embarcados e atuar com elementos da indústria 4.0, como IoT (do inglês, *Internet of Things*), computação em nuvem, entre outros, podem deparar-se com desafios significativos devido à falta de conhecimento sobre os métodos de ingresso nesse universo. O presente trabalho vem com o objetivo de projetar e implementar uma arquitetura IoT básica, utilizando, como estudo de caso, um alimentador para animais controlado à distância, levando em consideração ferramentas, recursos e materiais gratuitos. Para isso, foi utilizado o microcontrolador Esp32. Pensando em computação em nuvem, foi criado um servidor em Node Js fornecendo serviço ao microcontrolador utilizado e acessando também um banco de dados Firebase de onde irá extrair os dados. Por fim, para a interação entre o usuário e sistema foi criado uma aplicação Web, usando o framework Angular. Os resultados obtidos se mostraram satisfatórios, já que foi comprovada a eficácia do funcionamento da arquitetura, provendo ração aos animais em quantidades e horários especificados. Houve no decorrer do desenvolvimento vários desafios, tais como a tratamento de dados sensíveis, troca dos módulos relés inicialmente utilizados, etc.

Palavras-chave: Arquitetura IoT, Esp32, Servidor, Banco de Dados, Angular.

ABSTRACT

The student or professional of Control and Automation Engineering has the academic objective of training and integrating different areas, such as electrical, electronics, mechanics and computing, seeking more comprehensive knowledge, which is an advantage. However, the challenge then arises of the lack of more specific knowledge or the path to be followed to obtain such knowledge in some cases. Individuals interested in pursuing a career in the area of embedded systems and working with elements of industry 4.0, such as IoT (*Internet of Things*), cloud computing, among others, may face significant challenges due to the lack knowledge about the methods of entering this universe. The present work aims to design and implement a basic IoT architecture, using, as a case study, a remotely controlled animal feeder, taking into account free tools, resources and materials. For this, the Esp32 microcontroller was used. Thinking about cloud computing, a Node Js server was created providing service to the microcontroller used and also accessing a Firebase database from which the data will be extracted. Finally, for interaction between the user and the system, a Web application was created using the Angular framework. The results obtained were satisfactory, as the effectiveness of the architecture's functioning was proven, providing feed to the animals in specified quantities and times. There were several challenges during the development, such as the processing of sensitive data, changing the relay modules initially used, etc.

Keywords: IoT architecture, Esp32, server, database, Angular.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Algo que vem se tornando cada vez mais comum no cotidiano humano é a utilização da robótica de menor escala, os chamados sistemas embarcados, que surgiram entre as décadas de 1950 e 1960. Conforme descrito por Lee e Srinivasan (2016):

> "O surgimento dos sistemas embarcados pode ser rastreado até o desenvolvimento dos primeiros circuitos integrados, que permitiram a integração de múltiplos componentes eletrônicos em um único chip de silício"(LEE; SESHIA, 2016).

Inicialmente, os sistemas embarcados foram projetados para atender a necessidades específicas, com o objetivo de executar apenas uma tarefa em particular. Porém, com o passar do tempo e desenvolvimento de novas tecnologias, houve um incremento nessas plataformas de tal forma que, dadas as devidas proporções, se assemelham ao comportamento dos computadores, podendo executar tarefas diferentes, com requisitos de projeto mais amplo. Os sistema embarcados têm a capacidade de unir diversas funcionalidades, como sistema para controle hídrico, térmico, elétrico e o que mais for necessário para alcançar o que é desejado.

No ano de 2011, pela primeira vez foi utilizado o termo Indústria 4.0, durante a Feira de Hannover, na Alemanha. Em 2013, foi apresentado na mesma feira um estudo em versão final, propondo avanços e inovações tecnológicas na área da automação, integração de tecnologias e robótica para aumento de produtividade e virtualização de processos, assim surgindo a Quarta Revolução Industrial, conhecida como Indústria 4.0 (SEBRAE, s.d.).

Com a chegada da indústria 4.0, houve uma ampliação das possibilidades para a utilização dos conceitos e aplicações da robótica e sistemas embarcados, não mais se limitando à comumente chamada “indústria”, ou seja, os grandes centros industriais onde são fabricados os mais diversos produtos, desde o trabalho que objetiva a produção do aço, muito lembrado pelos altos fornos, passando pela fabricação de automóveis, até os processos robóticos que auxiliam na produção e processamento de alimentos.

Um dos elementos dessa indústria 4.0 é a conectividade, representada pelo conceito de IoT (Internet das Coisas). De forma sucinta, de acordo com a IoT os dispositivos tendem a se tornar “inteligentes” a partir do momento que estão trocando informações entre si, conectados em rede (OLIVEIRA, 2017).

Ao unir a robótica dos sistemas embarcados com a conectividade IoT, fomentada pela possibilidade de transmissão de dados e comandos através da internet, o leque de possibilidades se torna virtualmente infinito. Existem diversas aplicações de arquiteturas IoT, uma delas é na

domótica, já que o objetivo não é apenas controlar a residência, mas que esse controle possa ser feito a longas distâncias, proporcionando mais segurança residencial e comodidade ao usuário. Mas não é exclusivamente na domótica ou na vida urbana que o controle à distância de pequenos sistemas robóticos impacta, basta pensar no controle de plantações, obtendo medidas como umidade do solo, temperatura local, velocidade dos ventos e outras variáveis climáticas. Outro exemplo é o controle de posição do gado em um pasto com a utilização dos famigerados drones ou mesmo na alimentação dos mesmos, como seria cômodo a um fazendeiro, criador de frangos por exemplo, poder configurar através de seu computador ou aparelho celular, horários e quantidades de alimento para seus animais. No mercado atualmente existem dois alimentadores muito comuns, em que com frequência são sugeridos ao procurar por este tipo de produto na internet. Esses dois produtos são: o alimentador Peti Dogis da empresa Dogis e o alimentador da marca RoHS.

Mediante ao discorrido, foi proposta uma arquitetura básica IoT utilizando como estudo de caso a implantação de um alimentador para animais.

1.1 Objetivo Geral

O objetivo deste trabalho foi projetar e implementar um arquitetura IoT (Internet das Coisas) básica para um alimentador de animais, usando o microcontrolador ESP32 e ferramentas digitais gratuitas.

1.2 Objetivos Específicos

- Implementar servidor Node.js e banco de dados Firebase.
- Desenvolver aplicação web por meio do framework Angular de forma responsiva.
- Projetar o alimentador físico e sua programação com ESP32.

1.3 Justificativa

Restringindo um pouco a gama de possibilidades de aplicações de projetos IoT apenas ao setor de animais domésticos, é possível visualizar a relevância financeira do ramo de alimentadores de animais. Conforme reportado pelo portal virtual *O Presente Rural* (2023), o setor de animais de estimação no Brasil, fechou o ano de 2022 com um faturamento total de R$

41,9 bilhões, sendo que a área de alimentação de pets representou a maior parcela, totalizando R$ 33,3 bilhões (80%), o que corresponde a aproximadamente 80% do faturamento total. Esses números destacam a importância financeira desse setor e indicam que o desenvolvimento de soluções inovadoras para alimentadores de animais domésticos possui um significativo potencial de retorno financeiro. Unindo o aspecto de grande retorno financeiro da área de pets, juntamente com o fato da população brasileira passar cada vez mais tempo fora de suas residências surge então a primeira motivação para construção do projeto. Essa primeira motivação vem do desejo de aplicar a arquitetura IoT a um alimentador a fim de gerar conforto e praticidade aos donos de animais, para que os mesmos possam se ausentar de suas residências com tranquilidade. Dessa forma, os animais receberão as porções de rações apenas nos horários e quantidades previamente estipulados pelo dono.

Do ponto de vista pessoal do autor, a principal justificativa para a definição do tema deste trabalho é, dado o seu interesse nessa área, aprimorar e aplicar seus conhecimentos em IoT, uma vez que durante a sua graduação não houve a oportunidade de cursar disciplinas, obrigatórias ou eletivas, que tivessem um enfoque nesse campo do conhecimento.

Considerando que esta pode ser uma realidade também para outros estudantes de outras instituições, surge então a ideia de projetar e implementar uma arquitetura IoT básica, usando os alimentadores como estudo de caso. O projeto tem a intenção de ser uma possível referência para que os interessados possam sair do desconhecimento em relação a área de embarcados volta a IoT, podendo conhecer de forma simples uma arquitetura que pode ser utilizada como pontapé inicial para adentrar nessa área.

1.4 Organização e Estrutura

O restante deste documento está dividido em 4 capítulos. O segundo capítulo contém explicações sobre o funcionamento dos equipamentos, componentes e tecnologias utilizadas. No terceiro capítulo, são apresentados os métodos utilizados no desenvolvimento. No capítulo 4, é discorrido o resultado e suas considerações pertinentes. No capítulo final, são apresentadas as conclusões e possíveis incrementos e melhorias futuras.

2 REFERENCIAL TEÓRICO

Neste capítulo será realizado um embasamento acadêmico sobre assuntos necessários ao entendimento dos elementos do projeto. Na seção 2.1 será explicado o conceito de Sistemas Embarcados, na 2.2 a definição e uma explicação mais extensa de Internet das Coisas (IoT) que vem a ser o coração do projeto. Por fim, na seção 2.3 será discorrido sobre Banco de Dados.

2.1 Sistemas Embarcados

Os sistemas embarcados detêm poder computacional em um equipamento de pequenas dimensões, recursos de máquina "empacotados" em um circuito integrado, sendo mais que um computador simples, mas objetivando a execução de apenas uma tarefa (CUNHA, 2007).

Outra visão interessante acerca de embarcados é feita por Tim Wilmshurst, quando o mesmo explica sistemas embarcados sendo computadores ocultos, cujos usuários interagem sem nem mesmo perceber, projetados junto a produtos para que seja possível o seu controle (WILMSHURST, 2010).

Segundo Barros e Cavalcante (2010) os sistemas embarcados detêm características em comuns:

1. Funcionalidade única: usualmente um sistema embarcado executa somente um programa repetidamente. Por exemplo, um *pager* é sempre um *pager*, enquanto que um computador pessoal pode executar uma variedade de programas;

2. Restrições de projeto mais rígidas: todos os sistemas de computação possuem em geral alguma restrição de projeto a ser satisfeita, como por exemplo custo, tamanho, desempenho, potência dissipada, etc. Nos sistemas embarcados, no entanto, estas restrições são normalmente mais rígidas, por exemplo, o custo de um sistema não pode ser muito alto para não onerar o custo do equipamento, o tempo de resposta deve permitir em várias aplicações processamento em tempo real e devem dissipar pouca potência para permitir uma maior duração da bateria ou não necessitar de um sistema de refrigeração;

3. Sistemas reativos de tempo real: muitos sistemas embarcados devem reagir a mudanças no ambiente e devem fornecer resultados em tempo real. Por exemplo, um piloto automático *(cruise controller)* continuamente monitora e reage à velocidade e aos sensores de freio. Ele deve computar a aceleração e desaceleração repetidamente num intervalo de tempo. Caso haja um retardo, o controle do carro pode ser perdido.

No presente momento, é complexo classificar um sistema como sendo puramente embarcado, pois, por mais restrito em memória, desempenho nas trocas de dados e potência computacional, ainda sim, pode desempenhar mais de uma função específica. Mesmo aparelhos como televisores não podem ser considerados embarcados, se utilizado o conceito com total rigidez, dado que os televisores mais atuais possuem acesso à internet, uma capacidade a mais, além de simplesmente captar informações de radiofrequência e transmitir via gerenciamento de luz em um *display*.

2.2 Internet das Coisas (IoT)

Em 1991, Weiser (1991) expressou pela primeira vez a conceituação de Internet das Coisas (IoT - do inglês, *Internet of Things*), discorrendo sobre a tendência à onipresença dos dispositivos e tecnologias no cotidiano humano, o que foi chamado de Computação Ubíqua. Cenário esse que é perceptível cada vez mais, no qual até mesmo geladeiras, relógios, acessórios e roupas, possuem a capacidade de troca de informações com o objetivo de auxiliar o cotidiano das pessoas.

Dado que a IoT é uma área de pesquisa e desenvolvimento que tem ganhado destaque nas últimas décadas, envolvendo a interconexão de dispositivos, objetos e sistemas, sua definição é multifacetada e abrange. A interconexão de objetos do mundo real com a internet, permitindo a coleta, transmissão e compartilhamento de informações em tempo real é uma possível definição de IoT (ATZORI; IERA; MORABITO, 2010).

Sendo a conectividade o principal atributo na área de sistemas IoT, Maschietto (2021) afirmam que:

> "Estar conectado não é apenas uma necessidade do usuário final, é uma imposição ao mundo dos negócios, pois o consumo está cada vez mais direcionado a fatores que envolvem mobilidade, flexibilidade, comodidade e velocidade, em que grande parte dos consumidores estão conectados à internet junto de seus dispositivos."

Assim, mesmo havendo arquiteturas em que a conexão com a internet pode ser fraca, o futuro da IoT é uma forte conexão dos sistemas locais com a rede mundial de computadores interligados, assim como o processamento bruto dos dados também ficará a cargo de servidores e serviços remotos.

A Internet das Coisas tem um amplo espectro de aplicações em diferentes setores. Na área da saúde, por exemplo, possibilita o monitoramento remoto de pacientes, a gestão de medicamentos e a análise de dados para tomada de decisões clínicas (Atzori et al., 2010). Além

disso, a IoT tem aplicações em indústrias, cidades inteligentes, transporte e diversos outros domínios.

Como esta área contém uma vasta gama de tecnologias com origens diferentes, cada uma contendo uma peculiaridade em particular, não há como definir apenas uma arquitetura genérica que contemple todos os possíveis casos de utilização (MASCHIETTO, 2021).

Alguns conceitos muito importantes tanto para o melhor entendimento de arquiteturas IoT, quanto do trabalho em questão são *Broker, Gateway e Middleware*.

Broker: é um programa cujo objetivo é receber, enfileirar e despachar as mensagens recebidas dos microcontroladores para a interface web, tendo assim como aspecto mais forte o roteamento das informações (OLIVEIRA; KNIESS, 2023).

Gateway: podendo ser um software ou um hardware, o gateway faz a intermediação entre dispositivos IoT e a comunicação com aplicações *back-end* ou com a nuvem.

Middleware: o middleware funciona como um sistema/tecnologia intermediária, servindo como conexão entre diversas tecnologias tais como redes IoT, banco de dados e outros recursos e ferramentas (AMAZON, 2023).

Computação em Nuvem: é um modelo de computação que permite acesso a recursos e serviços de TI, sem a necessidade de tê-los de forma física ou locais, assim em vez de ter uma máquina hospedando site, banco de dados, servidor e demais recursos de armazenamento e processamento, toda essa responsabilidade é delegada a provedores de nuvem, tal como Amazon Web Services (AWS) [1]. Como esses provedores trabalham com demanda de recursos, basta fazer *upgrade* do plano contratado para aumentar todo o poder computacional, de armazenamento e recursos que se necessita (AMAZON, 12 de Dezembro de 2023).

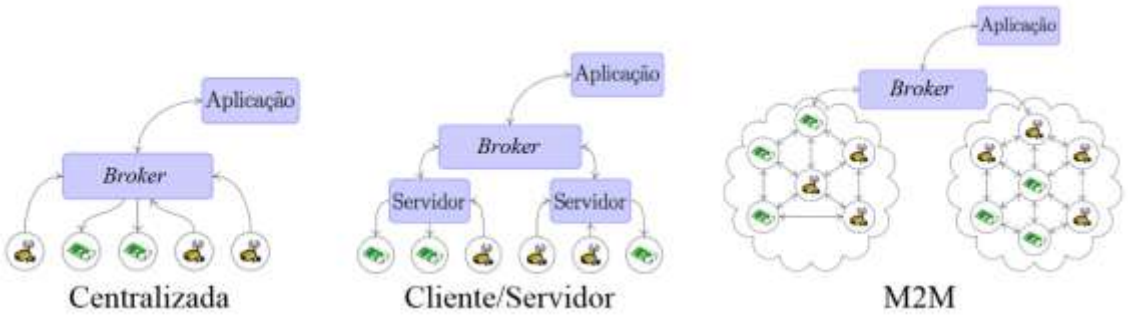
Como dito anteriormente, a área de arquiteturas IoT é muito ampla, havendo uma grande quantidade de arquiteturas e abordagens possíveis. Na Figura 2.1 é possível ver modelos de arquiteturas de Internet das Coisas de acordo com os estudos de Mussio, Maia e Lopes (2015).

Pode ser observado que o Broker gerencia as informações vindas dos dispositivos e distribui para a aplicação, na arquitetura centralizada. Já na arquitetura Cliente/Servidor a transmissão de dados ocorre de forma bidirecional entre servidor-broker e broker-aplicação, assim o broker funciona como um componente que centraliza as informações dentro do sistema. Por fim, no modelo M2M (*Machine to Machine*) pela alta comunicação entre os próprios dispositivos de campo, tais como sensores, atuadores e os dispositivos IoT como sistemas embarcados,

[1] <https://aws.amazon.com/pt>

há uma menor dependência de conexão com a internet se comparado à modelos como cliente/-servidor, mas ainda sim, essa conexão ainda é consideravelmente importante.

Figura 2.1 – Alguns tipos de arquitetura IoT



Fonte: Mussio, Maia e Lopes (2015)

Outra abordagem muito comum quando se trata de arquiteturas IoT é o conceito de arquiteturas em camadas. No livro Maschietto (2021) são discorridas as arquiteturas de 3, 4, 5 ou 6 camadas. Na arquitetura de 4 camadas, por exemplo, há a seguinte divisão:

- **Camada das Coisas:** nesta se encontram as redes de sensores, medidores, atuadores e também dispositivos embarcados, sendo nessa onde ocorre a captação das variáveis do ambiente.

- **Camada de Adaptação:** essa camada faz a intermediação e agregação dos dados coletados pela Camada das Coisas e proporciona meios para que sejam enviadas as informações para a Camada de Internet, tendo de lidar com protocolos de comunicação para desempenhar tal função.

- **Camada de Internet:** responsável pela conectividade entre os dispositivos IoT e gateway, servidor e demais componentes da rede de internet, fazendo o gerenciamento de endereços IP ou outros identificadores de rede, de forma confiável e segura, fazendo uso de protocolos necessários para tal conectividade, como MQTT (Transporte de telemetria de fila de mensagens), CoAP (Protocolo de aplicação restrito) ou HTTP (Protocolo de Transferência de Hipertexto).

- **Camada de Aplicação:** é por meio desta camada que os usuários podem interagir com o sistema, fazendo o uso de interfaces, podendo enviar comandos aos atuadores e fazer leitura de dados já processados, analisados e filtrados, tais como gráficos, tabelas e demais

recursos de visualização, fornecendo controle do sistema por meio da virtualização para o usuário final.

2.2.1 Protocolos IoT

Protocolos em IoT podem ser entendidos como um conjunto de normas e regras, de tal maneira que permitam a comunicação e fluxo de dados entre dispositivos IoT, softwares ou soluções tecnológicas, garantindo a integridade dos dados visando a segurança dos mesmos (MASCHIETTO, 2021).

De acordo com Delicato, Pires e Batista (2013), a função dos protocolos é possibilitar a comunicação entre dispositivos de forma fácil, agindo como uma linguagem universal, independente do dispositivo, marca, fabricante e modelo, permitindo que o controle seja realizado de qualquer lugar do mundo.

Dentre muitos protocolos existentes, há alguns que se destacam. Como sugerem Maschietto (2021), em se tratando de protocolos para comunicação e transporte na camada de rede, aqueles que mais sobressaem são:

- **WIFI:** muito disseminado, é uma tecnologia baseada no padrão 802.11n com o intuito de promover a transmissão de dados de dispositivos eletrônicos, utilizando ondas de rádio em redes locais ou de longas distâncias.

- **Bluetooth:** como descreve no site oficial da Intel, a tecnologia *bluetooth* visa conectar dois dispositivos diretamente a um curto alcance, sem a necessidade de um suporte de rede, como roteador ou ponto de acesso (INTEL, 2023).

- **LoraWAN:** voltado à conexão com redes de longo alcance, como rede WAN (Rede de área ampla), próprio para redes de baixa potência, propiciando segurança e mobilidade a um baixo custo para cenários IoT.

Quando o assunto é camada de aplicação, alguns dos protocolos utilizados na área de IoT, são:

- **Transporte de Telemetria da Fila de Mensagens (MQTT):** projetado para ser leve e eficiente, operando por meio de assinatura e publicação, estabelecendo uma conexão persistente entre dispositivo IoT e *middleware*, diminuindo a latência e o consumo de energia, trabalhando de forma assíncrona, em que os assinantes “escutam” as publicações emanadas.

- **Protocolo de Aplicação Restrita (CoAP):** criado com o objetivo de ser uma boa opção para aplicativos M2M, especializado para operar em redes com recursos restritos.
- **Protocolo Extensível de Mensagens e Presença (XMPP):** utilizando o formato de mensagem XML (linguagem de marcação extensível), este protocolo foi originalmente criado para trabalhar com mensagens instantâneas.
- **HTTP:** podendo ser considerado o protocolo mais conhecido, principalmente dentro do contexto de comunicação pela internet de forma geral. O protocolo HTTP é a base da comunicação World Wide Web, utilizando o padrão de solicitação e resposta. Uma melhoria que pode ser implementada para tornar esse protocolo mais seguro é a criptografia das mensagens, realizada por meio do protocolo HTTPS.

2.3 Banco de Dados

Ao pensar em armazenamento de dados, o que vem à mente geralmente costuma ser o termo "banco de dados", remetendo de forma bruta a um local como HD's de computadores, no qual se pode gravar arquivos, porém, apenas a parte do armazenamento não totaliza esse conceito, sendo então necessário uma determinada organização e estruturação lógica nessa acumulação de dados. Os bancos de dados são utilizados há décadas, tanto por grandes multinacionais de quaisquer setores até as menores empresas e com o tempo, mesmo microempreendimentos desejaram ter em posse um histórico dos dados, utilizando tal recurso.

Segundo o site oficial da Oracle:

> "Um banco de dados é uma coleção organizada de informações - ou dados - estruturadas, normalmente armazenadas eletronicamente em um sistema de computador. Um banco de dados é geralmente controlado por um sistema de gerenciamento de banco de dados (DBMS)."(ORACLE, 2023)

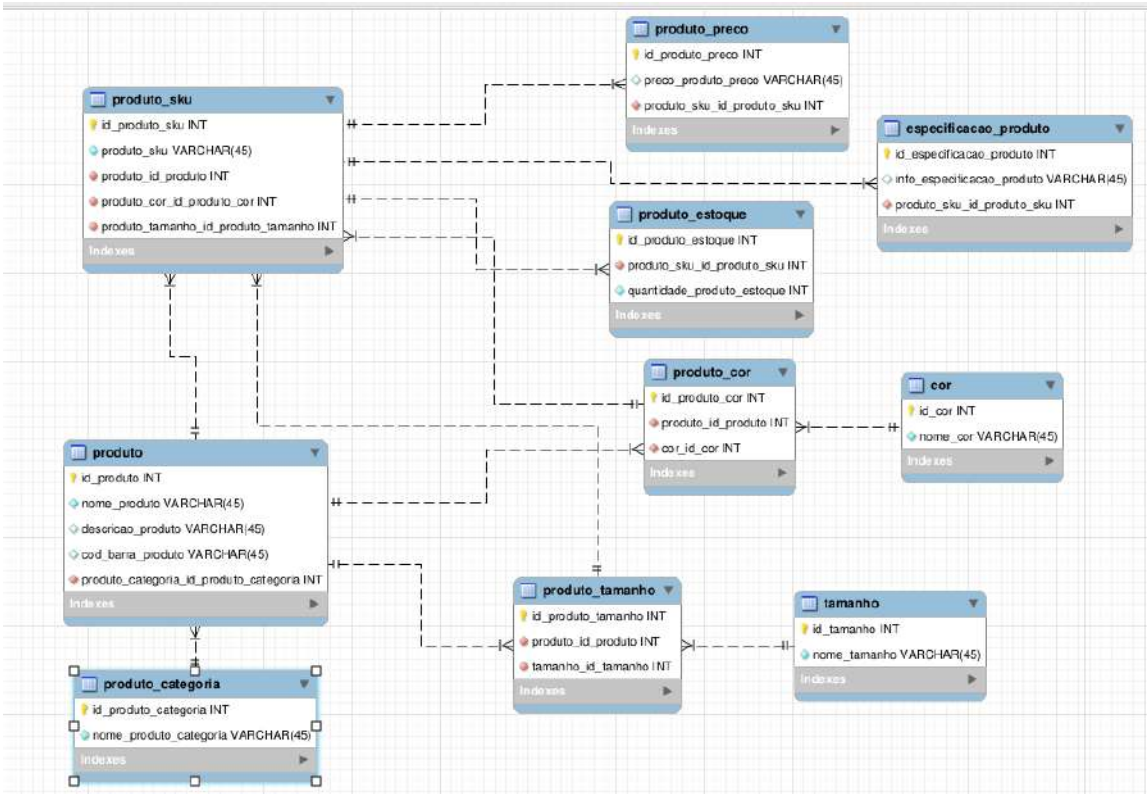
Há muitos tipos de banco de dados, cada um apropriado a necessidades específicas, porém para o presente trabalho pode-se dividir em dois grandes grupos: banco de dados relacionais e banco de dados não relacionais.

O primeiro se trata de um banco cuja estrutura se baseia em tabelas, atributos, valores, relação entre tabelas através das chamadas chaves primárias e chaves estrangeiras, chaves essas que representam os vínculos entre as tabelas, como apresentado na Figura 2.2. Esse tipo de banco possui uma estrutura rígida a mudanças se comparado ao não relacional, requerendo na maioria dos casos conhecimento sobre a linguagem de consulta SQL, utilizada para consultar,

remover, atualizar, armazenar e recuperar informações de um banco relacional (SERVICES, 2023).

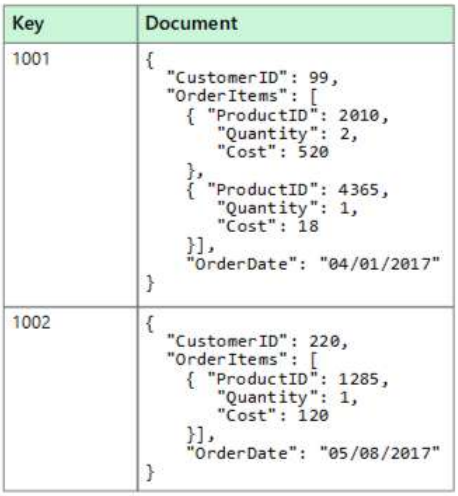
Quando a flexibilidade e facilidade de estruturação de um banco de dados são fundamentais, os bancos de dados NoSQL se destacam. Em vez de usar tabelas, os bancos de dados não relacionais adotam modelos de armazenamento otimizados. Um padrão muito comum é a utilização do formato chave/valor, semelhante a objetos no formato JSON (Notação de objeto JavaScript) como exemplificado na Figura 2.3, formato esse cuja estruturação de dados é simples e amplamente aceita por inúmeras tecnologias, seja em solicitações de servidores/API (Interface de Programação de Aplicativos), microcontroladores, bancos de dados não relacionais, como o Firestore, etc. (MICROSOFT, 2023).

Figura 2.2 – Modelo Relacional



Fonte: Dzp (2023)

Figura 2.3 – Estrutura de armazenamento não relacional



Key	Document
1001	{ "CustomerID": 99, "OrderItems": [{ "ProductID": 2010, "Quantity": 2, "Cost": 520 }, { "ProductID": 4365, "Quantity": 1, "Cost": 18 }], "OrderDate": "04/01/2017" }
1002	{ "CustomerID": 220, "OrderItems": [{ "ProductID": 1285, "Quantity": 1, "Cost": 120 }], "OrderDate": "05/08/2017" }

Fonte: Microsoft (2023)

3 MATERIAIS E MÉTODOS

O conteúdo que virá a seguir conterá os materiais e ferramentas utilizadas no decorrer do projeto, subdividido em materiais físicos, recursos digitais e custos do projeto. Em se tratando da construção física e controle do protótipo foi utilizado o microcontrolador ESP32 para fazer o processo de controle de vazão da ração. Através da programação o microcontrolador aciona o relé 1 ou o relé 2, afim de energizar o motor em sentido de abertura ou fechamento, liberando ou impedindo a vazão. Para alimentar o motor foi usada a fonte de tensão ajustável, para alimentar o ESP32, os relés e leds foi utilizado um cabo USB conectado um computador.

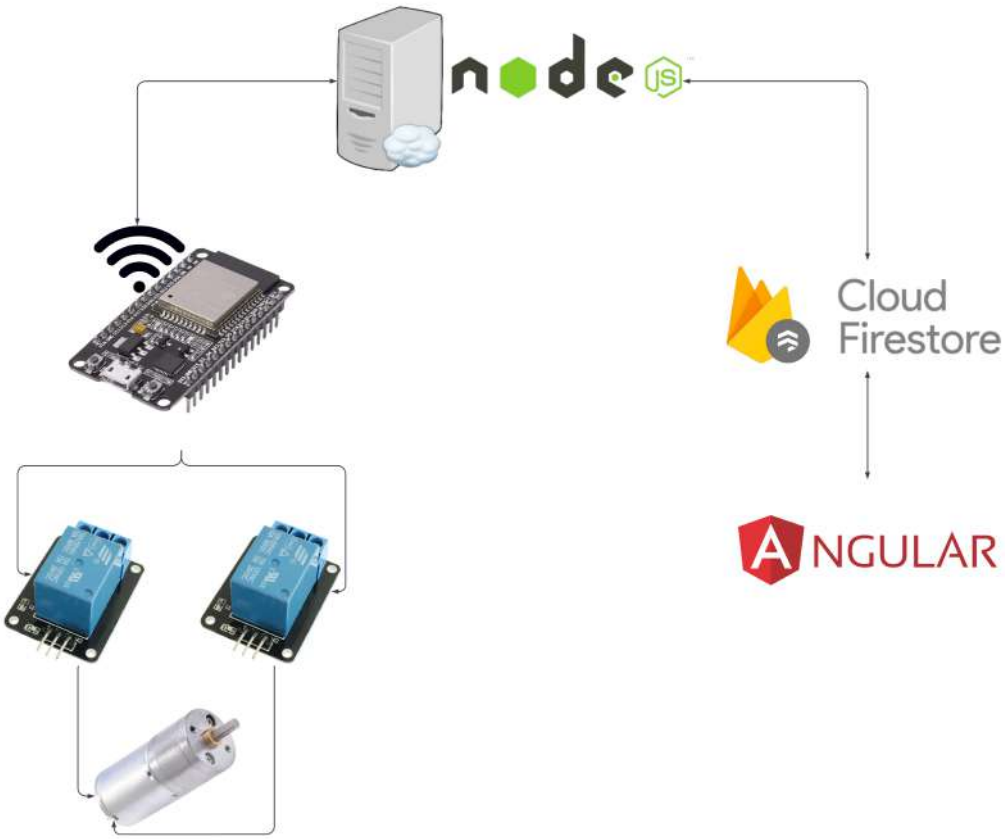
Quando trata-se de recursos digitais, utilizou-se a plataforma Firebase para: hospedar o site gerado que serviu de interface de interação; criação do banco de dados Firestore (local onde foram armazenados os dados); criação do serviço de autenticação do usuário. Para extrair os dados do banco para fornecê-los para o ESP32, foi construído um servidor utilizando a tecnologia Node.js.

Dentro várias opções de linguagens para construção do servidor, tal como PHP, que é um linguagem consolidada a muito tempo, escolheu-se trabalhar com o Javascript. Dado a existência da tecnologia Node.js e do framework Angular que utilizam Javascript, unidos ao fato das validações de campos necessitarem da linguagem Javascript, teve-se então esse agregado de pontos em comum para a escolha dessa linguagem. Se fosse utilizado por exemplo PHP para criação do servidor, haveria ainda a necessidade de uma aplicação web e mesmo que essa fosse feita em PHP, haveria a necessidade de validar os campos com Javascript, sendo necessário utilizar duas linguagens diferentes. Porém, ao utilizar o conjunto, Node.js e Angular, por meio da programação Javascript, foi reduzido o número de linguagens usadas, melhorando a velocidade de desenvolvimento do projeto.

Antes de abordar especificamente cada material ou recurso utilizado, é possível ter uma noção geral de como foi realizado o projeto ao analisar a Figura 3.1. Nessa imagem é apresentada a arquitetura construída, em que os atuadores como motor e relés, fazem parte da Camada das Coisas. Constituindo a Camada de Adaptação tem-se o ESP32, fazendo o controle dos equipamentos da Camada das Coisas e se comunicando com o servidor Node.js. Este servidor Node.js pertence à Camada de Internet, servindo de intermediário entre o ESP32 (dispositivo IoT) e o banco de dados Firestore. Por fim, tem-se a aplicação web desenvolvida utilizando o framework Angular, aplicação essa pertencente à Camada de Aplicação. É por meio do site

que os usuários iteragem com o sistema, cadastrando os horários e quantidades de ração para o animal.

Figura 3.1 – Apresentação geral da arquitetura desenvolvida



Fonte: do Autor

3.1 Dispositivos de Hardware

Esta seção será referente aos recursos de hardware. A subseção 3.1.1 apresenta a placa de desenvolvimento ESP32-DevKitC (microcontrolador utilizado no projeto). Na seção 3.1.2 é mostrado a Fonte Ajustável utilizada para alimentar o motor. A subseção 3.1.3 trás o Motor 25GA370 utilizado no controle da vazão de ração. Por fim, na subseção 3.1.4 é exibido o Módulo Relé que foi utilizado para controlar a energia direcionada ao motor e o seu sentido de giro.

Figura 3.2 – Microcontrolador ESP32

Fonte: Eletrogate (2023b)

3.1.1 Placa de desenvolvimento ESP32-DevKitC

A placa de desenvolvimento ESP32-DevKitC é um sistema embarcado de baixo custo, baseada no microcontrolador ESP32 exibido na Figura 3.2, projetada pela Espressif Systems[1]. Esta placa é ideal para projetos envolvendo Internet das Coisas, devido à sua conectividade nativa com Wi-Fi e Bluetooth.

As Figuras 3.3 e 3.4 exibem as especificações técnicas do dispositivo em questão.

Figura 3.3 – Tabela de especificações ESP32 - parte1

Categories	Items	Specifications
Certification	RF certification	See certificates for ESP32-WROOM-32
	Wi-Fi certification	Wi-Fi Alliance
	Bluetooth certification	BQB
	Green certification	RoHS/REACH
Test	Reliablity	HTOL/HTSL/uHAST/TCT/ESD
Wi-Fi	Protocols	802.11 b/g/n (802.11n up to 150 Mbps)
		A-MPDU and A-MSDU aggregation and 0.4 μs guard interval support
	Center frequency range of operating channel	2412 ~ 2484 MHz
Bluetooth	Protocols	Bluetooth v4.2 BR/EDR and Bluetooth LE specification
	Radio	NZIF receiver with –97 dBm sensitivity
		Class-1, class-2 and class-3 transmitter
		AFH

Fonte: Eletrogate (2023b)

[1] <https://www.espressif.com/>

Figura 3.4 – Tabela de especificações ESP32 - parte2

Categories	Items	Specifications
	Audio	CVSD and SBC
Hardware	Module interfaces	SD card, UART, SPI, SDIO, I2C, LED PWM, Motor PWM, I2S, IR, pulse counter, GPIO, capacitive touch sensor, ADC, DAC, Two-Wire Automotive Interface (TWAI®), compatible with ISO11898-1 (CAN Specification 2.0)
	Integrated crystal	40 MHz crystal
	Integrated SPI flash	4 MB
	Operating voltage/Power supply	3.0 V ~ 3.6 V
	Operating current	Average: 80 mA
	Minimum current delivered by power supply	500 mA
	Recommended operating ambient temperature range	–40 °C ~ +85 °C
	Package size	18 mm × 25.5 mm × 3.10 mm
	Moisture sensitivity level (MSL)	Level 3

Fonte: Eletrogate (2023b)

O processador ESP32-D0WDQ6 é um microcontrolador de 32 bits com dois núcleos Tensilica LX6 e uma frequência de clock de até 240MHz. A placa também possui 520KB de memória SRAM, que permite armazenar temporariamente dados em execução.

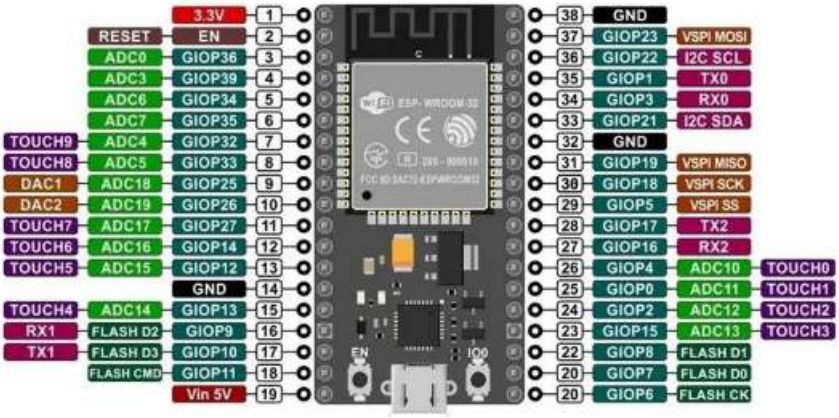
O armazenamento da placa é de 4MB de flash, que pode ser usado para armazenar programas e dados permanentes. Além disso, a placa suporta interfaces como UART (Receptor/transmissor assíncrono universal), SPI (Interface periférica serial), I2C (Circuito Interintegrado), I2S, SD/MMC (Cartão Digital Seguro), Ethernet, CAN 2.0 (Rede de área do controlador), IR (Infravermelho), PWM (Modulação por Largura de Pulso), ADC (Conversor Analógico para Digital) e DAC (Conversor Digital para Analógico), permitindo que a placa se comunique com outros dispositivos.

A placa também possui 38 GPIOs (*General Purpose Input/Output*) que permitem a conexão de sensores, atuadores e outros dispositivos. A placa pode ser alimentada com uma tensão de 3.3V ou 5V, tornando-a compatível com uma ampla variedade de fontes de alimentação. A fonte de alimentação usada nesse projeto para suprir as necessidades do ESP32 foi a energia vinda de um computador por meio do cabo usb com 5V.

A conectividade da placa é um dos seus recursos mais importantes, com suporte para Wi-Fi 802.11 b/g/n/e/i e Bluetooth v4.2 BR/EDR e BLE. Isso permite que a placa se conecte à Internet e se comunique com outros dispositivos Bluetooth. Esses recursos são ideais para projetos de IoT, permitindo que a placa colete e transmita dados pela Internet ou se comunique com outros dispositivos para realizar tarefas específicas.

Na Figura 3.5 é possível entender como estão dispostos os pinos do ESP32 utilizado e ao que se dirige cada pino.

Figura 3.5 – Esquemático eletrônico e funcional ESP32 38 pinos



Fonte: Eletrogate (2023b)

3.1.2 Motor 25GA370

O motor escolhido foi o motor 25GA370 de corrente contínua de escovas, sendo alimentado por uma tensão de 12V, para baixo consumo e um torque adequado. No próprio site [2] do vendedor é possível escolher a rotação nominal do motor, sendo selecionado 12 rpm, para garantir torque suficiente ao sistema pela baixa rotação e melhor controle do equipamento. Havia possibilidades como motores de passo, porém que algum grão de ração pudesse travar o fechamento da portinhola por falta de torque do motor, escolheu-se esse motor de corrente contínua ao invés de um motor de passo. O equipamento é mostrado na Figura 3.6.

Figura 3.6 – Motor 25GA370

Fonte: Robobuilders (2023)

[2] <https://www.robobuilders.com.br/>

3.1.3 Módulo Relé

Os relés são componentes eletromecânicos que se comportam como chaves que abrem ou fecham mediante a presença ou ausência de tensão em seus terminais. O módulo relé da Figura 3.7 utilizado é um muito comum na aquisição de kits de eletrônica básica cuja especificações técnicas estão listadas na Figura 3.7.

Figura 3.7 – Módulo Relé

Fonte: Eletrogate (2023a)

Figura 3.8 – Especificações do Módulo Relé

TENSÃO DE OPERAÇÃO	5V DC (VCC e GND)
TENSÃO DE SINAL	TTL - 5V DC (IN)
CORRENTE TÍPICA DE OPERAÇÃO	80 mA
O RELÉ POSSUI	contato NA e NF
CAPACIDADE DO RELÉ	30 V DC e 10A ou 250V AC e 10A
TEMPO DE RESPOSTA	5~10ms
POSSUI	Indicador LED de funcionamento
DIMENSÕES	43mm (L) x 17mm (C) x 19mm (H)

Fonte: Eletrogate (2023a)

3.2 Recursos de Software

Esta subseção será referente aos recursos de software. A subseção 3.2.1 apresenta a linguagem Javascript, utilizada na maior parte do projeto . A subseção 3.2.2 trata da tecnologia utilizada para construir o servidor, chamada Node.js. Na subseção 3.2.3 cita o recurso Github utilizada para controle e armazenamento dos códigos gerados. Na subseção 3.2.4 é apresentada a ferramenta Insomnia para testar as rotas do servidor criado. Na subseção 3.2.5 é discorrido

sobre a plataforma de hospedagem Vercel. Na subseção 3.2.6 é exposto um pouco sobre a plataforma Firebase, extremamente importante para o projeto em questão. Por fim, na subseção 3.2.7 é citado o framework Angular, usado para desenvolver a aplicação web do projeto.

3.2.1 Javascript

Uma linguagem de programação é o "idioma", um conjunto de regras e símbolos, utilizado por seres humanos para dar instruções para uma máquina sobre o que fazer e de como agir em determinadas circunstâncias, de forma que possam interagir entre si. O uso de uma linguagem de programação que se propõe de forma lógica e estruturada a solucionar um problema é chamado de algoritmo (SEBESTA, 2015).

Há inúmeras linguagens de programação, cada uma voltada a objetivos diferentes, seja performance, compatibilidade com determinada ferramenta ou recurso, etc. Uma das linguagens mais usuais para desenvolvimento web é o Javascript, muito pela sua natividade com os navegadores atuais, já que os mesmos foram projetados para rodar o Javascript sem a necessidade de conversor ou ferramenta de tradução, eliminando um grau a mais de complexidade computacional se comparado às outras linguagens. Mas não é somente para desenvolvimento web que se pode extrair um grande potencial dessa linguagem, como também na construção de servidores, jogos, automação de tarefas, etc.

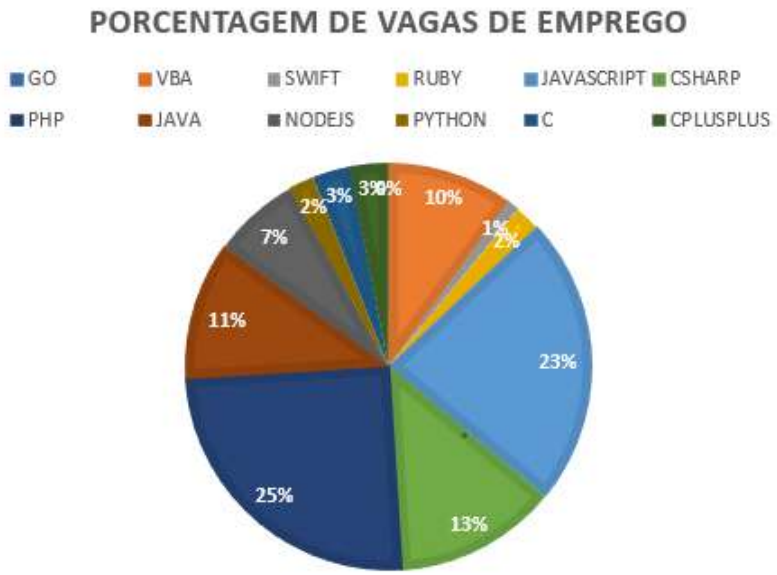
Por meio do estudo feito por Oliveira (2020), que objetivou captar e apurar dados sobre as linguagens mais requisitadas nas vagas de emprego ofertadas no Estado de São Paulo, foi possível constatar que Javascript é fortemente requisitada no mercado de trabalho como mostra o gráfico 3.9, muito devido às evoluções na linguagem e a popularidade relativamente recente do ambiente de execução Node.js, que também faz uso da mesma e permite, por exemplo, a criação de servidores. O Javascript é uma das linguagens mais utilizadas atualmente, assim como os frameworks e bibliotecas que fazem uso do mesmo, tais como Angular, React e Vue.

3.2.2 Node.js

Node.js[3] é um ambiente de execução de código em linguagem JavaScript que permite a execução no lado do servidor *(server-side)*, em vez de terceirizar tarefas para navegadores. O mecanismo de execução do Node.js é o V8, sendo este criado e utilizado pela Google para interpretação do Javascript.

[3] <https://nodejs.org/en>

Figura 3.9 – Gráfico comparativo de linguagens de Programação em Vagas de Emprego



Fonte: Adaptada de Oliveira (2020)

O Node.js se torna muito eficiente para aplicativos de rede que visam escalabilidade, pois trabalha sobre o paradigma não bloqueante, assim uma tarefa não precisa esperar que outra seja concluída para entrar em execução, já que as operações de entrada e saída são delegadas ao sistema operacional, deixando a *thread* principal de evento livre para receber as requisições realizadas (NODE.JS, 2023).

3.2.3 Github

Github[4] é um serviço que se apoia nos conceitos de computação em nuvem, próprio para gerenciamento e controle de versão de projetos, muito usufruído principalmente quando se trata de desenvolvimentos em equipe, já que um projeto pode ser subdividido em partes, que são desenvolvidos de forma independente e posteriormente unidos no que se chama de *branch master* ou *branch main* que é a ramificação em que todas as partes se convergem (HOSTINGER, 2023).

3.2.4 Insomnia

Insomnia[5] é um programa para desktop que permite testar API's ou recursos que dependem de API's de forma fácil. As API's são interfaces de processamento de aplicações entre um

[4] <https://github.com>
[5] <https://insomnia.rest>

servidor da Web e um navegador da Web. O Insomnia é software projetado para desenvolvedores, possibilitando o teste de rotas, utilização dos tipos de métodos e recursos passados como parâmetros em uma requisição, sendo capaz de testar aplicações web também (INSOMNIA, 2023).

3.2.5 Vercel

Vercel[6] é uma plataforma de hospedagem e implantação de aplicativos web, com suporte a diversas linguagens inclusive Javascript e a frameworks como Node.js, React, Next.js, etc. O Vercel traz funcionalidades que otimizam a aplicação, tal como: escalonamento automático, no qual os recursos necessários ao tráfego de dados atual se adaptam a essa demanda (VERCEL, 2023).

A Figura 3.10 mostra algumas das capacidades do plano gratuito da Vercel, sendo que essa não impõe limites claros de tamanho de projeto, pois o foco da fatura está sobre a quantidade e o consumo de recursos que os projetos fazem, tal como Largura de Banda, que no plano livre da Vercel chamado de Hobby é 100GB, ou seja, é o volume total de dados que é possível envia/receber do servidor Node.js projetado por mês.

3.2.6 Firebase

O Firebase [7] é uma plataforma mantida e fomentada pela Google, projetada para desenvolver aplicativos móveis e web de maneira eficaz e escalável, seguindo os princípios da computação em nuvem. Ele fornece uma ampla gama de serviços, ferramentas e funcionalidades que permitem que os desenvolvedores criem aplicativos confiáveis e altamente escaláveis.

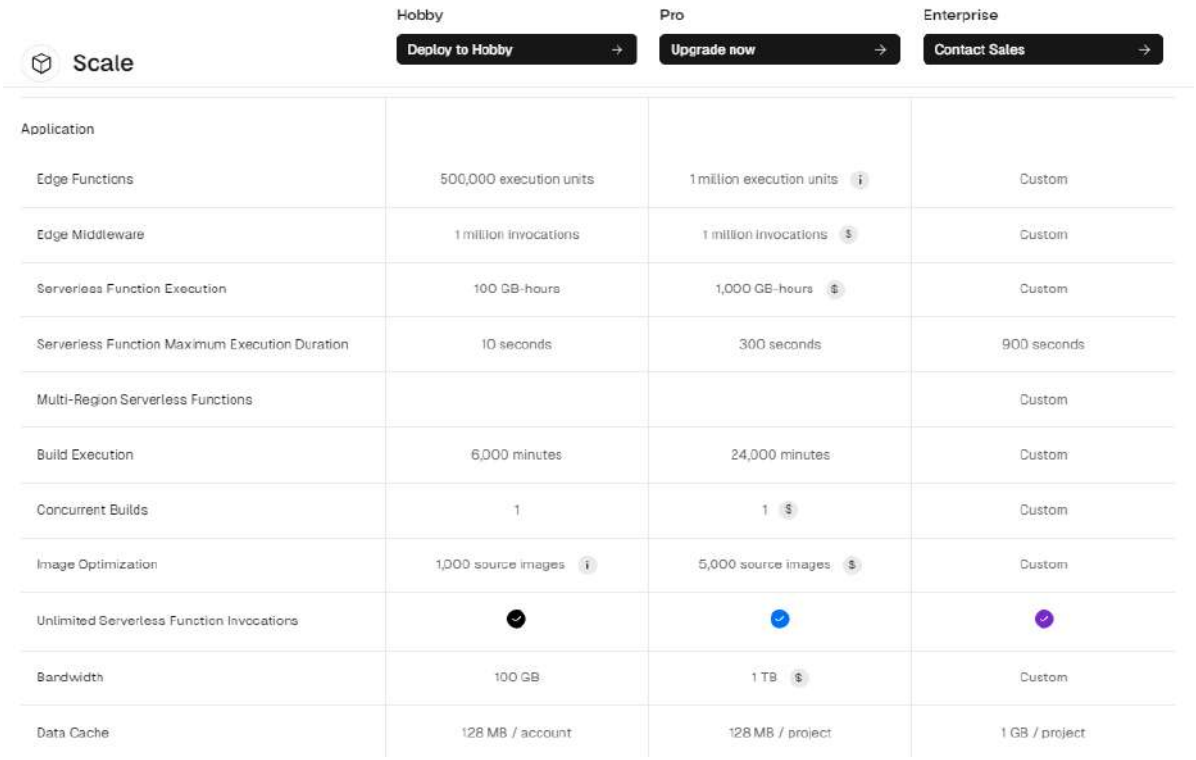
Dentro os serviços e funcionalidade do Firebase, alguns se destacam:

Banco de Dados em Tempo Real: dentro da plataforma há duas modalidades de banco, o Firebase Realtime Database e o Firestore, ambos sendo não relacionais (NoSQL) e de tempo real, permitindo que alterações no banco sejam automaticamente refletidas na aplicação que faz seu consumo e vice-versa.

Serviço de Hospedagem: O Firebase Hosting é o serviço de hospedagem do Firebase para sites e aplicativos web.

[6] <https://vercel.com>
[7] <https://firebase.google.com/?hl=pt-br>

Figura 3.10 – Planos Vercel



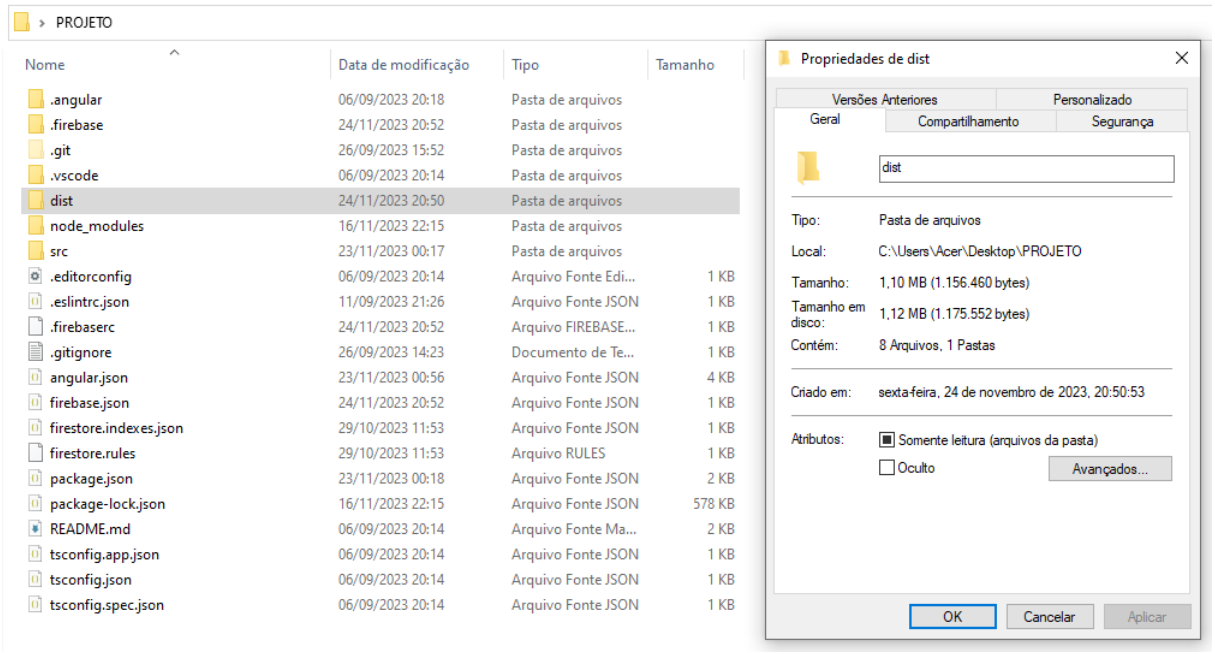
Scale	Hobby Deploy to Hobby →	Pro Upgrade now →	Enterprise Contact Sales →
Application			
Edge Functions	500,000 execution units	1 million execution units	Custom
Edge Middleware	1 million invocations	1 million invocations $	Custom
Serverless Function Execution	100 GB-hours	1,000 GB-hours $	Custom
Serverless Function Maximum Execution Duration	10 seconds	300 seconds	900 seconds
Multi-Region Serverless Functions			Custom
Build Execution	6,000 minutes	24,000 minutes	Custom
Concurrent Builds	1	1 $	Custom
Image Optimization	1,000 source images	5,000 source images $	Custom
Unlimited Serverless Function Invocations	✓	✓	✓
Bandwidth	100 GB	1 TB $	Custom
Data Cache	128 MB / account	128 MB / project	1 GB / project

Fonte: do autor

É importante ressaltar que um dos objetivos do projeto é fazer uso de recursos e ferramentas gratuitas digitais, um dos motivos pelos quais a platarforma Firebase foi considerada uma boa escolha. Outro ponto muito relevante foi a escolha do banco de dados, sabendo que o tipo NoSQL se enquadra melhor no projeto por sua flexibilidade, facilidade de consulta e possíveis incrementos futuros, assim, devido esses fatores, duas das possíveis escolhas seriam o MongoDB e o Banco Firestore fornecido pela Firebase. Sempre com o intuito de tornar o menos complexo possível a construção e consequente reprodução do projeto, mantendo a qualidade da aplicação, o banco escolhido foi o Firestore, já que sua a plataforma além fornecê-lo, também oferece suporte à outros mecanismos necessários ao back-end, tal como hospedagem e autenticação de usuários, para a criação de aplicações web, centralizando uma parte do projeto em apenas um local.

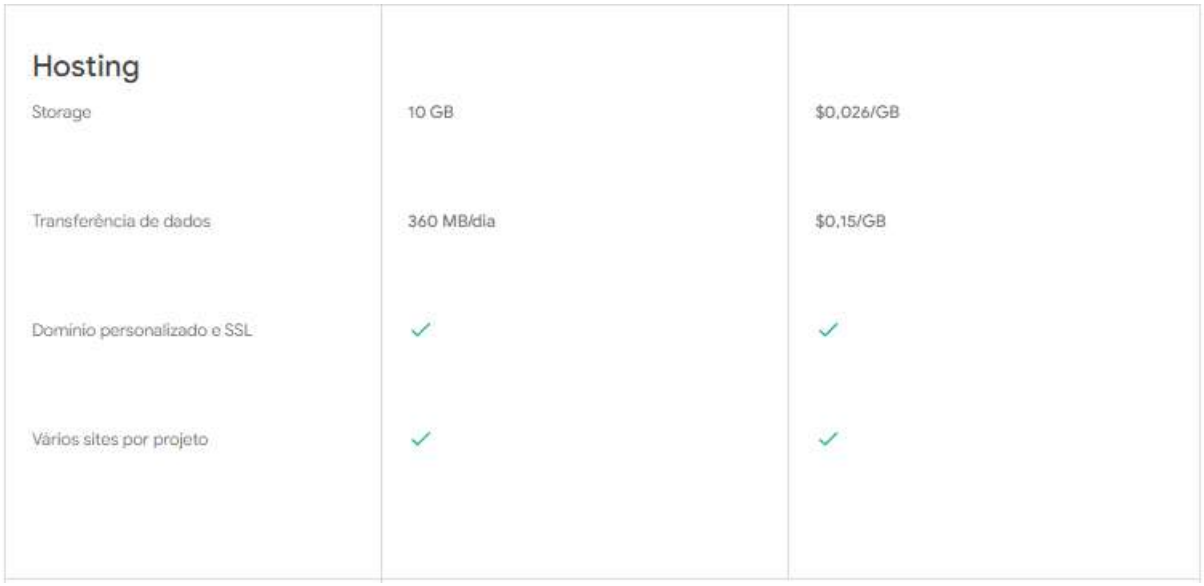
Ainda se tratando dos motivos para escolher o Firebase, também foi considerado as capacidades fornecidas pelo plano gratuito. Como evidencia a Figura 3.11, o tamanho do projeto em versão final tem pouco mais que 1MB, o que é pouquíssimo se comparado aos 10GB de espaço de armazenamento voltados à hospedagem, providos pela plataforma usada, apresentado na Figura 3.12.

Figura 3.11 – Tamanho do arquivo de produção



Fonte: do autor

Figura 3.12 – Recursos de hospedagem firebase plano gratuito

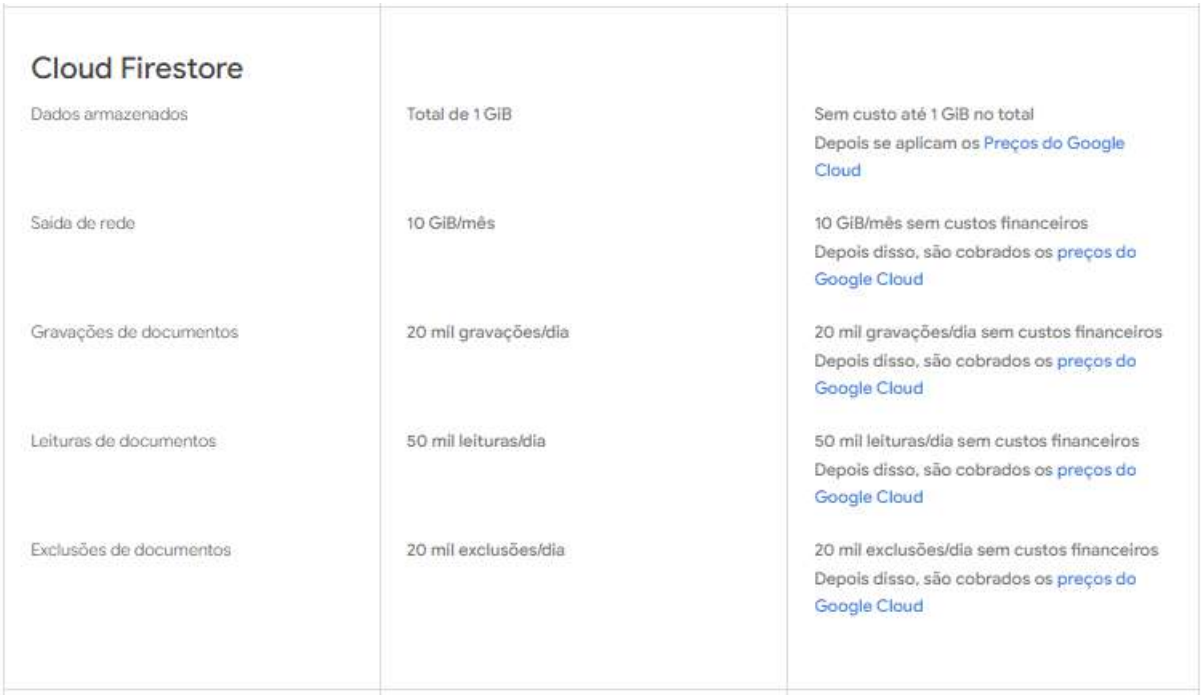


Fonte: do autor

A comparação feita anteriormente também é valida quando se trata de armazenamento de dados no banco. Como pode ser visto nas Figuras 3.13 e 3.14, a quantidade de gravações, leituras e o armazenamento bruto é muito superior ao utilizado, concedendo uma grande margem de segurança tanto para trabalhos profissionais quanto para projetos de teste.

Por fim, é possível ver na primeira coluna da tabela da Figura 3.15 que se refere ao plano gratuito, o volume de usuários que podem se autenticar no serviço, que excede em muito o que

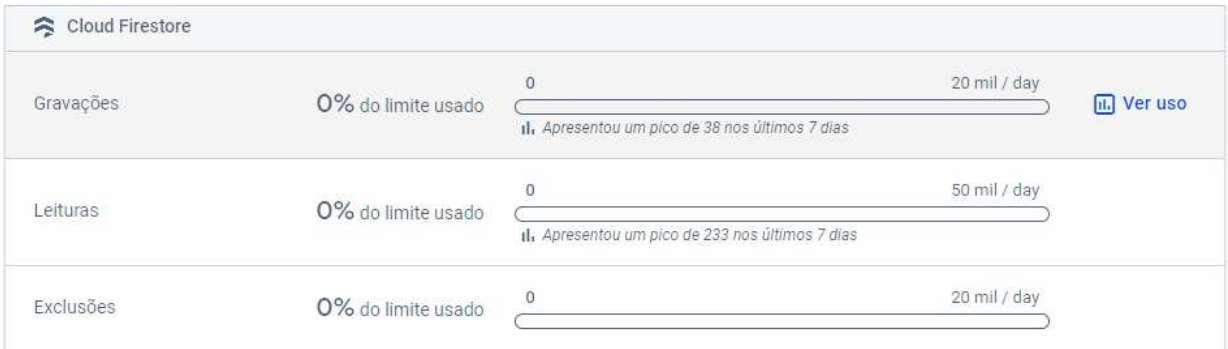
Figura 3.13 – Capacidades Firestore ofertado no plano gratuito



Cloud Firestore		
Dados armazenados	Total de 1 GiB	Sem custo até 1 GiB no total Depois se aplicam os Preços do Google Cloud
Saída de rede	10 GiB/mês	10 GiB/mês sem custos financeiros Depois disso, são cobrados os preços do Google Cloud
Gravações de documentos	20 mil gravações/dia	20 mil gravações/dia sem custos financeiros Depois disso, são cobrados os preços do Google Cloud
Leituras de documentos	50 mil leituras/dia	50 mil leituras/dia sem custos financeiros Depois disso, são cobrados os preços do Google Cloud
Exclusões de documentos	20 mil exclusões/dia	20 mil exclusões/dia sem custos financeiros Depois disso, são cobrados os preços do Google Cloud

Fonte: do autor

Figura 3.14 – Consumo do Firestore pelo projeto



Fonte: do autor

Figura 3.15 – Demonstrativo de recursos para Autenticação

A/B Testing	Sem custos financeiros	
Analytics	Sem custos financeiros	
App Check	No-cost, subject to quotas and limits that vary based on attestation provider.	
App Distribution	Sem custos financeiros	
Autenticação		
Autenticação por telefone: todas as regiões	10 SMS enviados por dia	Cobrado por SMS enviado Consulte as taxas
Outros serviços de autenticação	✓	✓
Com o Identity Platform		
Usuários ativos por mês	50 mil/mês	50 mil MAUs sem custos Em seguida Preços do Google Cloud
Usuários ativos por mês - SAML/OIDC	50/mês	Até 50 MAUs sem custos Em seguida, Preços do Google Cloud

Fonte: do autor

geralmente é necessário para construção de projetos de teste ou mesmo um projeto de médio porte profissional.

3.2.7 Angular

Angular [8] é um framework para construção de aplicativos web dinâmicos e de página única (SPA), desenvolvido e mantido pelo Google. Ele utiliza a linguagem TypeScript, uma superset (um incremento da linguagem) do JavaScript, introduzindo recursos adicionais como tipagem estática e outros recursos de programação orientada a objetos.

3.3 Custos do Projeto

Quando se trata de custos de um projeto tal como o projeto em questão, pode ser divido em custos dos equipamentos de hardware e custos de ferramentas e recursos online, assim como possíveis treinamentos.

Em se tratando dos custos refentes a toda parte digital do projeto o custo foi R$ 0,00, pois todas as ferramentas são gratuitas, tais como github e insomnia. O uso das plataformas de desenvolvimento Firebase e Vercel também não geraram custos, pois no plano gratuito de ambas os recursos fornecidos foram muitas vezes mais que o suficiente para o projeto em questão. Mesmo os treinamentos, como curso de Angular e Node.js também incidiram em custos uma vez que foram todos conteúdos gratuitos consumidos de blogs, sites e canais da internet.

Figura 3.16 – Tabela de Custos dos Componentes

NOME DO DISPOSITIVO	CUSTO UNITÁRIO (R$)	QUANTIDADE	CUSTO (R$)
Motor 25GA-370	72,38	1	72,38
Módulo Relé QJC-3FF-S-Z	18,05	2	36,10
ESP32 38 pinos	58,41	1	58,41
Estrutura em madeira do alimentador	100,00	1	100,00
Protoboard, jumpers e leds	10,00	1	10,00

Fonte: do autor

Quando se trata do projeto físico do protótipo houveram custos, custos esses que podem ser visto na Figura 3.16. É importante ressaltar que sendo o foco a arquitetura, não houve uma intenção de otimizar ao máximo esses custos e por esse motivo o projeto pode ser tornar ainda mais eficiente financeiramente, trazendo menos despesas caso se torne um produto de

[8] <https://angular.io/>

fato. Os valores tabelados têm como referência o segundo semestre de 2023 e podem sofrer alterações com o passar do tempo ou mesmo serem substituídos por tecnologias mais eficientes e de melhor custo-benefício. O custo total para conceber o protótipo foi de R$ 276,89, porém como dito anteriormente há pontos de otimização dos custos dado que o foco foi na arquitetura, além claro, de refletir o custo unitário de produção, custo esse que diminui consideravelmente quando há a produção em escala.

4 DESENVOLVIMENTO E RESULTADOS

Para alcançar o objetivo de fornecer ração ao animal em horas e quantidades determinadas, o protótipo de alimentador pode ser subdividido em 4 partes de desenvolvimento. A divisão e consequente apresentação segue uma ordem lógica que se destina também a servir como guia para o leitor ter maior proveito e facilidade caso deseje implementar o projeto.

Como requisitos, havia-se o desejo de produzir uma arquitetura IoT básica em que fosse possível alimentar animais domésticos à distância controlando o horário e a quantidade de ração a ser fornecida e para tal, utilizar uma interface gráfica para os usuários interagirem com o sistema, com a flexibilidade de ser acessada tanto utilizando um computador quanto um celular.

Conforme mostra a figura 4.1, os dispositivos de interação com o meio, seja para coletar dados ou para atuar no ambiente fazem parte da Camadas das Coisas. Em seguida tem-se a Camada de Adaptação, na qual há um processamento das informações coletadas pela camada anterior ou um processamento para geração de sinais enviados para Camada das Coisas. É na Camada de Adaptação que se encontra o Gateway que no trabalho em questão é o microcontrolador ESP32, cuja função é coletar dados da Camada das Coisas e fornecer meios para que essas informações seja transmitidas para a Camada de Internet, um exemplo é o recurso de WIFI utilizado para fazer essa transmissão. Em se tratando de Camada da Internet no presente documento pode-se entender que essa camada abarca tanto o Servidor Node.js quanto o Banco de Dados Firestore, sendo o servidor considerado como Middleware, fazendo a conexão entre local de armazenamento de dados (banco de dados) e dispositivo IoT (ESP32). Ao final da arquitetura depara-se com a Camada de Aplicação, em que a mesma fornece um meio de interação do usuário com o sistema, utilizando para isso interfaces gráficas, como a aplicação Angular desenvolvida para o projeto.

Figura 4.1 – Arquitetura em Camadas

Fonte: do autor

4.1 Banco de Dados Firestore

O elemento inicial do projeto foi a criação do banco de dados, já que tudo gira em seu entorno, conectando servidor Node.js e Aplicação Angular de forma indireta, fazendo parte do que pode ser considerada a Camada de Intenet juntamente com o servidor Node.js, dado que ambos formam um par de armazenamento, processamento e distribuição de dados. Sendo NoSQL, houve o cuidado para padronizar o tipo de dado cadastrado e os registros criados, uma vez que por sua flexibilidade poderia ser facilmente inserido um campo para receber um valor numérico em vez de uma string, gerando conflito de tipos.

4.1.1 Regras de consumo do banco

O Firestore possui uma camada de segurança opcional que são as Regras, viabilizando a construção de lógicas para o consumo dos dados armazenados. Uma regra muito comum é a liberação total desse consumo, para usuários autenticados, porém, como a criação de regras complexas implica em uma maior complexidade na conexão entre o Firebase e a aplicação Angular ou mesmo o servidor e o Firebase, a regra aplicada libera acesso livre a qualquer pessoa que tenha conhecimento da URL utilizada para acessar o banco como mostra a figura 4.2. É possível observar que qualquer pessoa terá as permissões de leitura e escrita de todos os documentos do banco Firestore, mas como se trata de um protótipo e apenas o autor conhece essa URL não houve a necessidade de tornar o projeto mais complexo. É importante salientar que apesar dessa regra dar livre acesso aos dados, não diz respeito à autenticação de usuário em relação à aplicação, diz respeito ao consumo do banco em específico.

Figura 4.2 – Regra Firestore utilizada

```
1  rules_version = '2';
2  service cloud.firestore {
3    match /databases/{database}/documents {
4      match /{document=**} {
5        allow read, write;
6      }
7    }
8  }
```

Fonte: do autor

4.2 Desenvolvimento de Servidor Node.js

Quando se trata do desenvolvimento de um servidor, é possível que o mesmo seja desenvolvido para atender requisições HTTP e HTTPS, sendo que a maior diferença está na utilização

ou não de licença, licença essa que se trata de um arquivo emitido apenas por autoridades certificadoras, tal como a Let's Encrypt. A escolha por desenvolver o servidor para atender requisições HTTP (sem licença) se deve à maior simplicidade tanto para desenvolver a API (servidor), quanto para que o ESP32 fizesse requisições HTTP, que também é algo menos complexo em sua codificação se comparado ao HTTPS.

Para atender tanto o projeto, quanto para uma boa base de estudos para o leitor, utilizou-se o framework Express para a codificação do servidor dentro do Node.js, como mostra a figura 4.3. Esse framework abstrai em si configurações de um grau de complexidade considerável, tal como acesso e configuração de rotas, que é o principal recurso utilizado do mesmo, possibilitando criar de forma mais simples uma requisição no formato evidenciado na Figura 4.4, no padrão "metodo (endpoint, corpoDaRequisicao, respostaDaRequisicao)".

Figura 4.3 – Configurações iniciais do Servidor

Fonte: do autor

A figura 4.3 apresenta o trecho inicial da programação da API mostrando como é feita a configuração tanto de criação dos recursos de roteamento quanto da conexão com o firebase, que necessita de um arquivo contento as chaves para essa conexão. Algo muito importante, foi colocar o projeto do servidor em um diretório público do github inicialmente, pois ao fazer isso qualquer pessoa tem acesso as configurações principais da conexão e pode se beneficiar disso, para fazer um ataque profissonal ou mesmo apenas causar transtorno sem um objetivo específico. Para notificar os pontos de vulnerabilidade tanto a plataforma Firebase notifica o ocorrido e começa a barrar requisições como mostra a figura 4.5, devido à ausência de regras em união a detecção de que há uma chave de configuração privada exposta na internet, quanto a Google envia um email de notificação relatando o problema, conforme a figura 4.6.

Figura 4.4 – Rota Node.js para aquisição de registros

```
router.get('/verificaRegistros', (request, response) => {
    const docConfigs = db.collection('configsMicro').doc('IWCPZ9JYSrccJptETwtL');
    const docRegistros = db.collection('registros').doc('lho9vSUSO55I6GOG3l5A');

    docConfigs.get()
    .then((docConf) => {
        if (docConf.exists) {
            const atualizado = docConf.data()['registrosMicroAtualizado'];

            if(!atualizado){
                docRegistros.get()
                .then((docReg) => {
                    if (docReg.exists) {
                        const data = docReg.data();
                        response.status(200).json(data);
                    } else {
                        response.status(404).json({ erro: 'Documento inexistente' });
                    }
                })
            }else{
                response.status(204).json({});
            }
        } else {
            response.status(400).json({ erro: 'Coleção inexistente' });
        }
    })
    .catch((error) => {
        const endpoint = request.originalUrl;
        response.status(500).json({ erro: `Falha ao processar a requisição ${endpoint}. Erro: ${error}` });
    });
});
```

Fonte: do autor

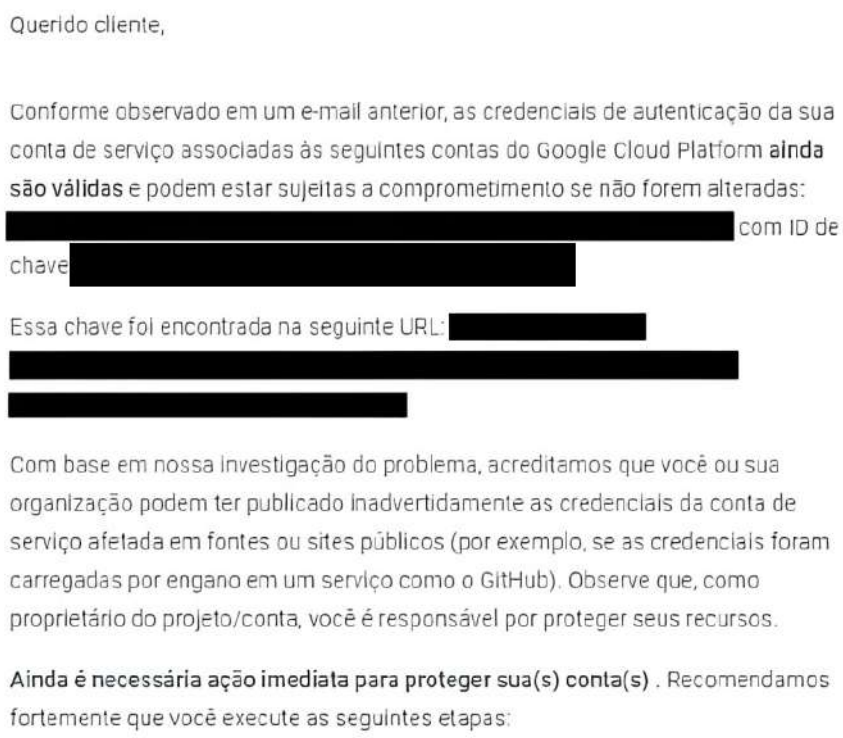
Figura 4.5 – Notificação de requisições barradas

Fonte: do autor

Para corrigir o problema bastou adicionar uma regra ao banco e converter o diretório público que detêm o servidor, em diretório privado, assim, apenas os autores do projeto que nesse caso é apenas o autor desse trabalho tem acesso ao mesmo. Tal ocorrido é extremamente interessante por dois motivos:

- A confirmação de camada de segurança da própria plataforma Firebase, mesmo sem a configuração para essa segurança em específico.

Figura 4.6 – Email de alerta Google


Querido cliente,

Conforme observado em um e-mail anterior, as credenciais de autenticação da sua conta de serviço associadas às seguintes contas do Google Cloud Platform **ainda são válidas** e podem estar sujeitas a comprometimento se não forem alteradas:
[redacted] com ID de chave [redacted]

Essa chave foi encontrada na seguinte URL: [redacted]

Com base em nossa investigação do problema, acreditamos que você ou sua organização podem ter publicado inadvertidamente as credenciais da conta de serviço afetada em fontes ou sites públicos (por exemplo, se as credenciais foram carregadas por engano em um serviço como o GitHub). Observe que, como proprietário do projeto/conta, você é responsável por proteger seus recursos.

Ainda é necessária ação imediata para proteger sua(s) conta(s) . Recomendamos fortemente que você execute as seguintes etapas:

Fonte: do autor

- As notificações e comunicação do erro pelo Firebase e pela Google, evidenciando que há uma tranquilidade em desenvolver utilizando esses serviços, pois os desenvolvedores são alertados, para que não fiquem sem um norte de por qual razão houve um ataque, algum incidente ou mal funcionamento do projeto.

Como mostra a figura 4.3 e evidenciado na figura 4.4, a configuração feita para essa API permite que os métodos de requisição HTTP GET, PATCH, PUT, POST e DELETE sejam aceitos de todas as origens e sem a necessidade de restringir os cabeçalhos, dando a flexibilidade que um projeto protótipo precisa. Os métodos utilizados no projeto foram GET e PUT, já que o ESP32 necessitava dos registros existentes para fazer o controle da vazão de ração e o método PUT para atualizar a quantidade de ração presente no reservatório. Na figura 4.7 são mostrados os *endpoints* utilizados para alcançar o objetivo do projeto.

Em vez de testar as rotas utilizando o navegador, utilizou-se o software gratuito Insomnia. Com o Insomnia testou-se as rotas e seus diferentes métodos, o que não seria possível com facilidade pelo navegador uma vez que apenas o método GET é simples ao ponto de usar uma URL sem a necessidade de um cabeçalho ou corpo da mensagem de forma explícita. Na

figura 4.8 é possível ver como é feito o método PUT e sua respectiva resposta, sendo importante observar que é necessário passar um objeto Json como corpo da requisição.

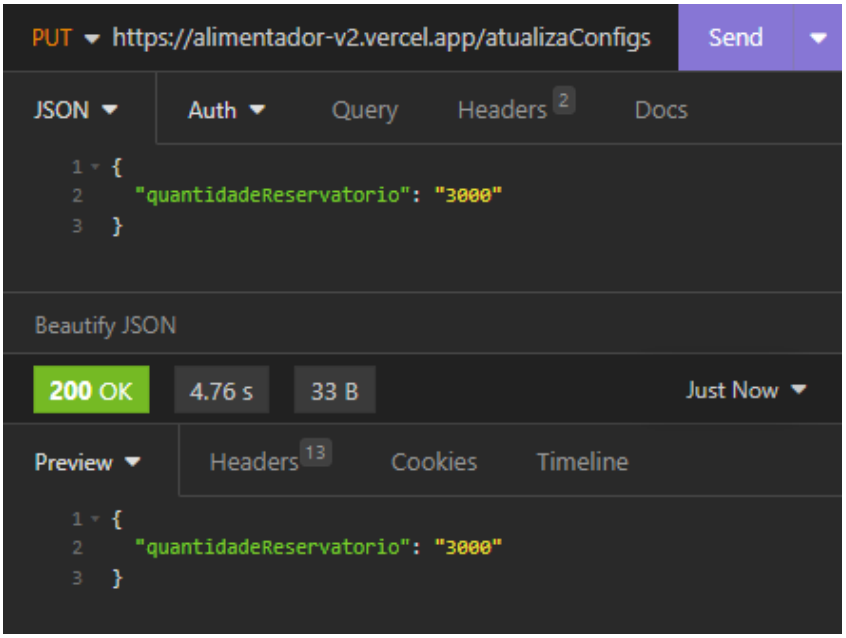
Objetivando a reprodução por interessados, foi disponibilizado o acesso ao projeto do servidor Node.js criado por meio do link https://github.com/willian2022a/Projeto-Servidor.

Figura 4.7 – Tabela de Endpoints

Método Http	Endpoint	Descrição
PUT	/atualizarConfigs	Atualiza a quantidade de ração do reservatório
GET	/reservatorio	Obtém a quantidade de ração do reservatório
GET	/verificaRegistros	Obtém uma carga de dados contendo todos os registros de hora e quantidade cadastradas

Fonte: do autor

Figura 4.8 – Requisição PUT com Insomnia



Fonte: do autor

4.3 Desenvolvimento de Aplicação Angular

No contexto de Camada de Aplicação, o framework Angular se prestou muito bem ao objetivo de criação de uma interface para visualização e interação com dados, porém, há de se pontuar que o aprendizado de tal ferramenta não é algo fácil ou trivial, sendo de fato a parte do projeto que exigiu maior dedicação do autor pelo nível de detalhe e a obrigatoriedade de um bom grau de domínio sobre a ferramenta. No início do projeto houve o desejo de fazer

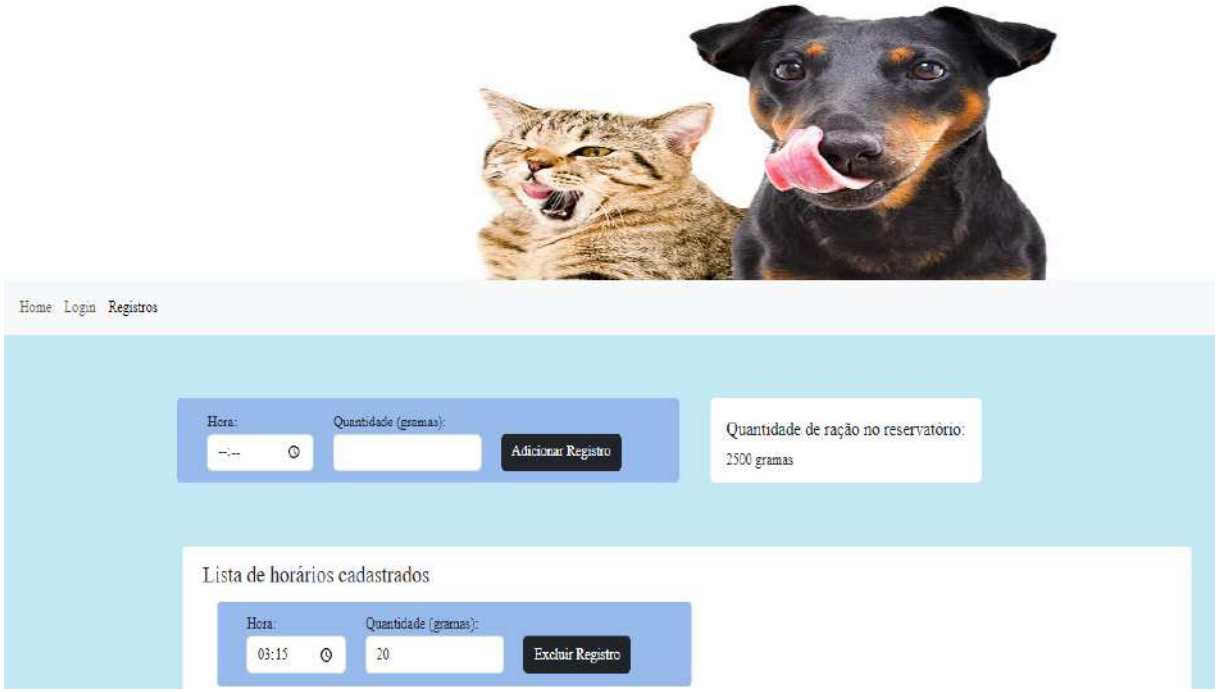
a interface utilizando React, outro framework muito em alta no momento da confecção do presente trabalho, porém como contribuição pessoal, vinda da atuação no mercado de trabalho por parte do autor, surgiu a notícia de que em um futuro próximo os desenvolvedores deveriam dominar Angular na empresa onde trabalha o autor, pois passaria a ser utilizado, assim, ocorreu a migração para o framework Angular, confirmando a aplicação prática do mesmo no mercado de trabalho.

Para melhor entendimento e reprodução do projeto por aqueles que desejarem, foi disponibilizado um repositório público contendo o projeto Angular produzido que pode ser acessado pelo link https://github.com/willian2022a/Projeto-Alimentador.

4.3.1 Autenticação Firebase

Um dos pontos mais importantes da aplicação Angular foi a criação de um mecanismo para que apenas usuários com permissão pudessem acessar as páginas que não fossem apenas a página de *login*, protegendo assim principalmente a página de manipulação dos registros, mostrada nas figuras 4.9 e 4.10.

Figura 4.9 – Página de registros - Menu visível



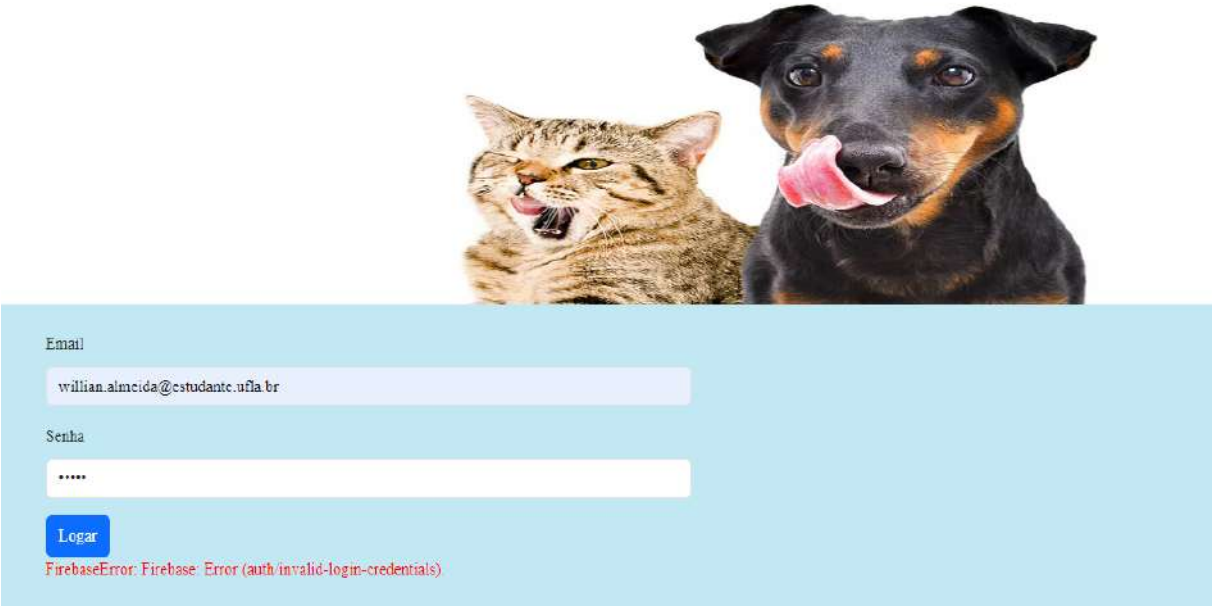
Fonte: do autor

Figura 4.10 – Página de registros

Fonte: do autor

Esse mecanismo foi a união do recurso de autenticação Firebase com o uso de uma variável interna no projeto que indica se o usuário está autenticado ou não, limitando a atuação do mesmo sobre a aplicação e mantendo esse usuário desconhecido na página de *login*, alertando que suas credenciais estão incorretas, como mostra a figura 4.11.

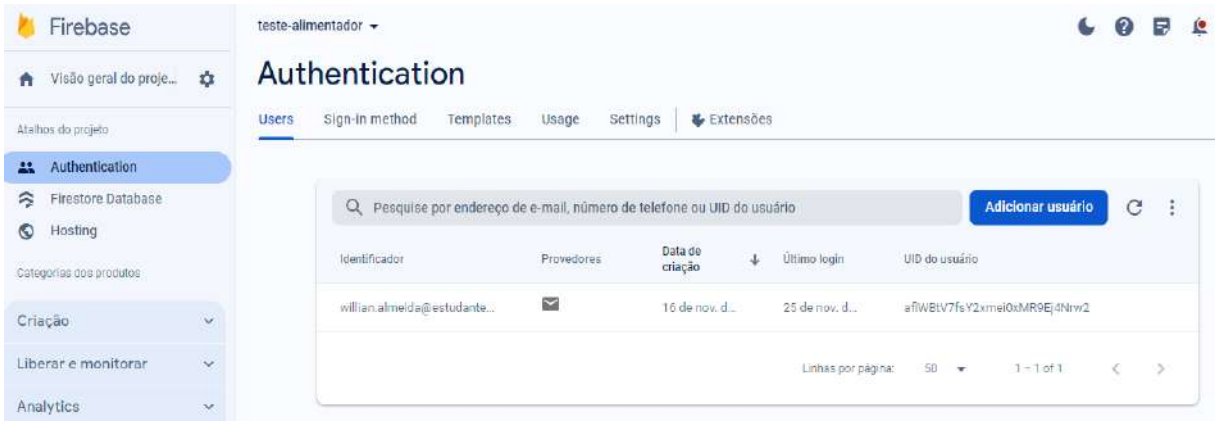
Figura 4.11 – Tentativa de Login - Usuário sem permissão



Fonte: do autor

A autenticação Firebase empregada foi por meio de email e senha, em que é preciso cadastrar emails e suas respectivas senhas como mostra a figura 4.12, para posteriormente criar a vinculação entre Firebase e projeto Angular. Essa vinculação pode ser feita baseando-se na documentação oficial do Firebase e em uma série de configurações que o projeto Firebase disponibiliza.

Figura 4.12 – Cadastro de usuários válidos na aplicação



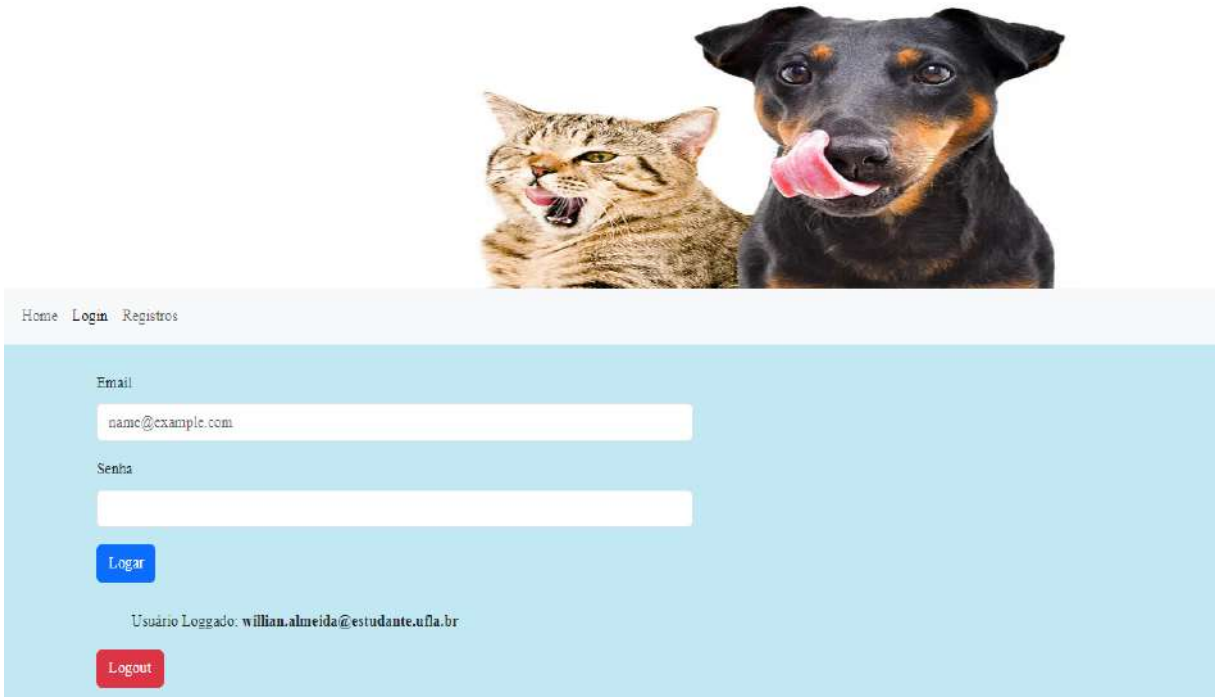
Fonte: do autor

Caso o *login* esteja correto o usuário é direcionado para a página Home, apresentando ao mesmo um breve resumo do funcionamento do projeto, como mostra a figura 4.13. Caso haja uma necessidade de verificar qual o usuário que está logado, basta ir na página de *login*, como mostra a figura 4.14.

Figura 4.13 – Página Home do site

Fonte: do autor

Figura 4.14 – Página de Login - usuário já logado



Fonte: do autor

4.3.2 Utilização de Observáveis

Para que houvesse a restrição de acesso a páginas de acordo com a variável de autenticação interna foi utilizado o conceito de Observáveis *(Observables)*. Os Observáveis são parte da biblioteca *RxJS*[1] , frequentemente utilizada no contexto do Angular para programação reativa, o que significa lidar com eventos de forma assíncrona, contrastando com o comportamento padrão do JavaScript, que segue o modelo de execução *single-threaded* com um loop de eventos *(event loop)*.

Não somente para autenticação foi utilizada essa solução, como também para transmitir as atualizações assíncronas dos registros e da quantidade de ração no reservatório, vindas do banco Firestore que possui o serviço de "Real Updates", em que sempre que há uma alteração dos dados no banco, automaticamente é enviada uma carga para a aplicação contendo o conteúdo atualizado. Mesmo que essa carga de dados seja enviada para a aplicação isso não garante a atualização na mesma, por isso é necessário usar os Observáveis, pois estarão sempre "escutando" as alterações enviadas.

4.3.3 Design Responsivo Bootstrap

Um dos aspectos mais importantes principalmente quando se trata de um produto/serviço profissional é o desenvolvimento de interfaces responsivas, o que significa dizer que há um conjunto de ajustes de tela para que uma mesma aplicação se enquadre bem em telas grandes, médias ou pequenas, sem haver o chamado "estouro de tela" que nada mais é do que um elemento passar os limites da tela ou das proporções que lhe cabem mediante às dimensões do display e os outros elementos que compõe o front-end da página. Para alcançar essa responsividade, foi utilizado elementos e classes próprias do Bootstrap que fazem a abstração de Medias Query, que são esses ajustes de tela, assim não foi necessário fazer do zero essas adaptações que são muito laboriosas, mas foi necessário ter o entendimento do funcionamento para utilização dessas classes.

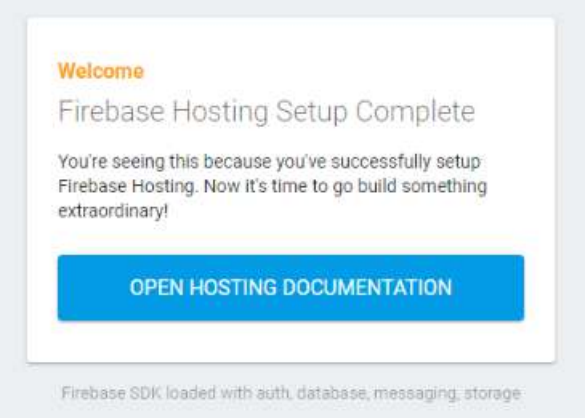
4.3.4 Hospedagem

Para fazer a hospedagem foi necessário gerar o *build* de produção, que é um diretório que contém os arquivos compilados e otimizados, apenas com os recursos necessários para o

[1] <https://rxjs.dev/>

site online(implantado), retirando tudo que foi usado para auxiliar apenas no ambiente de desenvolvimento. Esse momento exigiu uma atenção especial, já que uma das opções no momento da implantação de arquivos do Firebase no projeto Angular, pode sobrescrever o arquivo index.html correto, colocando o arquivo index.html padrão de apresentação do Firebase, fazendo com que o projeto seja implantado, mas a aplicação não consiga inicializar corretamente, como demonstra um exemplo desse erro na figura 4.15.

Figura 4.15 – Tela padrão de hospedagem Incorreta



Fonte: do autor

4.4 Construção e controle do protótipo físico

Nesta seção será discorrido como foi e quais os desafios encontrados no decorrer do desenvolvimento da parte física do protótipo e seu respectivo controle através da programação do microcontrolador ESP32.

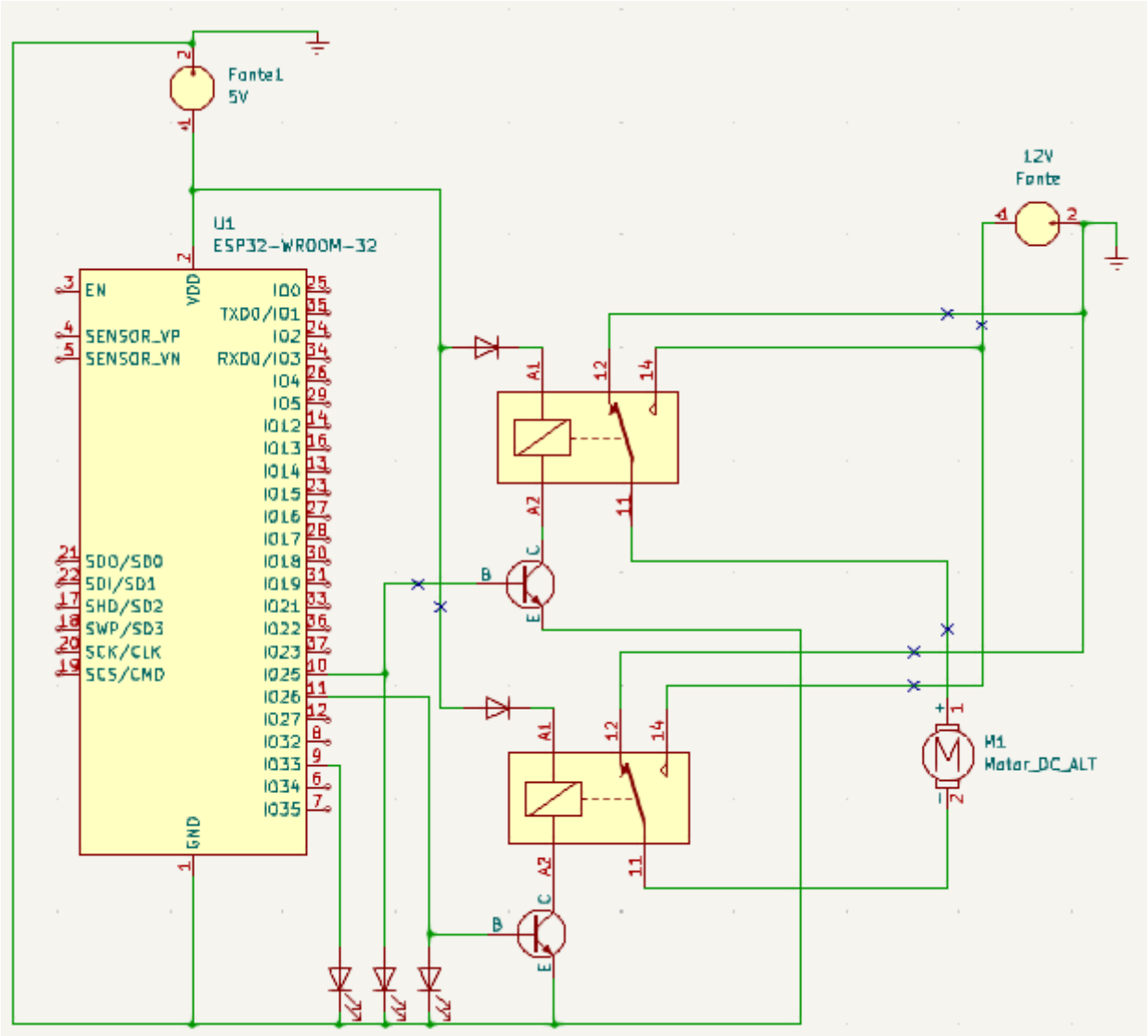
4.4.1 Esquema elétrico do projeto

Quando se trata do acionamento do motor foi necessário utilizar uma fonte de tensão de 12V 1A. Essa corrente é suficiente para alimentar tanto o motor quanto o ESP32. A intenção inicial era ligar o motor, o microcontrolador e os relés que controlam o acionamento do motor, todos em um mesmo terra, utilizando a fonte de 12V junto ao regulador de tensão, enviando 12V para o motor e 5V do regulador para os relés e para o Esp, porém ao conectar todos os componentes juntos surgiu um desafio, relacionado ao suprimento de energia que será discorrido mais adiante.

O esquema elétrico gerado no software KiCad para o acionamento do motor e alimentação do microcontrolador pode ser visto na Figura 4.16. Para o acionamento e giro do motor

foram utilizados dois módulos relés, sendo que o ESP32 controla o momento em que estes relés serão acionados e a fonte de 12V fica exclusivamente para suprir a corrente que o motor necessita.

Figura 4.16 – Esquema elétrico do projeto



Fonte: do autor

4.4.2 Desafios na montagem da parte física

Para estimar quanto de ração caia por segundo para ter a vazão do alimentador e mediante a isso calcular o tempo de aberto, fechamento e pausa do motor, calculou-se a vazão mássica do sistema.

Essa vazão mássica é dada, dada pela expressão:

$$\dot{m} = \rho \cdot \dot{V}$$

Em que $\dot{V}$ é a vazão volumétrica, podendo também ser expressa como produto da área de escoamento pela razão entre o comprimento do orifício por onde sai a ração e o tempo de queda. Como mostra expressão:

$$\dot{V} = \frac{A \cdot l}{t}$$

Sendo a A a área circular por onde cai a ração, com 4cm de diâmetro e $\frac{l}{t}$ a velocidade média de queda da ração, foi necessário então calcular essa velocidade, considerando que a ração está em queda livre, percorrendo 2cm (comprimento do orifício do alimentador), tem-se o seguinte desenvolvimento MRUV:

$$y = y0 + v0 \cdot t + \frac{1}{2} g \cdot t^2$$

$$y - y0 = \frac{1}{2} g \cdot t^2$$

$$t = \sqrt{\frac{2 \cdot l}{g}}$$

Assim, a expressão da velocidade média de queda da ração e vazão volumétrica desse sólido são dadas pelas seguintes equações respectivamente:

$$Vmedia = \frac{l}{t}$$

$$\dot{V} = A \cdot Vmedia$$

A expressão rearranjada para vazão mássica é:

$$\dot{m} = \rho \cdot A \cdot Vmedia$$

Ao fazer uma busca por fontes que contivessem a densidade de ração para animais domésticos, principalmente cães e gatos nada foi encontrado, por esse motivo foi adotado um tamanho médio de um grão de ração pequeno e sua massa para que ao fazer o cálculo de vazão

houvesse uma precisão maior, já que grãos menores implicam em mais grãos por segundos, permitindo um cálculo mais próximo do ideal. Sabendo que 27 grãos de ração Especial Cat Prime, que foi usada para testes, pesa 1g e tem um diâmetro próximo a 7,5mm, foi possível calcular sua densidade, chegando ao resultado de $167.67 Kg/m^3$. Com os dados obtidos até então, constatou-se que a vazão de ração é aproximadamente $20 g/s$, valor que pode variar de acordo com densidade ração selecionada, assim, caso fosse necessário manter a mesma linha de projeto podendo variar o tamanho da ração, poderia-se adicionar um campo no banco de dados para descriminar se é uma ração para animais muito pequenos, pequenos, médios ou grandes e a partir disso ter na programação do microcontrolador uma adaptação dos parâmetros relativos ao escoamento de cada ração em relação ao porte do animal que irá consumir.

Outro desafio a ser solucionado foi com a escolha inicial dos módulos relés utilizados já que os mesmos necessitam de uma corrente maior do que o ESP32 consegue fornecer, assim, foi necessário buscar por um outro módulo que satisfizesse o projeto, por isso o módulo relé utilizado também de 5V 1 canal foi o QJC-3FF-S-Z.

Muito próximo de finalizar de fato a montagem física do protótipo unindo o microcontrolador já programado ao conjunto de relés e motor, observou-se que ao colocar a fonte de tensão ajustável para alimentar o ESP32 já com toda a programação, os LEDS usados como sinalizadores piscavam repetidamente, porém, mesmo mudando o GND o comportamento incorreto permanência, logo, o segundo passo foi começar a comparar o funcionamento do microcontrolador utilizando o cabo USB em relação a alimentação usando a fonte, com isso, percebeu-se essa diferença em que com o cabo o comportamento estava normal e com a fonte estava disfuncional, mesmo inserindo a correta tensão no Esp. Como já se havia uma suspeita prévia, foram feitos testes inserindo e executando apenas trechos do código completo, assim, foi possível constatar que ao tentar fazer a conexão com o WIFI o microcontrolador acaba utilizando mais energia que a corrente fornecida pela fonte e então havia esse reset forçado, por isso o circuito sempre piscava os LEDS como se estivesse em um loop de executar apenas o código da função setup do microcontrolador. Mediante a essa situação decidiu-se utilizar a fonte apenas para alimentar o motor, deixando a alimentação dos LEDS e relés a cargo da tensão fornecida pelo Esp, estando esse último conectado via usb ao computador.

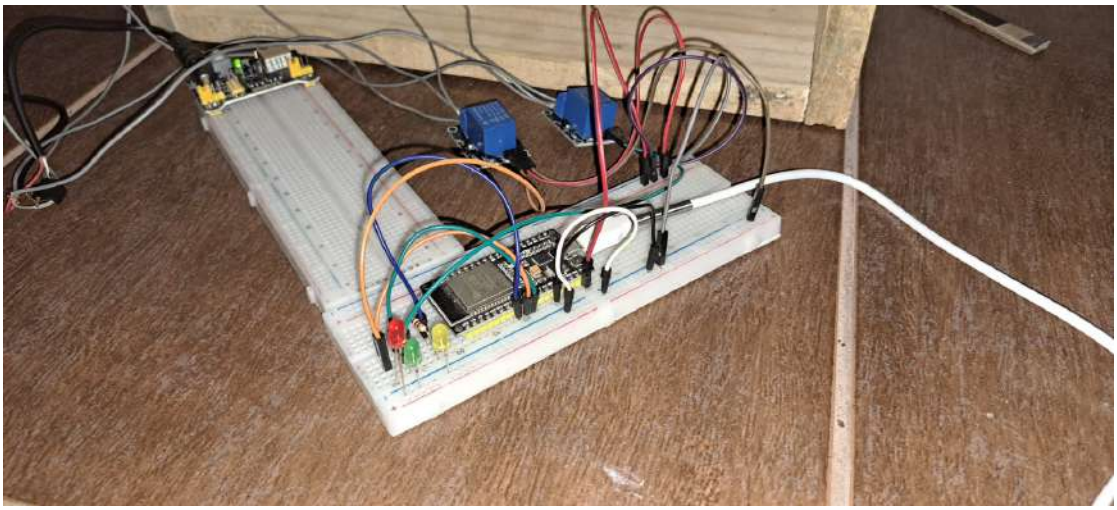
Nas figuras que se seguem é possível ver *prints* do vídeo de demonstração do funcionamento do sistema completo. Na figura 4.17 é mostrada uma visão de cima, o alimentador com pouca ração justamente para evidenciar o diâmetro e o local por onde essa ração irá escoar.

Para conseguir sinalizar em que momento o motor estava em giro de aberto, fechamento e para sinalizar quando a condição de hora cadastrada condiz com a hora atual, foram usados 3 LEDS, nos quais o amarelo indicava que a condição de hora foi satisfeita, o verde sinalizando abertura da vazão e o vermelho o para fechamento da vazão, como mostra as figuras 4.17, 4.18 e 4.19.

Figura 4.17 – Protótipo físico do alimentador - Visão de cima

Fonte: do autor

Figura 4.18 – Protótipo físico do alimentador - Visão do circuito



Fonte: do autor

Figura 4.19 – Protótipo físico do alimentador - Visão total

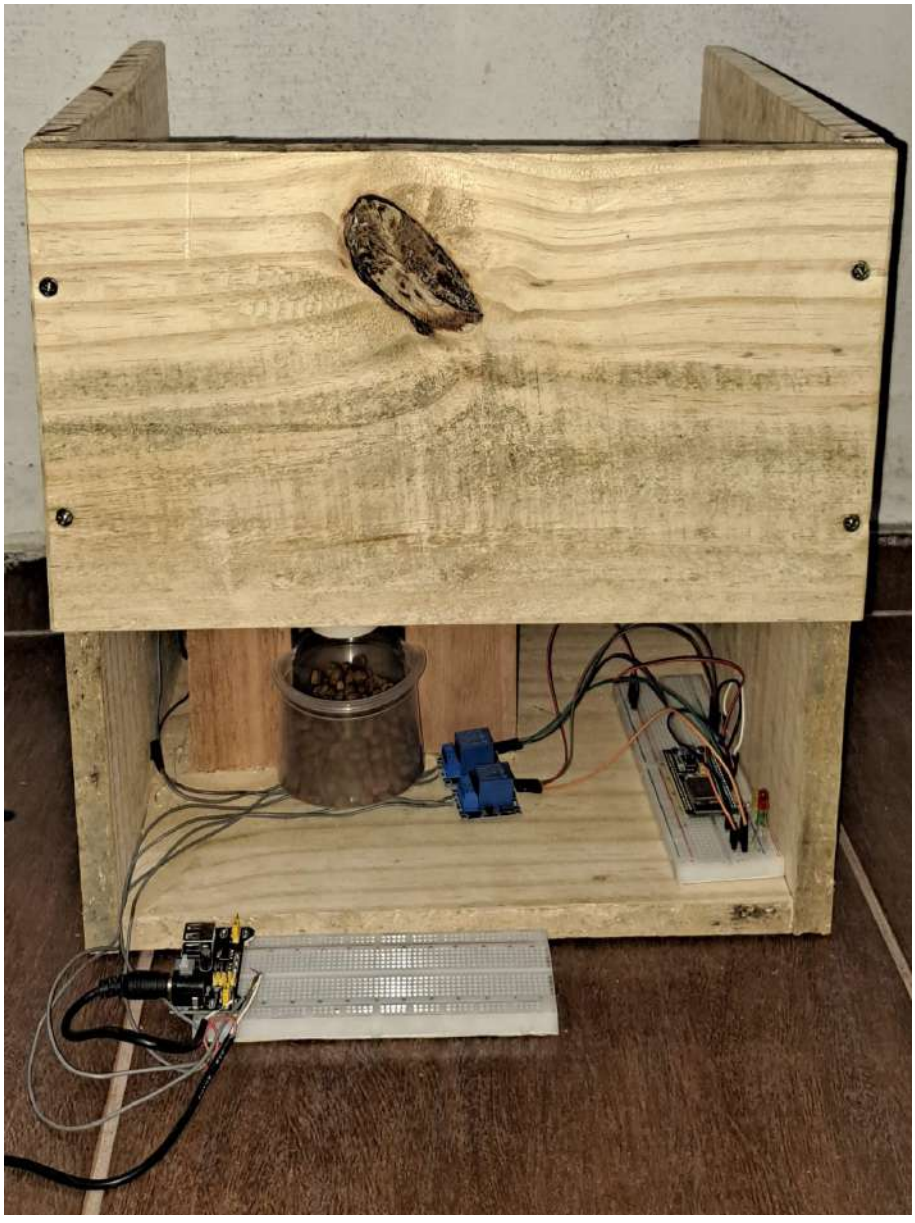
Fonte: do autor

Nas figuras 4.20 e 4.21 é possível ver o motor na posição vertical e em seu eixo uma tampa de material PVC. Esse PVC tampa o orifício por onde vaza a ração, logo, para que esse orifício seja aberto o motor gira no sentido horário e para cessar o escoamento dessa ração o motor gira em sentido anti-horário.

Figura 4.20 – Alimentador visto de frente - visão mais próxima

Fonte: do autor

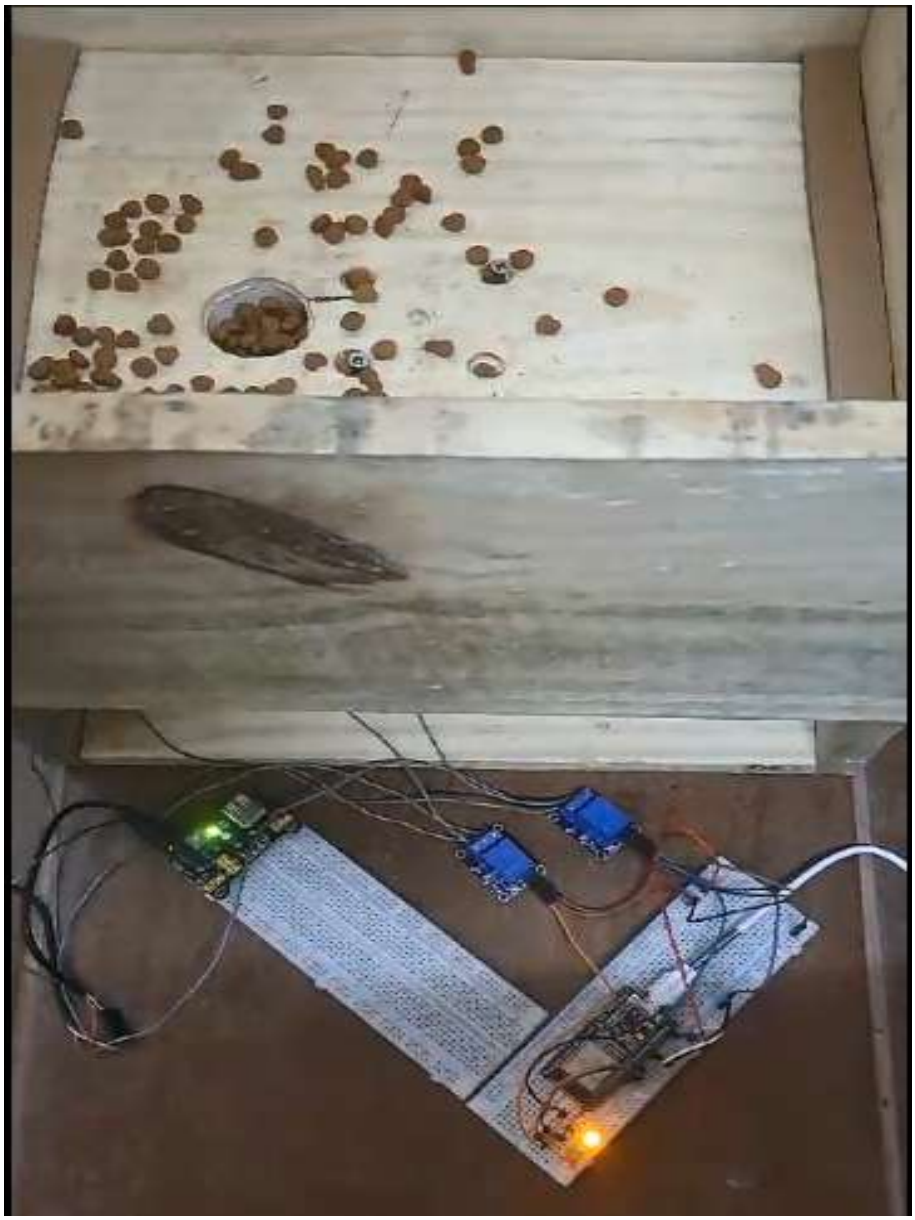
Figura 4.21 – Alimentador visto de frente - visão mais afastada



Fonte: do autor

Nas figuras 4.22, 4.23, 4.24, 4.25, há a demonstração de como é usada a sinalização luminosa, sendo que em sequência cronológica há primeiramente a indicação de que a condição de hora foi satisfeita, em seguida o motor é acionado em abertura, como evidencia o LED verde, após isso há apenas o LED amarelo novamente mostrando que há um tempo em que alimentador fica aberto para escoamento e por fim há o acender do LED vermelho simbolizando o fechamento que pode ser visto também no próprio orifício de escoamento que fica tampado ao final do processo.

Figura 4.22 – Sinalização de condicão de hora satisfeita



Fonte: do autor

Figura 4.23 – Abertura do escoamento - usando led verde para simbolizar

Fonte: do autor

Figura 4.24 – Protótipo em livre escoamento

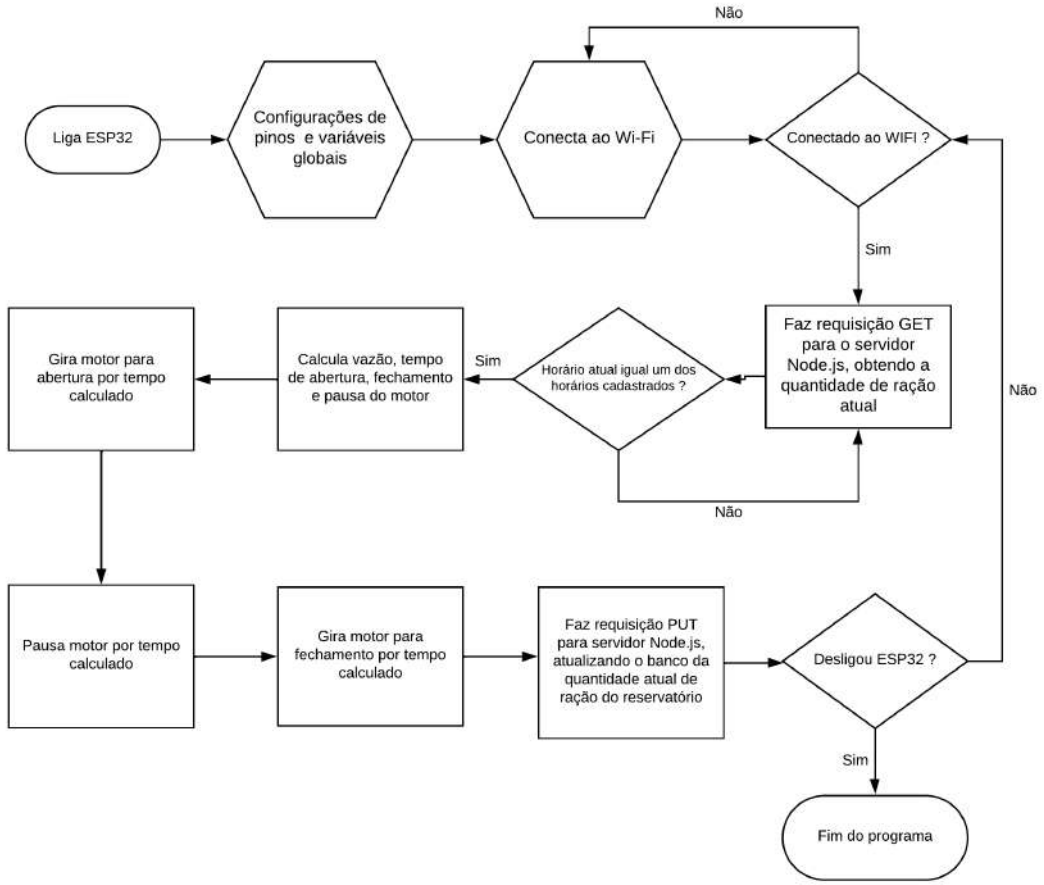
Fonte: do autor

Figura 4.25 – Fechamento do escoamento - usando led vermelho para simbolizar

Fonte: do autor

4.4.3 Programação do ESP32

Figura 4.26 – Diagrama de blocos da programação ESP32



Fonte: do autor

Ao programar o ESP32 há uma sequência lógica a ser seguida para que seja alcançado o objetivo de liberar alimento ao animal, apenas nas quantidade e horários cadastrados. A figura 4.26 apresenta o fluxograma da programação codificada no microcontrolador, na qual há sempre a verificação de conexão com Wifi ativa. Caso essa conexão estiver ativa, a instrução é ficar constantemente fazendo requisições GET para o servidor Node.js, para que este retorne os registros cadastrados no banco de dados Firestore. De posse dos registros, o ESP32 irá varrer esses registros a fim de encontrar algum registro cujo horário seja igual ao horário atual obtido pelo ESP32 via consumo da API que pode ser acessar pelo link http://worldtimeapi.org/api. Como visto no bloco de decisão, caso essa condição de horários correspondentes seja satisfeito, é feito uma série de cálculos para determinar como o motor de ser acionado para fornecer a ração determinada pelo registro em questão. Entre esses cálculos está o tempo necessário, sabendo que a vazão do sistema é de 20g/s. Ao final do processo é feita uma requisição com o método

PUT para o servidor, passando como corpo da mensagem a nova quantidade de ração existente no reservatório, assim o servidor irá receber e atualizar o banco inserindo essa nova quantidade.

Todo o processo descrito acima pode ser visto por meio do monitor serial, como consta na Figura 4.27.

Figura 4.27 – Monitor serial durante um controle completo do processo de escoamento da ração

```
Conectando ao WiFi...
Conectando ao WiFi...
  SSID: GIL
  Senha: 48622684
Conectado ao WiFi
Endereço IP: 192.168.0.101
Quantidade Atual: 5210
================================================
Hora: 19
Minuto: 35
Segundo: 14
Payload recebido: {"1":{"quantidade":"100","hora":"19:35"},"2":{"hora":
ABERTURA::: 490
DELAY 4000
FECHAMENTO::: 505
Rele acionado para ABERTURA
Rele acionado para FECHAMENTO
NOVA:::: 5110
Meu valor float: 5110
Requisição PUT bem-sucedida 2
Resposta do servidor: {"quantidadeReservatorio":"5110"}
PARADA:::: 45000
================================================
Hora: 19
Minuto: 36
Segundo: 23
Payload recebido: {"1":{"quantidade":"100","hora":"19:35"},"2":{"hora":
================================================
```

Fonte: do autor

5 CONCLUSÃO

O objetivo deste projeto foi construir o alimentador e demonstrar a eficácia da arquitetura que se pretendia fazer, utilizando tecnologias em alta, tal como o microcontrolador Esp32 que já vem com módulo Wifi embutido; Banco de dados através do serviço Firebase; Interface responsiva e agradável usando um *front-end* mais elaborado com o framework Angular e Desenvolvimento de um servidor Node.js, para transferir os dados para o microcontrolador.

Em se tratando da programação do ESP32, o maior desafio foi em encontrar conteúdo sobre requisições HTTP para ESP32, sendo necessário tentar entender não somente os conteúdos contidos em blogs, fóruns, canais ou sites, como também na documentação oficial da biblioteca usada para fazer tais requisições.

Outro ponto de desafio foi a busca de uma maneira de como saber a vazão, para poder controlá-la. A solução encontrada foi através do cálculo de vazão mássica, utilizando os conhecimentos físicos adquiridos nas disciplinas do curso.

Com todos os desafios, experiência de mercado, conhecimentos adquiridos durante o curso e previsões feitas ao iniciar o projeto, constatou-se que cada vez mais as áreas afins mesclam-se, havendo então uma necessidade dos estudantes e profissionais da área de automação em expandirem seus horizontes. Com o decorrer de todo o trabalho foi percebido o quanto importante é ter um banco de dados bem estruturado, pois se comparado ao armazenamento físico dos microcontroladores é extramente superior, dado a limitação física imposta por esse tipo de hardware. Assim como a limitação de processamento desse equipamento se comparado aos servidores na nuvem que além de terem um processamento muitas vezes mais potente, há também a facilidade de trocar de plano. Atualmente há muitos planos que funcionam sob demanda, adaptando aos requisitos correntes do projeto, assim, a fatura é cobrada pelo consumo.

Ao fazer uso de um microcontrolador offline limita-se a aplicabilidade do mesmo, então para que o profissional não se veja barrado pelo hardware em um mundo cada vez mais digitalizado e conectado em que há sempre o aumento do controle de equipamentos a longas distâncias, é necessário que esse profissional consiga conectar seu projeto à rede mundial de computadores e como desdobramento desse fato há a busca do uso de banco de dados como forma expansível de armazenamento de informações, além da possibilidade de fazer integrações entre bancos e recursos online, como por exemplo a interação com planilhas, geradores de gráficos e outros programas que podem ser utilizados na análise de carga de dados e geração de históricos para possíveis previsões de cenários.

Como se tratou de um projeto de arquitetura básica, há pontos e propostas de melhorias futuras que são: adaptar o projeto para fazer uso de requisições HTTPS tanto na construção do servidor quanto no consumo pelo microcontrolador; sistema de pesagem de precisão e consequente adequação na construção do protótipo físico; uso de servo motor para redução de custos e maior facilidade para controlar a vazão; construção de uma aplicação mais performática e aprimoramento do *front-end* para melhor experiência do usuário.

Mediante a toda experiência adquirida na produção deste trabalho, conclui-se que para todos que desejam trabalhar com sistemas embarcados, principalmente na área de IoT e não desejam ficar limitados e posteriormente ultrapassados no mercado é essencial e na prática obrigatório o uso de recursos alocados na nuvem, assim como o aprendizado no desenvolvimento de aplicações web, já que não basta ter uma boa coleta e processamento de dados, se o profissional ficar dependente de outro para poder confeccionar uma interface do serviço/produto, sendo importante salientar que uma boa usabilidade e conforto visual para os usuários é um enorme diferencial competitivo.

REFERÊNCIAS

AMAZON. **O que é a computação em nuvem?** 12 de Dezembro de 2023. Disponível em: <https://aws.amazon.com/pt/what-is-cloud-computing/>.

AMAZON. **O que é middleware?** 2023. Disponível em: <https://aws.amazon.com/pt/what-is/middleware/>.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. **Computer Networks**, v. 54, n. 15, p. 2787–2805, 2010.

CUNHA, A. F. O que são sistemas embarcados. **Saber Eletrônica**, v. 43, n. 414, p. 1–6, 2007. Citado na página 17.

DELICATO, F. C.; PIRES, P. F.; BATISTA, T. **Middleware Solutions for the Internet of Things**. London: Springer, 2013. ISBN 978-1447151088.

DZP, P. **Modelo Relacional**. 2023. Disponível em: <https://portal.dzp.pl/nhk/modelagem-de-dados-modelo-relacional.html>.

ELETROGATE. **Módulo de Relé de 1 Canal 5V**. 2023. Disponível em: <https://www.eletrogate.com/modulo-rele-1-canal-5v>.

ELETROGATE. **Módulo WiFi ESP32s Bluetooth 38 pinos**. 2023. Disponível em: <https://www.eletrogate.com/modulo-wifi-esp32s-bluetooth-38-pinos>.

HOSTINGER. **Por Que o GitHub é Tão Popular**. 2023. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-github#Por_Que_o_GitHub_e_Tao_Popular>.

INSOMNIA. **Get Started**. 2023. Disponível em: <https://docs.insomnia.rest/insomnia/get-started>.

INTEL. **What Is Bluetooth Technology?** 2023. Disponível em: <https://www.intel.com/content/www/us/en/products/docs/wireless/what-is-bluetooth.html>.

LEE, E. A.; SESHIA, S. A. **Introduction to Embedded Systems: A Cyber-Physical Systems Approach**. [S.l.]: MIT Press, 2016. ISBN 978-0262533812.

MASCHIETTO, L. G. **Arquitetura e Infraestrutura de IoT**. [S.l.]: SAGAH, 2021. ISBN 978-6556901947.

MICROSOFT, L. **Dados não relacionais e NoSQL**. 2023. Disponível em: <https://learn.microsoft.com/pt-br/azure/architecture/data-guide/big-data/non-relational-data>.

MUSSIO, L. D.; MAIA, R. F.; LOPES, G. W. Um estudo de arquiteturas iot para dispositivos embarcados. **Sao Bernardo do Campo**, v. 5, 2015.

NODE.JS. **About Node.js**. 2023. Disponível em: <https://nodejs.org/en/about>.

OLIVEIRA, C. ComparaÇÃo entre linguagens de programaÇÃo no mercado, ensino superior e pesquisas cientÍficas na cidade de sÃo paulo. **Artigo apresentado no Instituto Federal de São Paulo**, 2020.

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

OLIVEIRA, S. S. de; KNIESS, J. Arquitetura baseada em internet das coisas para medição e monitoramento de resíduos. **Acta Scientiae et Technicae**, 2023. Disponível em: <https://periodicos.univali.br/index.php/acotb/article/view/12744/0>.

ORACLE. **What Is a Database?** 2023. Disponível em: <https://www.oracle.com/br/database/what-is-database/>.

ROBOBUILDERS. **Motor Dc 12v Alto Torque 25ga 370**. 2023. Disponível em: <https://www.robobuilders.com.br/motor-dc-24v-alto-torque-25ga-370-escolha-a-rotacao>.

ROBOCORE. **Fonte Ajustável para Protoboard**. 2023. Disponível em: <https://www.robocore.net/regulador-de-tensao/fonte-ajustavel-para-protoboard>.

SEBESTA, R. W. **Concepts of Programming Languages**. [S.l.]: Pearson, 2015.

SEBRAE. **Quando surgiu a indústria 4.0**. s.d. Disponível em: <https://sebrae.com.br/sites/PortalSebrae/artigos/quando-surgiu-a-industria-40,4542c009cbce3810VgnVCM100000d701210aRCRD>. Acesso em: 8 dez. 2023.

SERVICES, I. A. W. **What Is Amazon RDS?** 2023. Disponível em: <https://aws.amazon.com/pt/what-is/sql/>.

VERCEL. **Vercel Documentation**. 2023. Disponível em: <https://vercel.com/docs>.

WEISER, M. The computer for the 21st century. **Scientific American**, v. 265, p. 94–105, 1991.

WILMSHURST, T. **Designing Embedded Systems with PIC Microcontrollers: Principles and Applications**. The Boulevard, Langford Lane, Kidlington, Oxford OX5 1GB, UK: Elsevier, 2010. ISBN 978-1856177504.