UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DAINF - DEPARTAMENTO ACADÊMICO DE INFORMÁTICA E
DAELN - DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

ALINE BORGES
JULIO VANCINI

FREEZER TROPICÁLIA

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2021

ALINE BORGES
JULIO VANCINI

FREEZER TROPICÁLIA

Tropicalia Freezer

Proposta de Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná, como requisito parcial para a obtenção do título de Bacharel.

Orientador: Hugo Vieira Neto
DAELN - Departamento Acadêmico de Eletrônica - UTFPR

Coorientadora: Anelise Munaretto
DAINF - Departamento Acadêmico de Informática -UTFPR

CURITIBA
2021

ALINE KOLCZYCKI BORGES
JULIO VANCINI BERNARDI

FREEZER TROPICÁLIA

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 17 / dezembro / 2021

_____
Anelise Munaretto Fonseca
Pós-doutorado na Centre de Recherche Inria Saclay
Universidade Tecnológica Federal do Paraná, Campus Curitiba

_____
João Alberto Fabro
Pós-doutorado na Faculdade de Engenharia da Universidade do Porto
Universidade Tecnológica Federal do Paraná, Campus Curitiba

_____
Keiko Veronica Ono Fonseca
Pós-doutorado na Technische Universität Dresden,
Universidade Tecnológica Federal do Paraná, Campus Curitiba

_____
Paulo Cézar Stadzisz
Doutorado na Université de Franche Comté
Universidade Tecnológica Federal do Paraná, Campus Curitiba

CURITIBA
2021

AGRADECIMENTOS

Dedicamos nossos sinceros agradecimentos, primeiramente, ao nosso orientador Hugo Vieira Neto, que infelizmente não está mais entre nós para ver este projeto ser defendido. Ficamos extremamente agradecidos à professora Anelise Munaretto por ter nos acolhido após esse triste evento e nos ajudado na reta final para a entrega deste TCC.

Agradecemos a UTFPR, na forma de Instituição, e a todos os professores com os quais tivemos o privilégio de aprender muito e evoluir como pessoa e profissional. Os frutos dos conhecimentos aprendidos durante o curso já estão sendo colhidos nas trajetórias profissionais.

Agradecemos às nossas famílias e amigos pela paciência e suporte durante o todo este processo. Agradeço ao Alexandre Ribeiro, esposo da Aline, pelo apoio na jornada e também pelos LEDs, Arduinos e osciloscópios emprestados. E também ao ex-aluno Everton Santos Barreto Junior, por ter nos auxiliado no começo do projeto na parte do sistema embarcado e também do suporte em impressão 3D.

ALINE KOLCZYCKI BORGES
JULIO VANCINI BERNARDI

FREEZER TROPICÁLIA

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 17 / dezembro / 2021

_____
Anelise Munaretto Fonseca
Pós-doutorado na Centre de Recherche Inria Saclay
Universidade Tecnológica Federal do Paraná, Campus Curitiba

_____
João Alberto Fabro
Pós-doutorado na Faculdade de Engenharia da Universidade do Porto
Universidade Tecnológica Federal do Paraná, Campus Curitiba

_____
Keiko Veronica Ono Fonseca
Pós-doutorado na Technische Universität Dresden,
Universidade Tecnológica Federal do Paraná, Campus Curitiba

_____
Paulo Cézar Stadzisz
Doutorado na Université de Franche Comté
Universidade Tecnológica Federal do Paraná, Campus Curitiba

CURITIBA
2021

RESUMO

BORGES, Aline; VANCINI, Julio. Freezer Tropicália. 2021. 63 f. Trabalho de Conclusão de Curso – Curso de Bacharelado em Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Curitiba, 2021.

O projeto proposto é denominado Freezer Tropicália, cuja inspiração e demanda surgiu de uma *startup* denominada Tropicália (TROPICALIA, 2020). O objetivo do projeto é facilitar a venda de marmitas congeladas através de um sistema automatizado, que possa ser instalado em diversos locais, sem a necessidade de uma pessoa vendedora. Para isso, a equipe adaptou um freezer com um sistema embarcado, que contabiliza a quantidade de marmitas retirada pelo usuário para cobrança automática. Ele também controla a abertura e fechamento de uma trava automática das portas para permitir o acesso aos produtos. O freezer conecta-se a um aplicativo móvel, que possibilitará ao usuário pedir a abertura da trava e realizar o pagamento das marmitas retiradas. O sistema está dividido em duas partes: um freezer, com um sistema embarcado, e o aplicativo para celular que se contecta a um banco de dados na nuvem. O processo de desenvolvimento adotado foi *Scrum*, com o sistema embarcado desenvolvido usando uma placa *ESP32*, conexão com o celular via Bluetooth, aplicativo móvel para iOS e para o banco de dados foi utilizado o *Firebase*.

Palavras-chave: IoT, *bluetooth*, aplicativo, marmita, máquina de vendas.

ABSTRACT

BORGES, Aline; VANCINI, Julio. Tropicália Freezer. 2021. 63 f. Trabalho de Conclusão de Curso – Curso de Bacharelado em Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Curitiba, 2021.

The proposed project is called "Tropicalia Freezer". The inspiration and demand came from a startup called Tropicalia. The project's goal is to facilitate the selling of frozen meals through an automated system that could be installed in several different places without the need for a cashier. To achieve that, the team has adapted a freezer with an embedded system that counts the number of frozen meals taken by the user. It also controls the opening and closing of the locks that allow access to the products. The freezer connects to a mobile app through Bluetooth, allowing the user to ask for the locks to open and pay for the amount of frozen meals taken.

Keywords:IoT, *Bluetooth*, mobile app, frozen meals, vending machine

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Contexto e Motivação do Trabalho

A motivação principal deste TCC é projetar uma solução para uma empresa denominada Tropicália Prático Saudável (TROPICALIA, 2020) que necessitava otimizar suas vendas através de pontos estratégicos. Esta solução deveria possibilitar o aumento da escala de vendas sem que o custo aumente de forma proporcional.

Assim, a Tropicália Prático Saudável (TROPICALIA, 2020), na época uma *startup* de alimentação saudável, visava inovar a maneira como as pessoas se alimentam, tanto na hora da refeição quanto na hora da compra. A demanda desta empresa era ter uma abordagem mais distribuída por meio de freezers localizados em pontos estratégicos, que poderiam vender automaticamente seus produtos sem a necessidade de um funcionário em cada ponto de venda. Desta maneira, seria possível otimizar os custos logísticos, escalar as vendas e manter a marca visível aos consumidores.

Uma *Vending Machine* (CAFÉ, 2018) tradicional é uma máquina em que o usuário escolhe o produto que gostaria, paga por ele e a máquina entrega o produto, e é comumente utilizada para venda de latas ou *snacks*. Esta máquina costuma ter um custo bem elevado e não permite que o usuário interaja diretamente com os produtos.

A proposta deste projeto é construir um protótipo similar a uma *Vending Machine*, de menor custo e com mais liberdade para o usuário. O sistema do Freezer Tropicalia permite ao usuário, de maneira totalmente automatizada, retirar a quantidade de marmitas congeladas que gostaria de comprar, calcula essa quantidade e realiza a cobrança.

Este trabalho também traz algumas novidades de tecnologia em forma de um arranjo diferente de tecnologias já existentes. O modelo se enquadra em um projeto de caráter IoT(Internet das Coisas em português), já que possui uma conexão entre o sistema embarcado e a internet (GOIÁS, 2020). Entretando o sistema embarcado não faz a conexão direta com a internet, e sim utiliza a conexão com *Bluetooth Low Energy* com um aplicativo móvel, que possui conexão com a internet para que este realize a transmissão de dados para a nuvem. Isto permite uma maior flexibilidade na escolha de locais para instalar o freezer, já que uma conexão direta com a internet não é um requisito do sistema embarcado.

Esta integração entre uma *startup* e este projeto de TCC traz um exemplo da interação universidade/faculdade com as empresas e como ambas podem se beneficiar desta relação (GARCIA, 2018).

1.2 Objetivos

O objetivo principal deste projeto é desenvolver um protótipo funcional que, com uma adaptação de um freezer disponível no mercado, seja capaz de realizar uma venda automatizada, sem necessidade de uma pessoa para atender o cliente e efetuar a venda. Este será composto principalmente em dua partes: a primeira é um sistema embarcado, sendo composto pelo freezer, pelos equipamentos eletrônicos e suporte de travas. E em segundo, o aplicativo para celular. O usuário irá ter acesso direto aos produtos e, através deste aplicativo para celular, poderá interagir com o freezer liberando acesso, escolhendo os produtos e realizando o pagamento no próprio aplicativo. Finalmente, será realizado o levantamento dos custos para montar o protótipo.

1.2.1 Objetivos Específicos

Para atingir o objetivo principal, os objetivos específicos do projeto são:

- Projetar e implementar a adaptação física do freezer, com travas de abertura, balança para cálculo da quantidade de marmitas retiradas e sensores nas portas que detectam se as portas estão abertas ou fechadas.
- Projetar e desenvolver um software embarcado capaz de realizar as leituras e escritas dos componentes de hardware e se comunicar com o aplicativo utilizando *Bluetooth Low Energy*.
- Projetar e desenvolver um aplicativo móvel para *iOS* que provê a interface gráfica para o usuário realizar as compras das marmitas congeladas.
- Estruturar um banco de dados na nuvem que permita que sejam registradas todas as informações necessárias, assim como prover autenticação para o usuário.

1.3 Organização do documento

O presente trabalho está dividido em sete capítulos. Este capítulo apresenta uma visão geral das motivações, justificativas e objetivos deste trabalho. O Capítulo 2 apresenta uma contextualização teórica dos principais assuntos abordados neste trabalho. O Capítulo 3 apresenta os requisitos do sistema e descrição detalhada do projeto. O Capítulo 4 aborda como o projeto foi implementado, incluindo hardware e software. No Capítulo 5 foram abordados os custos totais e comparativos do projeto. O Capítulo 6 apresenta os resultados do trabalho e, por fim, o Capítulo 7 expõe as conclusões retiradas deste trabalho e descreve as oportunidades de melhorias futuras.

2 FUNDAMENTAÇÃO TEÓRICA E MATERIAIS UTILIZADOS

Nesta seção será abordada a fundamentação teórica necessária para o desenvolvimento do projeto.

2.1 Freezer

O freezer escolhido para adaptação foi o freezer Slim SC124, mostrado na figura 1. Este modelo é um freezer do tipo expositor horizontal vitrine, e é específico para exposição e conservação de sorvetes ou outros produtos congelados. Sua estrutura externa é em chapa de aço kroma e, por dentro, possui uma única junção da chapa vincada. Também possui serpentina fixa por solda ponto nas 4 laterais, fundo e degrau do motor. Dessa forma, não se solta do tanque e garante que a temperatura do produto no interior seja distribuída por igual. O acesso ao interior é através de tampas de vidro deslizante, de baixa emissividade e com sistema de drenagem exclusivo. Seu isolamento é todo em poliuretano injetado (38 a 42Kg/m3), que é fabricado com o sistema *skin condenser* (sem micro-motor ventilador). Isto garante o baixo consumo de energia, um congelador silencioso e com eficiência energética. Todos os equipamentos são certificado pelo Inmetro segundo a fabricante (ÁRTICO, 2021).

Figura 1 – Freezer modelo SC124 da empresa Ártico

Fonte: (ÁRTICO, 2021)

As especificações do freezer são:

- Medidas Externas: (C) 615mm x (L) 720mm x (A) 924mm.

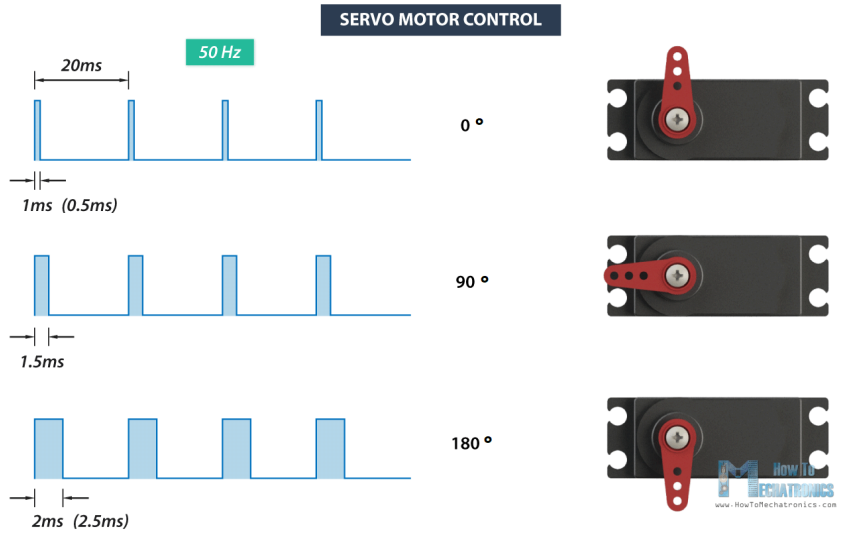
- Medidas Internas: (C) 430mm x (L) 540mm x (A) 620mm.
- Peso: 39 Kg.
- Capacidade: 120 litros.
- Tensão: 127V ou 220V/monofásico (o adquirido é de 127V).
- Temperatura média de funcionamento de -18ºC até -22ºC.

Neste modelo destacam-se as folhas de vidro, uma superior e outra inferior, já que essa configuração será importante para o tipo de adaptação escolhido para as travas e controle do fechamento e abertura do freezer.

2.2 Servo motor

Servo motores são controlados por pulsos que variam sua largura de banda, comumente chamados de PWM(*Pulse Width Modulation*) (MECHATRONICS, 2018). Essa largura de banda por sua vez codifica uma posição específica da posição do braço do servo motor, conforme a figura 2.

Figura 2 – Relação entre o PWM e a posição do servo



Fonte: (MECHATRONICS, 2018) adaptado.

Para as travas físicas das folhas de vidro, utilizou-se o servo motor SG-5010 da TowerPro, que possui rotação de 90° para cada direção totalizando 180°. Ele tem três pinos conforme a figura 3, em que o vermelho é para a tensão de alimentação, o marrom para o GND e o laranja para entrada de sinal. Demais especificações constam na tabela 1.

Para utilização do servo via placa ESP32 fez-se necessário o uso da biblioteca ESP32Servo, que controla o **PWM** da placa, os timers e facilita a sintaxe para a posição alvo desejada(HARRINGTON, 2020).

Figura 3 – Servo motor SG-5010 TowerPro

Fonte: (TOWERPRO, 2020)

Tabela 1 – Especificações do Servo Motor SG-5010

Model	SG5010
Peso(g)	39
Tensão de operação	4.8 – 6.6 V
Velocidade de operação @4.8V	0.20sec/60°
Velocidade de operação @6.6V	0.16sec/60°
Torque de estagnação @4.8V	5.5 kg-cm
Torque de estagnação @6.6V	6.5 kg-cm
Temperatura de operação (°C)	0 to 55
Largura da banda morta	1 µs
Tipo de engrenagem	Fibra de vidro
Grau de rotação	180°
Tomada do servo	JR
Comprimento do cabo	32 mm
Comprimento (mm)	38
Largura (mm)	20
Altura (mm)	40
Peso do envio	0.100 kg
Dimensões de envio	10 x 8 x 6 cm

Fonte: (TOWERPRO, 2020)

2.3 Balança

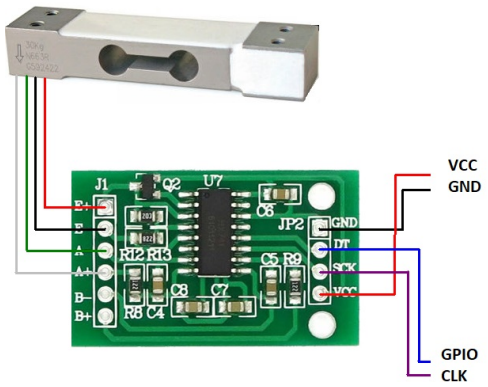
Foi utilizado um sensor de carga para a balança, denominado *load cell*, e o módulo HX711 que atua como conversor e amplificador de sinal. As saídas da célula foram ligadas no módulo segundo a figura 4. As especificações da *load cell* mais pertinentes ao projeto constam na tabela 2. Por estar dentro do freezer, o sensor de carga precisa suportar as temperaturas negativas, o que é possível com as especificações do sensor escolhido. Com o limite máximo de -20° de temperatura de operação, também é necessário ressaltar que a calibração deve ser feita com o freezer ligado para que não haja alteração da sensibilidade

da célula de carga com a temperatura, assim registrando valores incorretos. A precisão da célula é condizente com a aplicação do projeto.

O Módulo Conversor HX711, que pode ser visto também na figura 4 em verde e na figura 5, foi utilizado com a finalidade de fazer a conversão das alterações de valor da resistência dos sensores de uma balança em dados digitais, por meio do circuito ADC de 24-bit.

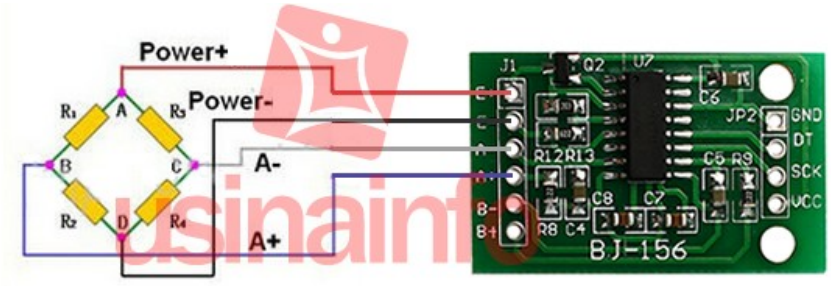
Como os sensores de peso instalados nas balanças não oferecem dados com grande precisão é necessário o uso de um conversor HX711, que também funciona como um amplificador de sinal para oferecer dados mais exatos. Ele foi projetado especialmente para balanças digitais e aplicações de controle industrial. Seu princípio de funcionamento é converter as mudanças medidas em alteração do valor de resistência, através do circuito de conversão em potência elétrica (USINAINFO, 2021).

Figura 4 – Célular de Carga e o módulo HX711.



Fonte: (MYBOTIC, 2020) adaptado.

Figura 5 – Módulo conversor e amplificador HX711 de 24 bit



Fonte: (USINAINFO, 2021).

Tabela 2 – Especificações CZL635 ou Célula de Carga

Características Mecânicas	
Capacidade de Carga	5kg
Material	Liga de alumínio
Tipo de célula de carga	Medidor de tensão
Características Elétricas	
Precisão	0.05 %
Classificação de Saída	1.0±0.15 mv/V
Efeito da temperatura em Zero (cada 10°C)	0.05% FS
Efeito da temperatura na amplitude (cada 10°C)	0.05% FS
Impedância de entrada	1130±10 Ohm
Impedância de saída	1000±10 Ohm
Tensão de excitação	5 VDC
Faixa de temperatura operacional	-20 até +55°C
Sobrecarga segura	120% capacidade
Sobrecarga final	150% capacidade

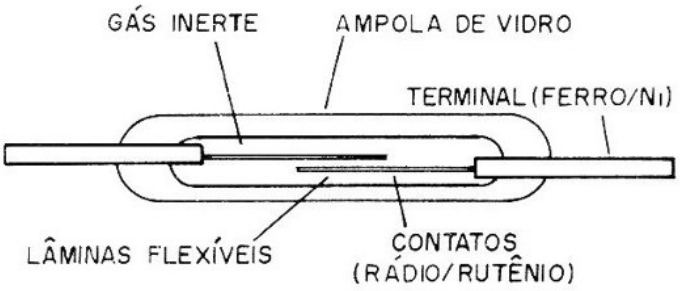
Fonte: (CZL635, 2011)

2.4 Reed Switch

Para realizar a leitura do estado atual das portas de vidro, se estão abertas ou fechadas , utilizou-se o sensor *reed switch*. Este sensor é um interruptor de lâminas, como o nome sugere, e pode ser descrito como um interruptor ou chave que pode ser acionado por um campo magnético de uma bobina ou de um imã. O tipo mais comum, que é o interruptor simples de lâminas, tem sua estrutura mostrada na figura 6. Este componente consiste numa ampola de vidro, no interior, da qual existem duas lâminas flexíveis com contatos especiais em suas extremidades. A ampola, para evitar a oxidação dos contatos, é preenchida com um gás inerte. Para o tipo normalmente aberto (NA), que funciona como um interruptor simples, em condições normais, as lâminas ficam separadas e este interruptor se mantém aberto. As lâminas são feitas com um material ferroso, o que significa que a presença de um campo magnético, como o de um imã, faz com que o circuito se feche, conforme mostra a figura 7 (BRAGA, 2010).

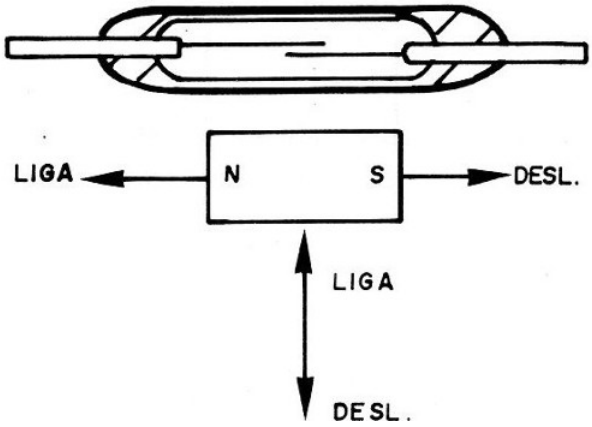
Neste projeto foi utilizado a forma de aproximação desta mesma figura. Para facilitar a tradução da resposta do sensor, foi utilizado o módulo interruptor magnético presente na figura 8. Este possui três pinos: um para a alimentação, outro para o negativo e o último para saída digital *DO*. Este retorna "1"quando estava sem estímulo do imã (para o projeto significava que a folha de vidro estava aberta) e "0"quando o circuito se fechava devido ao campo magnético do imã(sinalizando que a folha de vidro estava fechada corretamente). As principais características elétricas deste módulo são apresentadas na tabela 3.

Figura 6 – Sensor Reed Switch



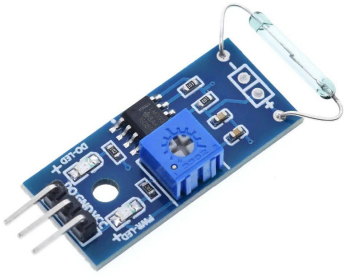
Fonte: (BRAGA, 2010).

Figura 7 – Aplicação e funcionamento do Sensor Reed Switch



Fonte: (BRAGA, 2010).

Figura 8 – Módulo Reed Switch



Fonte: (MAGSWITCH, 2020).

2.5 Bluetooth Low Energy

O *Bluetooth* clássico é uma tecnologia de transmissão de dados que utiliza ondas de rádio a 2.4GHz e é tradicionalmente utilizada para transmissão de áudio sem fio, entre

Tabela 3 – Especificações do Módulo Reed Switch

Características elétricas
Reed Switch Normalmente aberto
Saída do comparador
Sinal limpo
Capacidade de condução da forma de onda superior a 15mA
Tensão de operação 3.3V - 5V
Saída digital (0 and 1)
Medidas: 3.2x1.4cm
Comparador de ampla tensão LM393

Fonte: (MAGSWITCH, 2020).

outras aplicações. O *Bluetooth Low Energy* (BLE) utiliza a mesma frequência de rádio, porém é desenvolvida para ultra baixo consumo de energia (BLUETOOTH, 2021). Além do baixo consumo de energia, o BLE tem a grande vantagem de permitir a conexão, pareamento e troca de dados sem que o usuário precise manualmente parear os dispositivos. Na tabela 4 podemos ver algumas das diferenças entre os tipos de *Bluetooth*.

Tabela 4 – Características do Bluetooth Low Energy e Bluetooth Clássico

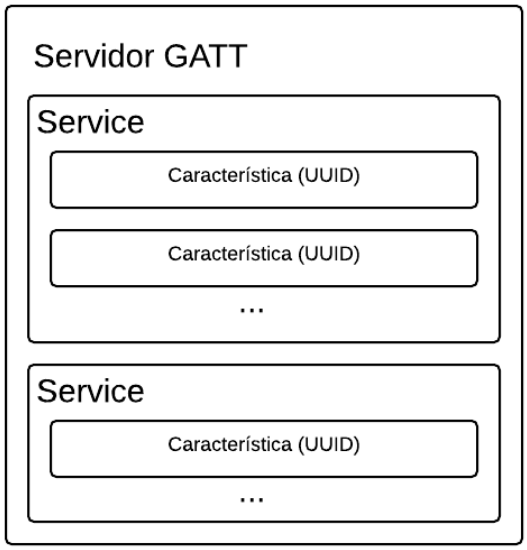
	Bluetooth Low Energy	**Bluetooth Clássico**
Frequência da Banda	2.4GHz ISM Band (2.402 – 2.480 GHz)	2.4GHz ISM Band (2.402 – 2.480 GHz)
Canais	40 Canais Espaçamento de 2GHz (3 canais de advertising 37 canais de dados)	79 canais espaçamento de 1 MHz
Transferência de Dados	Assíncrona com conexão Isócrona com conexão Assíncrona sem conexão Síncrona sem conexão Isócrona sem conexão	Asíncrona com conexão Síncrona com conexão

O *Attribute Protocol* (ATT) define dois papéis para os dispositivos: Servidor e Cliente. Um cliente pode solicitar dados de um servidor e o servidor envia dados para o cliente. O protocolo também define que se uma solicitação ainda está pendente, outras solicitações de dados não podem ser enviadas até que a resposta seja recebida e processada (PESSOA, 2016).

O GATT (*Generic Attribute Profile*) é a estrutura que define como o Servidor e Cliente irão trocar informações, por meio de características. Um Servidor contém dados

organizados em formas de atributos com um identificador único universal (UUID), de 16, 32 ou 128 bits. A figura 9 contém exemplificação de como são contidas as características.

Figura 9 – Ilustração de um servidor GATT e suas características



Fonte: Autoria própria

Cada característica pode ter uma definição de **permissão de acesso**, que determina o que o cliente pode consumir ou receber de dados do Servidor:

- **Leitura**: A característica pode ser lida por um cliente.
- **Escrita**: A característica pode receber um valor por escrita de um cliente.
- **Notify**: A característica envia dados para o cliente de forma contínua, notificando de alguma mudança.

Além da escrita e leitura, uma funcionalidade interessante do *BLE* são os updates iniciados pelo servidor, que é uma notificação de mudança de valores (*Value Notification*). Esses updates são pacotes asíncronos que o servidor pode enviar para mandar mudanças de valores de uma característica de forma contínua, sem que o cliente precise ficar perguntando a cada intervalo de tempo pelo novo valor. Esse modo economiza banda e energia do servidor.

2.6 Aplicação Móvel

Foi definido para esse projeto criar uma aplicação para dispositivos *iOS*, que é o sistema operacional utilizado em dispositivos móveis da plataforma Apple. Foi escolhido o aplicativo para apenas a plataforma iOS, em primeiro momento, pela maior familiaridade da plataforma pelos participantes da equipe. É proposto como um trabalho futuro a implementação do aplicativo para plataforma Android. O projeto da aplicação iOS foi

desenvolvido utilizando a linguagem Swift, que é a linguagem oficial para esse tipo de aplicação, e o ambiente Xcode. Em seguida, serão descritos alguns termos comuns do desenvolvimento de aplicações móveis.

2.6.1 Arquitetura VIP-C

Existem diversas arquiteturas de software comuns para desenvolvimento de aplicativos móveis, entre elas, MVC (*Model View Controller*) ou MVVM (*Model View ViewModel*). A arquitetura escolhida para este projeto foi a VIP-C (*View Interactor Presenter - Coordinator*) (ATANASOV, 2017). Cada tela do app é um "componente"que tem as três classes, *View, Interactor e Presenter*, conectadas entre si via composição. Nesta arquitetura, a responsabilidade de cada uma das classes é descrita abaixo:

- ***View***: Coloca e organiza os componentes na tela, como botões, imagens e listas.
- ***Interactor***: É a classe responsável pela lógica de negócios, cálculos importantes e comunicações externas com Banco de Dados ou Bluetooth.
- ***Presenter***: É a classe responsável por traduzir a lógica de negócios para a lógica de apresentação. Por exemplo, converte um código de erro em uma mensagem agradável ao usuário.
- ***Coordinator***: É a classe que cuida da navegação entre os componentes. Dessa forma, um componente de tela não precisa ter uma referencia de qual é a próxima tela a ser apresentada.

Além da descrição da arquitetura principal de apresentação das telas, alguns outros padrões de design de *software* utilizados foram:

- ***Service***: É uma classe que define as conexões com o banco de dados externo (no caso deste projeto, com o Firebase).
- ***Worker***: É uma classe que faz algum trabalho 'contínuo' e frequentemente é compartilhada entre diversos interactors. No caso do projeto, a conexão e conversação com o *Bluetooth* é um worker.
- ***Builder***: É uma classe que serve para construir e conectar("linkar") as referências corretamente entre as classes de um componente VIP.

A construção e definição das classes segue o conceito de inversão de dependências (MARTIN, 2019). Neste conceito, as dependências de uma classe (por exemplo, service ou worker) são contruídas fora da mesma e passadas via construtor. A classe terá apenas a referência ao protocolo (interface) dessa dependência, e não precisa saber qual é exatamente a classe concreta que implementa aquele protocolo.

2.7 QRCode

O **QRCode**, ou Código de resposta rápida, é um código que é lido rapidamente por um dispositivo móvel. Usando uma combinação de espaçamento como um tipo de Código de Barras em formato de matriz, o **QRCode** pode conter uma grande variedade de informações (STEIN, 2020). Na figura 10 podemos ver um exemplo de um **QRCode** e um código de barras tradicional. Um **QRCode** consegue armazenar muito mais informações que um Código de Barras, e qualquer texto pode ser codificado para uso (com um limite de 4296 caracteres alfanuméricos).

Figura 10 – Na esquerda, um **QRCode** e, na direita, um código de barras 2D.

Fonte: (STEIN, 2020)

2.8 Formato JSON

Comumente conhecido como JSON, o *JavaScript Object Notation* é um formato leve de troca de dados entre sistemas. É um formato simples de ser lido e criado por humanos, e fácil de ser entendido e gerado por máquinas (JSON..., 2021). Esse formato é independente de linguagem de programação e se baseia em estruturas de dados universais, em que praticamente todas as linguagens de programação suportam como, por exemplo, *arrays* e estruturas de chave/valor.

No projeto do sistema embarcado foi utilizada a biblioteca de código aberto *ArduinoJSON* e, para o aplicativo móvel, o protocolo *Codable* suportado nativamente pela linguagem *Swift*. A seguir pode-se ver um exemplo de como um JSON é estruturado.

```
1 {
2     "chave": "valor em texto",
3     "chave_inteiro": 1,
4     "array": [
5         "elemento 1",
6         "elemento 2",
7         "elemento 3"
8     ]
```

```
9 }
```

2.9 Banco de Dados

Cloud Firestore é um produto do *Firebase*, da *Google* (CLOUD. . . , 2021). É um banco de dados na nuvem flexível, escalável e usa a tecnologia NoSQL (*("Not only SQL")*. Nesse paradigma, no caso do *Firestore* o banco de dados não é relacional, e sim tem uma estrutura parecida com documentos JSON. Cada documento tem um id único e contém pares chave-valor. Os valores têm tipos que podem variar, como strings, inteiros, *booleans*, array e objetos complexos (MONGODB, 2021).

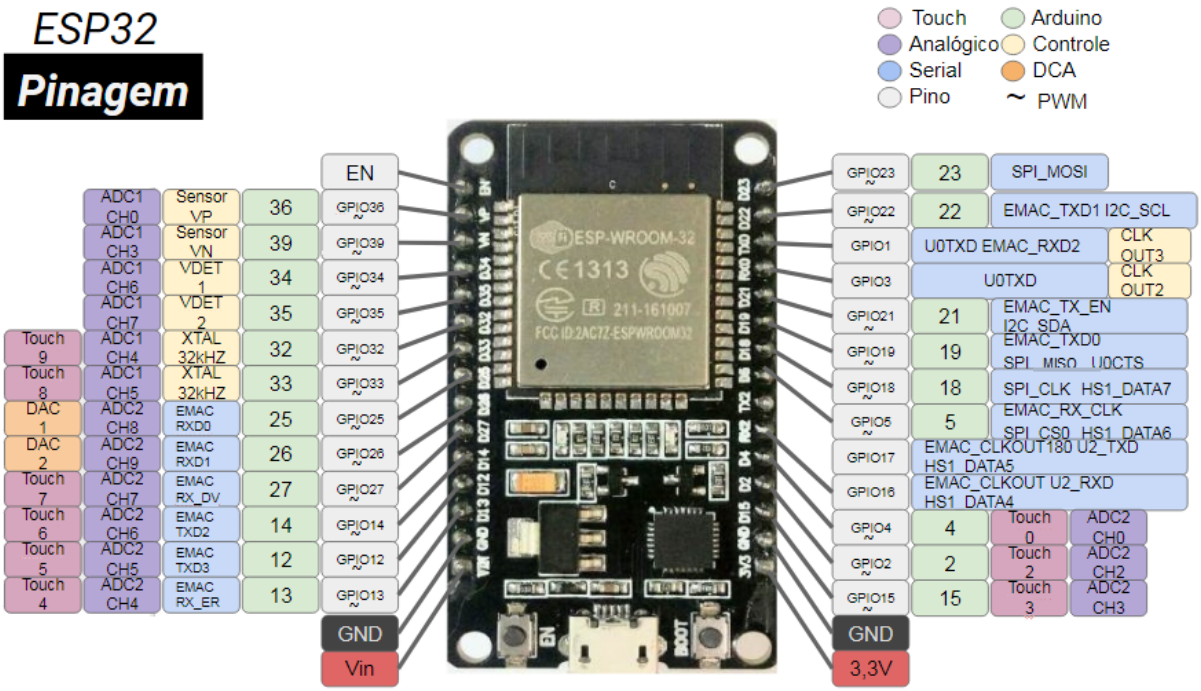
O *Cloud Firestore* é um banco de dados que conecta diretamente com iOS, Android e aplicações web facilmente através de um SDK. Com ele, é possível desenvolver aplicações *Serverless*, ou seja, o cliente faz escritas e leituras diretamente sem a necessidade de um serviço hospedado na *web*.

2.10 Sistema Embarcado

Para o controle de todo sistema embarcado foi escolhida a placa ESP32, desenvolvida pela empresa Espressif (ESPRESSIF, 2021). Ela foi escolhida porque esse microcontrolador é pequeno, simples de programar, de baixo custo e principalmente, por já ter o módulo *Bluetooth* acoplado. Para o projeto era extremamente necessário a compatibilidade com o BLE, principalmente por causa da maneira de como a placa se comunica com a rede. Essa discussão será abordada em outra parte deste documento. Nesta seção será dada ênfase a pontos específicos pertinentes ao projeto. Demais especificações podem ser encontradas no *datasheet*. Na figura 11 está representada a pinagem e as características de forma didática. Nesta imagem é possível ver um resumo de como utilizar corretamente as portas e suas limitações, como por exemplo, qual porta suporta o PWM visto em outra seção. Na figura 12 pode ser visto o diagrama de blocos do funcionamento da placa. O PWM gerado pela placa é dedicado por um sub-módulo, consistindo em timers próprios podendo ser síncronos ou independentes, onde cada operador PWM gera uma onda para cada canal. Com isso é possível gerenciar motores controlados digitalmente como o servo motor. O chip integra um Bluetooth Link Controller e Bluetooth Baseband, que executam os protocolos de banda base e outras rotinas de link de baixo nível, como modulação/desmodulação, processamento de pacotes, processamento de fluxo de bits, salto de frequência e etc. O Bluetooth Stack do chip é compatível com o Bluetooth V4.2 br/edr e Bluetooth Low Energy.

Dentro das características de operações recomendadas, destaca-se a temperatura mínima de -40° e máxima de 125°C. Dentro destas especificações é possível colocar, se

Figura 11 – Pinagem ESP32



Fonte: (CIRCUITO, 2018)

necessário, a ESP32 dentro do freezer em operação (ESPRESSIF, 2021).

2.11 Ferramentas utilizadas

Para desenvolvimento do software embarcado, foi utilizado a IDE Visual Studio Code com o plugin Platform IO (PLATFORMIO, 2021). Esse plugin permite o desenvolvimento em uma IDE mais robusta que a original do Arduino. Como linguagem, foi utilizado C++ com bibliotecas do Arduino e Programação Orientada a Objetos.

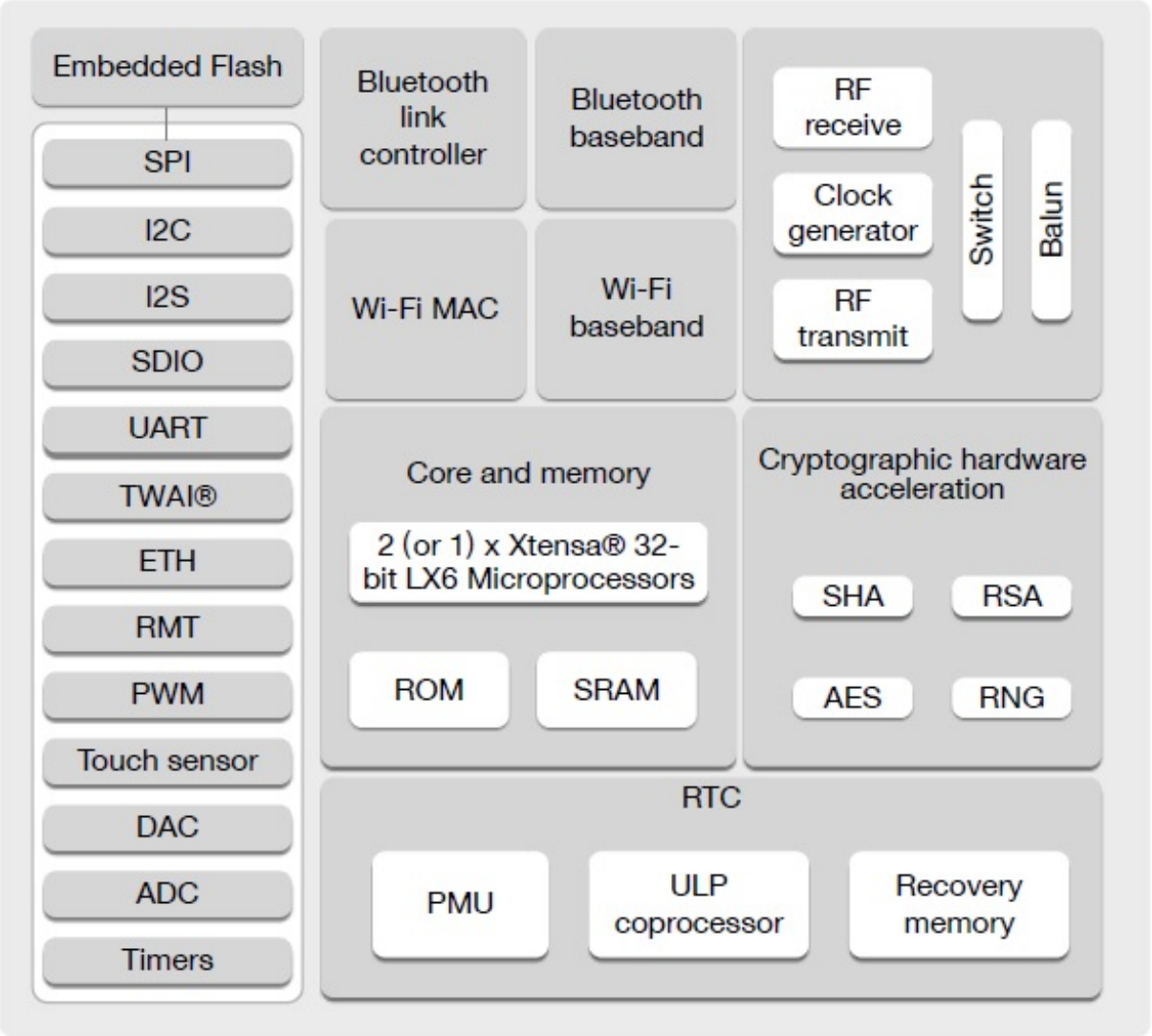
Para o desenvolvimento da Aplicação móvel em iOS, foi utilizado o *Xcode* como IDE de desenvolvimento, com a linguagem *Swift*.

Foram utilizadas algumas bibliotecas Open Source para auxiliar no desenvolvimento, tanto no software embarcado quanto do aplicativo móvel, listadas a seguir:

Software Embarcado:

- **ArduinoJson** (v6.17.3) Faz a decodificação e a codificação de mensagens no formato JSON para comunicação com o aplicativo móvel.
- **HX711** (v0.7.4) . Serve para leitura de célular de carga e balanças (NECULA, 2021).
- **ESP32Servo** (v0.9.0) Permite que a placa ESP32 controle servo motores. Podendo controlar vários tipos de servos e utiliza os timers do PWM. Consegue controlar até 16 servos individualmente (HARRINGTON, 2020).

Figura 12 – Diagrama de Blocos da Esp32



Fonte: (ESPRESSIF, 2021)

Aplicativo iOS:

- **AlamofireImage** (ALAMOFIRE, 2021) (MIT) *Download* de imagens a partir de uma URL.
- **Lottie** (AIRBNB, 2021) (Apache 2.0) Facilita a adição de animações complexas no aplicativo.
- **NewYorkAlert** (HASHIBA, 2020) (MIT) Layout de alertas no estilo *pop-up*.
- **RxSwift** (ZAHER; MISHALI, 2021) (MIT) Paradigma de programação reativa, orientada a eventos.
- **RxCBCentral** (UBER, 2020) (Apache 2.0) Facilita a comunicação com periféricos *BLE* utilizando o paradigma de programação reativa (*RxSwift*).
- **Firebase** (FIREBASE, 2021) (Apache 2.0) SDK de comunicação com o *Cloud*

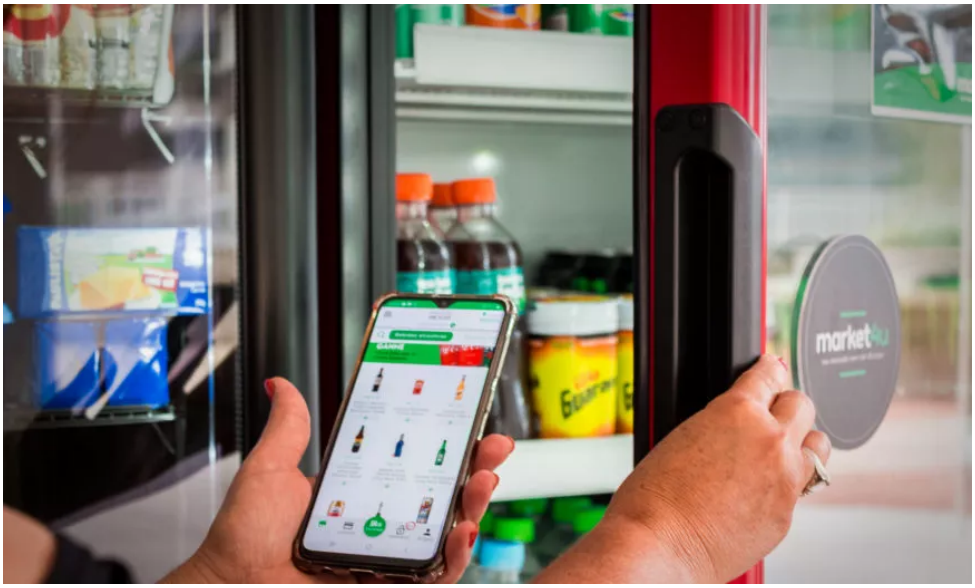
Firestore.

- **GoogleMaps** (GOOGLE, 2021) (Apache 2.0) SDK para visualização de mapas e busca de endereços.
- **SnapKit** (MIT) Facilita a criação de layouts com *constraints* no iOS.
- **SkyFloatingLabelTextField** (Apache 2.0) Componente de entrada de texto com *design* diferenciado.

2.12 Trabalhos relacionados

O projeto foi inspirado na famosa máquina de vendas ou *vending machine*. Esse conceito de máquina de vendas autônomo é relativamente novo, porém já bem difundido no meio empresarial, de hospitais e até de condomínios. Surgindo para sugerir soluções para aumentar a produtividade dos colaboradores com o fornecimento de lanches e café, por exemplo; promover praticidade no manuseio da máquina por colaboradores e clientes; redução de custo de mão-de-obra devido ao caráter automático e independente; redução de desperdício de alimentos e bebidas pelo fornecimento de cada porção de acordo com a demanda específica entre outros. (CAFÉ, 2018)

Figura 13 – Trabalhos relacionados: empresa Market4u



Fonte: (MARKET4U, 2020)

O projeto também foi baseado no modelo honestidade, como por exemplo o da Market4you (MARKET4U, 2020) e uma loja inaugurada em São Carlos (EPTV1, 2020), dentro de um condomínio fechado. Ambas possuem uma tecnologica similar que também se utilizam do **QRCode** para liberação ou acesso aos produtos(conforme figura 13) e também leitura de código de barras para compra dentro da loja.

3 PROJETO

Nesta seção será apresentado um detalhamento do projeto incluindo suas caraterísticas e especificações de requisitos funcionais e técnicos.

3.1 METODOLOGIA

Para mecanismos de gestão, o processo de desenvolvimento escolhido foi o método tradicional *Waterfall* ou Modelo de Cascata. Esta metodologia é uma abordagem de gerenciamento de projeto linear, em que os requisitos das partes interessadas(*stakeholders*) e do cliente são reunidos no início do projeto e, em seguida, um plano de projeto sequencial é criado para acomodar esses requisitos. O modelo em cascata tem esse nome porque cada fase do projeto desdobra-se na próxima, seguindo continuamente para baixo como uma cachoeira (MANAGER, 2021). Apesar da crescente escolha das metodologias ágeis para projetos de tecnologia, a equipe entendeu que para a primeira parte do projeto, caberia utilizar o *Waterfall*. Isso porque para até a primeira entrega do projeto, sabe-se muito bem o que precise ser entregue, o que deve ser feito e como deve ser feito. Como o alvo do projeto é entregar somente o protótipo e não analisar o feedback em campo, ficou mais adequada esta escolha. Como trabalhos futuros caberia um modelo híbrido de método ágil por causa da coleta e feedbacks dos usuários clientes. O projeto portanto foi dividido da seguinte forma: análise do problema, proposta da solução, avaliação dos riscos e custos, e por fim, a implementação. O problema foi levantado pela empresa Tropicália, quando foi constatada a necessidade de algum tipo de automação no seu processo de distribuição e venda. A equipe levantou algumas possíveis soluções, em que se destacou a idéia do Smart Freezer por ser mais viável financeiramente. Foram levantados alguns problemas, como a possibilidade de fraude pelo consumidor, falha na contabilização das marmitas, etc. O modelo de negócio a ser implantado e a escolha dos equipamentos a serem utilizados foram escolhidos para minimizar ou até mesmo eliminar os problemas levantados. Uma vez que as especificações do projeto foram refinadas, fez se necessário aumentar os conhecimentos a respeito das linguagens de programação, bibliotecas e frameworks a serem utilizados no desenvolvimento, através de cursos e livros.

3.2 Descrição do produto

3.2.1 Visão Geral

O objetivo geral do projeto é que o usuário possa comprar marmitas de forma completamente automatizada, sem nenhuma interação com um humano. Na figura 14 é possível ver a visão geral do projeto e como o sistema embarcado se comunica com o

Figura 14 – Visão Geral do Projeto

Fonte: Autoria própria

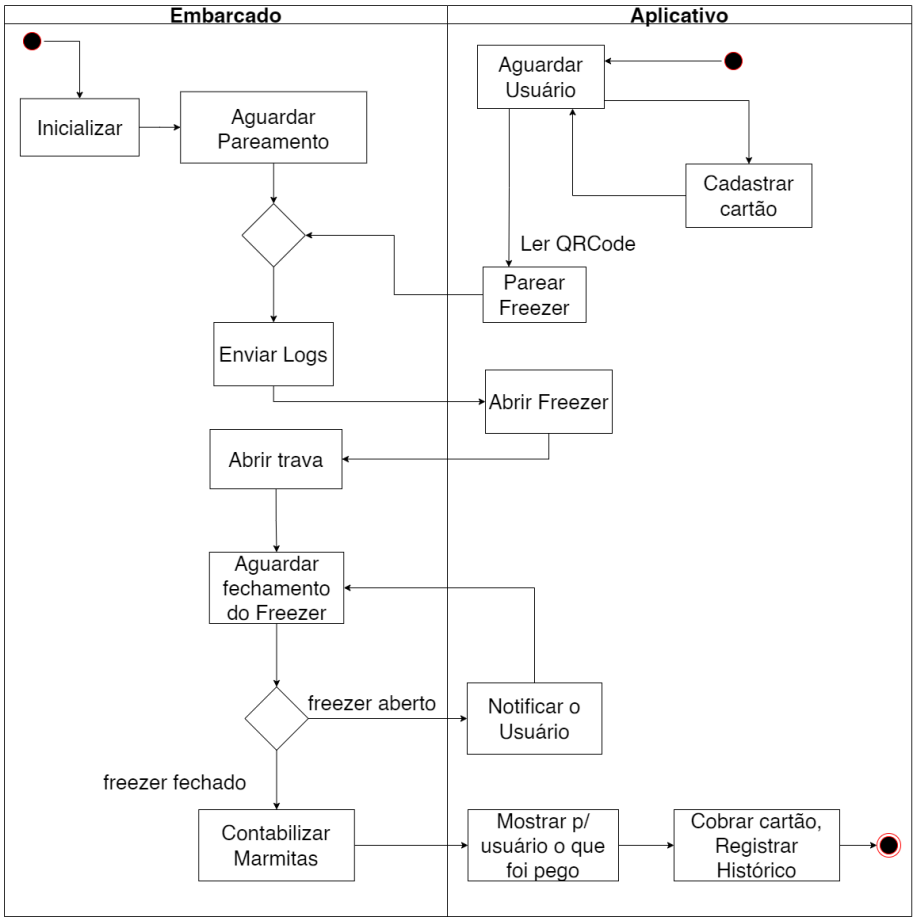
aplicativo, que então se comunica com a rede. No ponto de vista de um usuário do *Freezer Tropicalia*, em alto nível, podemos descrever o projeto, de forma sequencial, da seguinte maneira:

- Usuário utiliza o aplicativo para escanear o QRCode do Freezer do qual gostaria de comprar as marmitas.
- Usuário coloca uma forma de pagamento, caso ela ainda não tenha sido cadastrada previamente.
- Usuário envia um comando para o freezer ser desbloqueado, liberando acesso às marmitas.
- Usuário retira quantas marmitas quiser levar e fecha o freezer.
- Freezer calcula quantas marmitas foram retiradas e envia para o aplicativo.
- Usuário confirma a quantidade de marmitas e realiza o pagamento.

A figura 15 apresenta o diagrama de atividades das interações com o usuário cliente. As interações com o usuário administrador não serão demonstradas neste diagrama para efeitos de simplificação.

Este modelo de negócio baseado na honestidade do cliente tem crescido nos últimos anos, como mercados de condomínio em que o próprio cliente paga pela quantidade correta de itens que leva (FERRASOLI, 2020). Embora o cálculo das marmitas seja automatizado, existe sempre a possibilidade do usuário retirar as marmitas e não finalizar o pagamento pelo aplicativo. Para mitigar este risco, será implementado um sistema de *logs* no sistema embarcado. Cada transação realizada será registrada a partir do usuário e de um *id* de

Figura 15 – Diagrama de atividades



Fonte: Autoria própria

transação, assim como o que aconteceu no sistema internamente durante esta conexão. Quando o próximo usuário vier a realizar uma compra, estes logs serão enviados para o banco de dados na nuvem, de forma transparente. Neste sistema, o embarcado não precisa ficar conectado diretamente à internet, pois utiliza a internet do dispositivo móvel do usuário, pelo aplicativo. Isso facilita a instalação do freezer pois não há obrigatoriedade de uma internet no local para que o sistema funcione.

3.3 Especificação de Requisitos

O sistema como um todo é composto por duas partes principais: o freezer físico, que contém o sistema embarcado, e o aplicativo iOS. Para facilitar a visualização, os requisitos serão descritos separados para cada um dos sistemas.

3.3.1 Sistema Embarcado

Os requisitos funcionais e não funcionais do freezer, em conjunto com o sistema embarcado, são:

Requisitos funcionais:

- RF01 - O sistema deverá destravar as portas ao receber o comando do aplicativo via BLE.
- RF02 - O sistema deverá realizar a leitura do sensor que informa se a porta está aberta ou fechada.
- RF03 - O sistema deverá realizar a leitura da quantidade de marmitas retirada pelo usuário.
- RF04 - O sistema deverá enviar a quantidade de marmitas retirada para o aplicativo.
- RF05 - O sistema deverá permitir a calibração da balança.
- RF06 - O sistema deverá permitir a calibração das travas.
- RF07 - O sistema deverá registrar um log dos estados internos para cada interação com o aplicativo externo.

Requisitos não funcionais:

- RNF01 - O sistema deverá enviar o log para o aplicativo a cada conexão com sucesso.
- RNF02 - O sistema deverá destravar as travas e permitir a retirada das marmitas apenas após a identificação do usuário através do aplicativo.
- RNF03 - O sistema deverá expandir a MTU(*Maximum Transmission Unit*) do BLE para 512 bytes.
- RNF04 - O sistema deverá ter três características disponíveis no *Bluetooth Low Energy*:
  - RNF04.1 - Uma característica de WRITE, para enviar um comando para o Aplicativo.
  - RNF04.2 - Uma característica de READ, para receber comandos do aplicativo.
  - RNF04.3 - Uma característica de NOTIFY, para enviar valores de forma contínua.
  - RNF05 - O sistema deverá ter um código único por freezer, composto de 5 caracteres (apenas letras e números).

3.3.2 Aplicativo

Os requisitos funcionais e não funcionais do aplicativo são:

Requisitos funcionais do aplicativo:

- RF01 - O sistema deverá permitir a criação de uma nova conta ou login.
- RF02 - O sistema deverá possuir um mapa dos freezers Tropicalia disponíveis na cidade do usuário.
- RF03 - O sistema deverá mostrar o cardápio de opções de marmitas disponíveis.
- RF04 - O sistema deverá mostrar o histórico de compras do usuário .
- RF05 - O sistema deverá permitir o cadastro ou seleção de uma forma de pagamento.
- RF06 - O sistema deverá permitir a realização da compra de uma marmita.

  - RF06.1 - O sistema deverá permitir a leitura de um QRCode.
  - RF06.2 - O sistema deverá enviar um comando ao freezer para pedir que as travas se abram.
  - RF06.3 - O sistema deverá receber o estado de aberto/fechado do freezer.
  - RF06.4 - O sistema deverá, ao receber o estado de fechado do freezer, realizar a leitura da quantidade de marmitas.
  - RF06.5 - O sistema deverá mostrar para o usuário a quantidade de marmitas e realizar o pagamento.
- RF07 - Para usuários do tipo 'admin', o sistema deverá permitir a calibração da balança.
- RF08 - Para usuários do tipo 'admin', o sistema deverá permitir a calibração das travas.

Requisitos não funcionais do aplicativo:

- RNF01 - O aplicativo deverá ser desenvolvido para a plataforma iOS.
- RNF02 - O aplicativo deverá ser desenvolvido de forma nativa, usando Swift com o Xcode.
- RNF03 - O aplicativo deverá conectar-se via Bluetooth Low Energy com o sistema embarcado.
- RNF04 - O aplicativo deverá seguir a arquitetura VIP para a construção das classes.

Figura 16 – *Wireframe* de baixa definição do fluxo de compra.

Fonte: Autoria própria

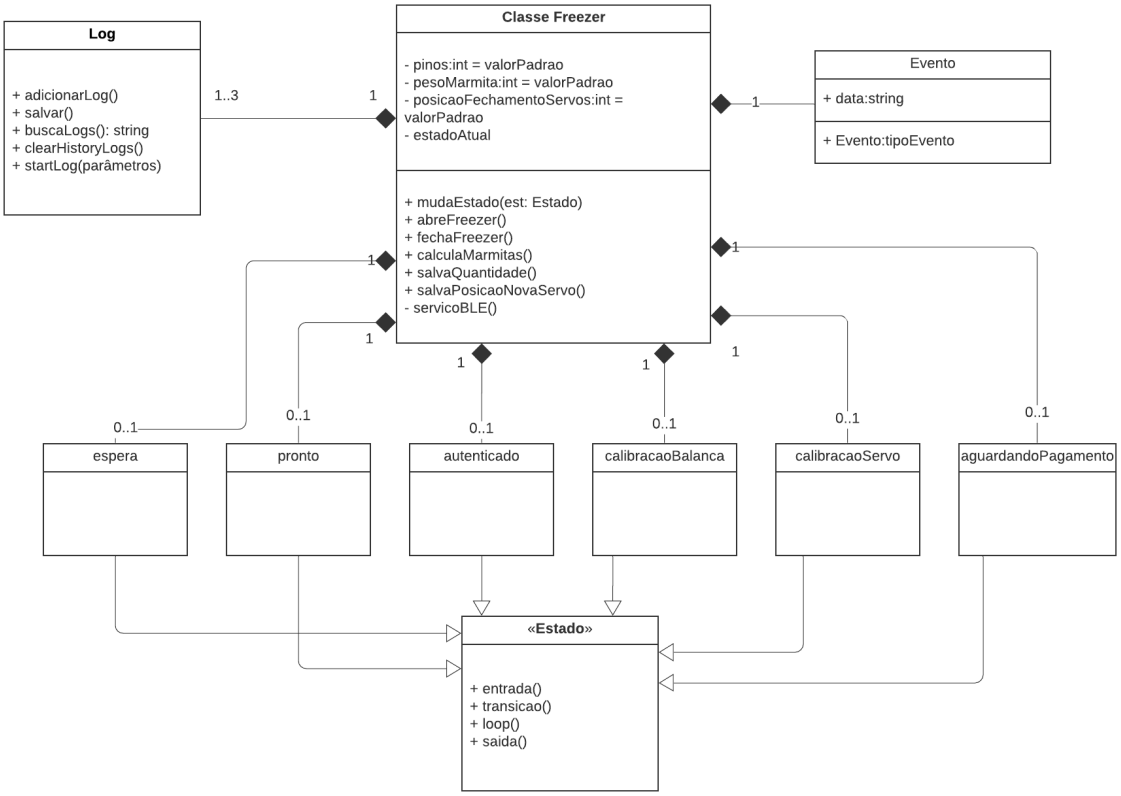
Na figura 16 pode-se ver o *Wireframe* de baixa definição do fluxo de compra no aplicativo. Na primeira tela, o usuário faz o leitura do QRCode utilizando a câmera do celular. Também é possível digitar o código único do freezer manualmente. Em seguida, a tela dá algumas intruções e apresenta um botão que, ao ser clicado, enviará o comando para o freezer destravar as portas. Em seguida, uma tela fica aguardando o usuário retirar quantas marmitas deseja e fechar o freezer. Ao receber a informação que as portas foram

fechadas corretamente, irá automaticamente para a tela de *checkout*, que mostra o valor total da compra (utilizando o cálculo automático da quantidade enviado pelo freezer) e a forma de pagamento selecionada. Ao finalizar a compra, o valor é cobrado do cartão do usuário.

3.4 Diagrama de classes

O diagrama de classes do sistema embarcado pode ser visto na figura 17. O diagrama de classes do aplicativo não foi realizado por causa da arquitetura e abordagem vistas nas seções anteriores.

Figura 17 – Diagrama de atividades do sistema embarcados.



Fonte: Autoria própria

4 IMPLEMENTAÇÃO

Nesta seção será detalhada a implementação do protótipo do freezer Tropicalia, o aplicativo para *iOS* e o banco de dados na nuvem. Os principais componentes do sistema, que serão detalhados a seguir, são:

- **Travas do freezer**: As travas que impedem a abertura do freezer são dos servo motores que mudam a posição para fechado ou aberto.
- **Balança**: É utilizada para o cálculo de quantas marmitas o usuário retirou do freezer.
- **Reed Switch**: O sensor de aproximação é o que verifica se as portas do freezer estão abertas ou fechadas.
- **Sistema Embarcado**: Utilizando a placa ESP32, faz o controle do hardware físico e comunicação com o aplicativo *iOS* através do *Bluetooth Low Energy*:
  - *Máquina de Estados*: Modelagem do sistema embarcado que permite a separação de responsabilidades.
  - *Comunicação dos Logs*: Como é feito o envio dos Logs do Sistema Embarcado para o aplicativo e, depois, para o banco de dados.
- **Aplicativo iOS**: É a interface pelo qual o usuário fará a compra. Por ele, é feita a leitura do QRCode, conexão bluetooth com o sistema embarcado e a comunicação com o banco de dados na nuvem.
- **Banco de Dados na nuvem**: Utilizando *Cloud Firestore*, são salvas as informações dos usuários, de compras e dos freezers, entre outros.

4.1 Adaptação física do Freezer

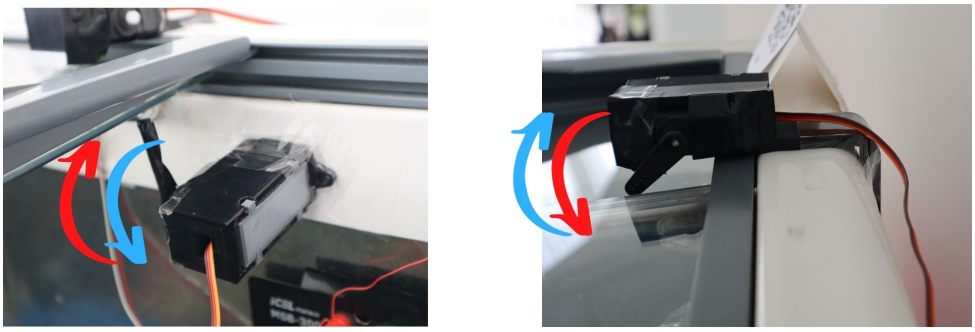
Para adaptação do freezer, foi utilizada uma abordagem minimalista, para que seja possível a instalação em qualquer freezer do segmento ou que tenha duas folhas de vidro sem prejudicar o funcionamento do mesmo. Na figura 18 observa-se o freezer já com a aplicação das modificações adaptativas. Na parte superior somente o que fica em evidência é o servo motor da folha de vidro superior, pois ele tem que estar posicionado desta maneira para não interferir na abertura ou fechamento da folha de vidro inferior. Os dois servo motores foram colocados segundo a figura 19. Desta forma, quando colocado em determinada posição controlada pelo microcontrolador, eles impedem fisicamente a abertura das folhas de vidro. As setas orientam o sentido: em azul está representada a abertura e as em vermelho o fechamento. O suporte em preto, que pode também ser visto na figura, sustenta os servo motores. Eles foram produzidos em impressora 3D e desenhados pelo ex-aluno da UTFPR Everton Barreto, e o arquivo do desenho está no Apêndice A.

Como o objetivo do projeto era um protótipo funcional, não foram realizadas

Figura 18 – Freezer com implementação.

Fonte: Autoria própria

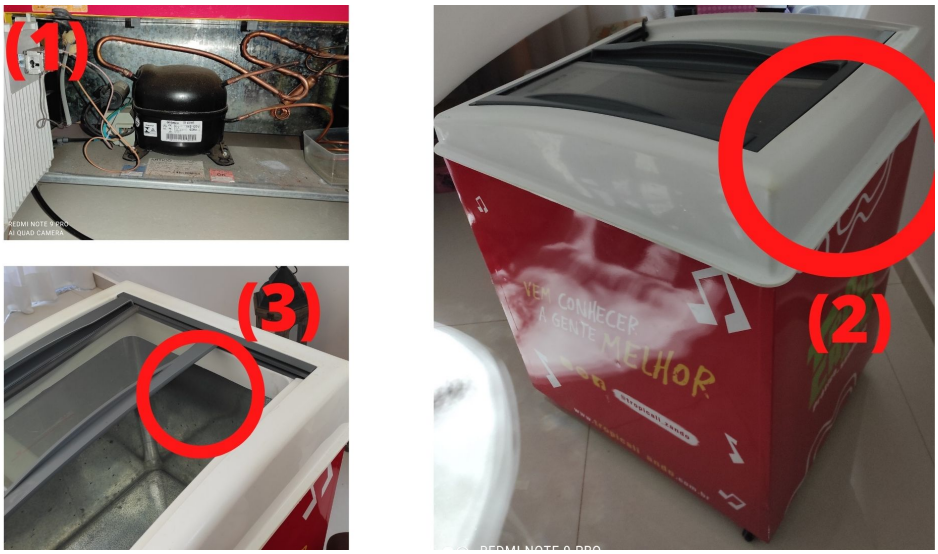
Figura 19 – Servo motores posicionados e em funcionamento. A seta azul mostra a posição de abertura e a vermelha, de fechamento.



Fonte: Autoria própria

a perfurações para fixação do servo motor na parte de dentro e nem aberto um orifício para passagem do barramento e dos fios que controlam os sensores e os servos. É possível instalar a placa, conforme a figura 20, dentro do compartimento motor(1), parte exterior posterior superior(2) e no interior na parte superior próximo ao servo motor interno(3). Esta última foi escolhida para aplicação do projeto.

Figura 20 – As posições possíveis de instalação da placa.



Fonte: Autoria própria.

4.2 Sistema embarcado

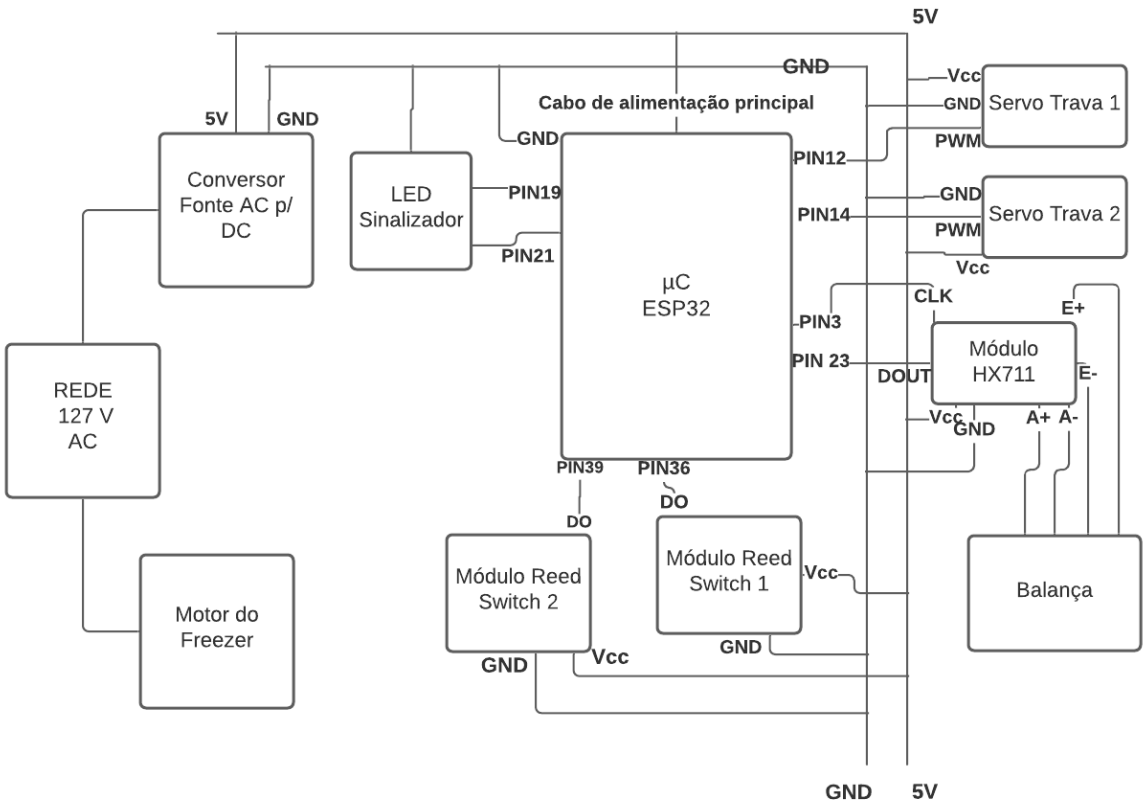
Nesta seção será apresentado o sistema embarcado que controla todo o hardware do sistema. A placa foi programada em C++, com plugin que simula a Arduino IDE dentro do *Visual Studio Code*. De forma geral, o sistema embarcado controla as seguintes partes do sistema:

- travas;
- balança;
- sensores reed switch;
- LEDs sinalizadores;
- comunicação com o aplicativo via bluetooth;
- contabilização das marmitas.

4.2.1 Esquemático

A instalação do circuito integrado para funcionamento do freezer automatizado pode ser vista na figura 21. Toda a parte desde o conversor de entrada 127V AC e saída 5V DC foi implementada para este protótipo. É possível também observar como as ligações são realizadas e o barramento dos 5V e do GND, as quais oferecem a alimentação e o negativo respectivamente para os demais componentes presentes no circuito. O controle de sinal *PWM* para os servo motores vêm da placa, porém a alimentação é proveniente do barramento externo que tem mais corrente para o acionamento dos motores. As portas do microcontrolador foram escolhidas levando em conta as especificações da fabricante da ESP32, conforme foi visto na seção 2.1.

Figura 21 – Esquemático do freezer automatizado.



Fonte: Autoria própria

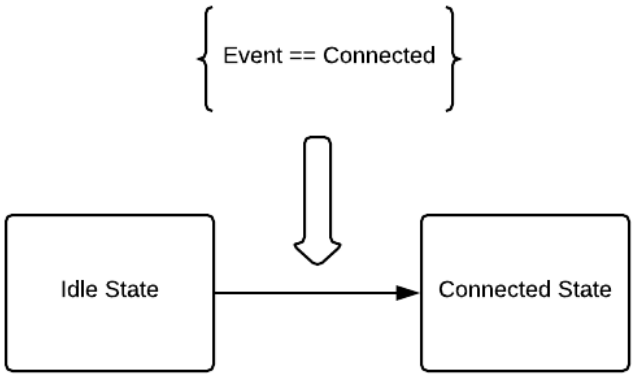
4.2.2 Máquina de estados

Uma máquina de estados é uma maneira de modelar o sistema como um conjunto de estados específicos com transições entre eles (FOWLER, 2010). A transição para um estado depende de duas informações: o estado atual e um evento do sistema.

O sistema embarcado foi modelado com estados, na linguagem C++, utilizando de inspiração a máquina de estados descrita por HOVHANNISYAN. A transição entre estados acontece a partir de um evento específico, que pode ser interno ou externo. Na figura 22 podemos ver um exemplo de transição entre dois estados. O estado *Idle* é quando o freezer está apenas esperando algo acontecer, sem fazer nada. Ao receber uma conexão via **Bluetooth** pelo aplicativo, ele envia um evento (do tipo interno) para o estado atual, que é o *Idle*. Esse estado, ao processar este evento, entende que precisa realizar uma transição para o estado *Connected*.

Um evento é uma `struct` que contem dois parâmetros: o tipo do evento e uma `string` de dados. Quando o evento é externo (ou seja, vindo do aplicativo através do BLE) ele virá com um JSON que contém a mensagem enviada. O evento é representado pelo

Figura 22 – Exemplo de uma transição de estados



Fonte: Autoria própria

código abaixo, de forma simplificada:

```
struct Event {
  public:
    stateEventType eventType; // External, Connected,
       Disconnected, Reset
    std::string data; // string de dados do evento
};
```

Existe uma classe 'central' que gerencia os estados, chamada Freezer. Ela é responsável por saber qual é o estado atual, receber os comandos externos e internos e repassar ao estado atual, e realizar as transições entre estados. De maneira simplificada, a seguir podemos observar a interface dessa classe em relação à máquina de estados:

```
class Freezer {
  public:
    // realiza a transi  o de um estado para outro
    void setState(FreezerState *newState);
    // recebe eventos e repassa ao estado atual
    void transition(Event event);
    ...
  private:
    // guarda o estado atual
    FreezerState* _currentState;
    ...
```

```
12 };
```

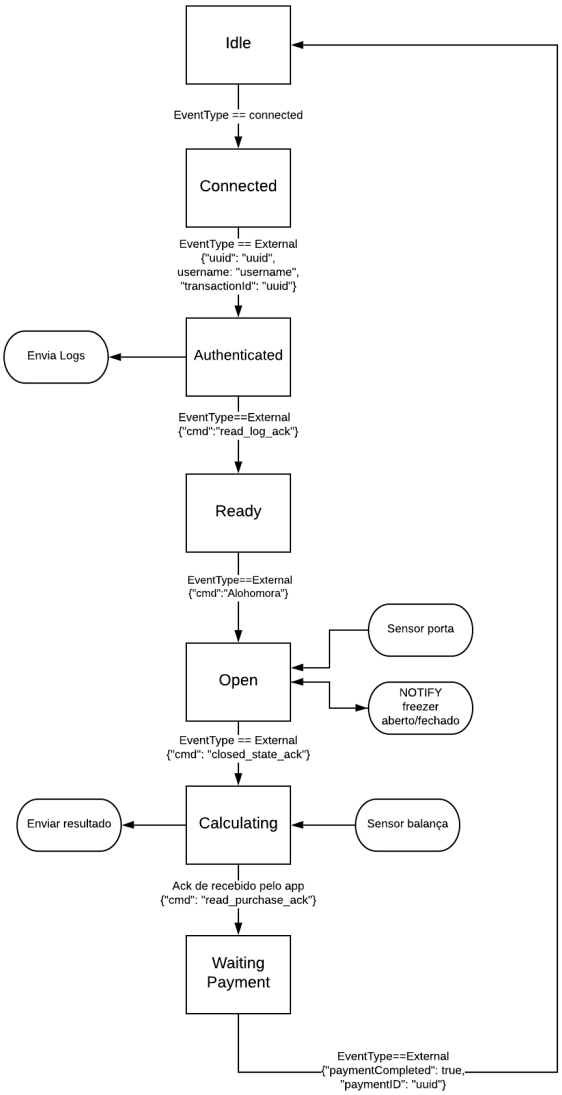
Nesta máquina de estados, cada estado é uma classe concreta específica, que faz uma herança da classe virtual `FreezerState`. Isso permite que cada estado tenha sua própria lógica completamente independente dos outros estados, o que segue os princípios de separação de responsabilidades, em que cada classe tem uma função específica.

Por ser definida como uma classe virtual com funções vazias, cada estado pode implementar, das funções disponíveis, apenas as que fazem sentido para seu contexto. A única função obrigatória para um estado é a `transition()`, já que o estado é o único que sabe qual é o próximo estado dado um evento específico. A representação simplificada de um estado é dada a seguir:

```
1  class FreezerState
2  {
3    public:
4      // Funciona como um setup de cada estado
5      virtual void enter(Freezer* freezer) {};
6      // o que o estado precisa limpar ou fazer antes de sair
         deste estado
7      virtual void exit(Freezer* freezer) {};
8      // fun   o loop tradicional do Arduino
9      virtual void loop(Freezer* freezer) {};
10     // dado um evento, o estado decide qual sera o proximo
         estado
11     virtual void transition(Freezer* freezer, Event event) =
         0;
12 }
```

```
1  class FreezerState
2  {
3   public:
4    virtual void enter(Freezer* freezer) {};
5    virtual void exit(Freezer* freezer) {};
6    virtual void loop(Freezer* freezer) {};
7
8    virtual void transition(Freezer* freezer, Event event) = 0;
9  }
```

Figura 23 – Máquina de estados.



Fonte: Autoria própria.

Os estados do Freezer Tropicalia estão, de maneira geral, diretamente relacionados a um fluxo de compra. A relação entre os estados está descrita na figura 23. A função de cada um dos estados é:

- `Idle State`: Estado 'parado' em que o freezer está aguardando uma conexão.
  - Transição: Evento Interno `Connected` leva ao `Connected State`.
- `Connected State`: Estado no qual o *Bluetooth* foi conectado.
  - Transição: Evento Externo contendo `userID` e `transactionID` leva ao `Ready State`.
- `Authenticated State`: Usuário está autenticado. Neste estado será feita a troca dos logs (detalhado na sessão 4.2.6).
  - Transição: Evento Externo contendo `userID` e `transactionID` leva ao `Ready`

`State`.

- `Ready State`: Usuário está autenticado e o hardware está pronto para receber o comando de destravar as portas.
  - Transição: Evento Externo contendo `cmd = 'alohomorra'` leva ao `Open State`.
- `Open State`: As portas do freezer são destravadas. Lê e envia o status de aberto/fechado para o aplicativo via `notify`.
  - Transição: Evento Externo contendo `cmd = 'closed_state_ack'` leva ao `Calculating State`.
- `Calculating State`: É realizada a leitura da balança e calculado quantas marmitas foram retiradas. Esta contagem é então enviada para o aplicativo.
  - Transição: Evento Externo contendo `cmd = 'read_purchase_ack'` leva ao `Waiting Payment State` (aguardando o pagamento).
- `Waiting Payment State`: É aguardado o sinal de que o pagamento foi realizado com sucesso.
  - Transição: Evento Externo contendo `cmd = 'payment_completed'` leva ao `Idle State`.

4.2.3 Balança

Uma etapa essencial do fluxo de compra proposto pelo projeto é o cálculo automático de quantas marmitas foram retiradas pelo usuário. Considerando que as marmitas vendidas pelo Freezer têm sempre um peso conhecido e constante (aproximadamente 250g), é possível pelo cálculo do peso total sobre a balança, inferir essa quantidade.

Figura 24 – Balança de perfil e aplicada dentro do freezer.

Fonte: Autoria própria.

Na figura 24 pode-se observar, à esquerda, a balança de perfil já montada e com

as folhas de madeira, onde a parte superior recebe as cargas das marmitas para a pesagem através da sensibilização do sensor. E à direita, a balança já dentro do freezer, onde foram realizados os testes do protótipo. O produto em produção real deverá ter uma balança para cada tipo de marmita diferente, que pode contabilizar quantas marmitas estão sobre a balança daquele tipo e cobrando corretamente do cliente quanto retirar as marmitas de dentro do freezer. As marmitas em um ensaio real, têm uma faixa máxima de diferença entre o peso em gramas. Por exemplo, uma marmita que idealmente deveria ter 250 gramas, tem uma faixa entre 245 a 255 gramas. Para solucionar isso foi utilizada uma função de arredondamento que será explicada mais adiante nesta seção. Importante lembrar que no caso do projeto, não foram discriminados diferentes tipos de marmitas. Foi considerado que todos os tipos tem a mesma quantidade aproximada de peso em gramas, como visto no exemplo anterior.

Na parte da programação no sistema embarcado foi utilizado a biblioteca HX711, que deu o suporte necessário para ler os dados do sensor da balança. O objeto `balanca` (que representa a balança) é instanciado no começo do programa com um fator de calibração padrão, que pode ser calibrado posteriormente como será visto nas próximas seções. Com este fator calibrado, é possível calcular a quantidade de marmitas presentes com precisão. Quando necessário, é chamada uma função da classe `Freezer`, que retorna a quantidade das marmitas sobre a balança. As funções relevantes são estas:

```
1  int calculaMarmitas()
2  {
3      //le o sensor da balanca
4      float peso_total = balanca.get_units();
5      // calcula a quantidade de marmitas no freezer
6      double quantMarmitas = peso_total/peso_por_marmita;
7
8      return (arredonda(quantMarmitas));
9  }
10
11 void salvaQuantidade() {
12     quantidadeAntesAbertura = calculaMarmitas();
13 }
14
15 int calculaMarmitasRetiradas() {
16     return (quantidadeAntesAbertura - calculaMarmitas());
17 }
18
19 int arredonda(double numero) { ... }
```

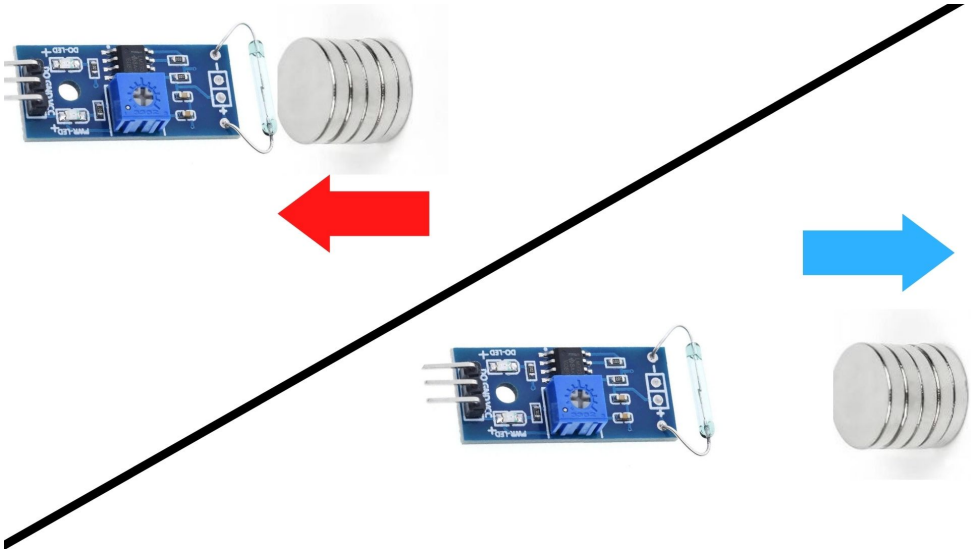
A função `balanca.get_units()`, que é da biblioteca mencionada acima, retorna o valor lido da balança em gramas. A função que a utiliza é a `int calculaMarmitas()`, que calcula, segundo o peso de cada marmita, quantas estão atualmente em cima da balança. O objetivo das demais funções auxiliares são:

- `void salvaQuantidade()`: guarda quantidade de marmitas antes de abrir o freezer para o usuário.
- `int calculaMarmitasRetiradas()`: compara com a quantidade salva anteriormente com a quantidade atual para saber quantas marmitas o usuário levou.
- `int arredonda()`: arredonda a quantidade para o número inteiro mais próximo, pois o número de marmitas são inteiros.

Os estados que se utilizam destas funções são: **CalculatingState** e **CalibrationState**. O primeiro faz a contabilização das marmitas que o usuário cliente tirou do freezer, chamando as funções mencionadas acima. O segundo, será destalhado na seção 4.4.

4.2.4 Reed Switch

Figura 25 – Exemplo do funcionamento do módulo reed switch.



Fonte: Autoria própria.

Os módulos *Reed Switch* servem para identificar se ambas as folhas de vidro do freezer estão abertas ou fechadas. Nestas foram colocados ímãs, posicionados de tal forma que, quando as folhas se fecham, são identificados pelos sensores *Reed Switch* conforme a figura 25. À esquerda o *reed switch* está com o circuito fechado e à direita está aberto, sinalizando '0' e '1' respectivamente. O sistema embarcado lê as portas referentes aos módulos dos sensores individualmente conforme a figura 21, através do comando `digitalRead(PIN_SENSOR)`. Quando o módulo retorna '0' significa que a folha de vidro referente àquele módulo está fechada. Desta forma, quando o cliente pede para abrir o

freezer, o módulo avisa ao sistema se as folhas de vidro foram fechadas e é seguro fechar as travas, para contabilizar as marmitas e prosseguir para o fluxo de compra. Resumindo temos a tabela 5.

Tabela 5 – Relação leitura módulo versus situação real

	Digital Out	folha de vidro superior	folha de vidro inferior
Módulo Reed Switch 1	1	aberta	X
	0	fechada	X
Módulo Reed Switch 2	1	X	aberta
	0	X	fechada

Fonte: Autoria própria.

4.2.5 Trava servo motor

A biblioteca utilizada foi a *ESP32Servo*, como visto anteriormente neste documento(2.1), que traduz a posição desejada do servo em pulsos que a placa emite. O controle dos servos é o que garante a segurança de que o freezer seja fechado fisicamente, impedindo o acesso antes da liberação por um cliente(19). Durante a calibração são gravadas as posições que os servos fecham as folhas de vidro, que serão descritas na próxima seção. A posição que os abre é a posição '0' em ambos. Os estados que chamam as funções públicas do objeto **Freezer** são estas:

```
1  void Freezer::openFreezer() {
2      //open servos to client
3      lock2.write(0);
4      delay(300);
5      lock1.write(0);
6      delay(300);
7  }
8
9  void Freezer::closeFreezer() {
10     //close servos to client
11     lock2.write(lockedPosition1);
12     delay(300);
13     lock1.write(lockedPosition2);
14     delay(300);
15 }
```

Note que os objetos instanciados como `lock1` e `lock2` são os servos do projeto. A função `write()` que cada servo chama é a função da biblioteca que codifica a posição desejada.

A função `openFreezer()` coloca ambos servos na posição **0** enquanto que na função `closeFreezer()` coloca-os na posição nas quais eles fisicamente fecham as folhas de freezer. Estas posições estão contida nas variáveis `lockedPosition1` e `lockedPosition2` e são obtidas após calibragem dos servos.

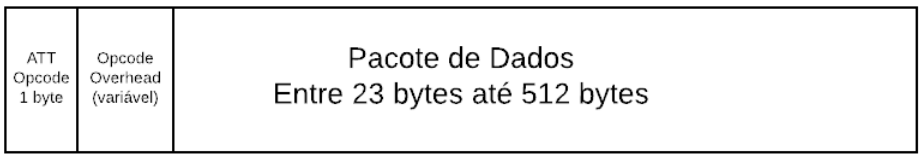
4.2.6 Comunicação dos Logs

A transmissão dos logs é importante para a verificação de usuários que possivelmente fraudaram o sistema, ou seja, usuários que retiraram as marmitas e não realizaram o pagamento. Para que o usuário possa fazer uma compra é requerido que exista uma conexão com a internet - através dela é coletada as informações do Freezer e realizado o pagamento no final. Portanto, o sistema embarcado utiliza a internet do dispositivo para enviar os logs para o aplicativo, que então envia para o banco de dados, ao invés do sistema embarcado estar diretamente conectado com a internet. Isso possibilida uma maior flexibilização dos locais em que um Freezer Tropicalia poderia ser instalado, já que possuir internet constante não é um requisito.

Um detalhe importante é que o pagamento nem sempre irá acontecer ao alcance do **Bluetooth**, já que um usuário pode sair do local do freezer enquanto realiza o pagamento. Isso significa que nem sempre o sistema embarcado receberá a confirmação que a compra foi finalizada. Por isso, esses registros ainda dependem do cruzamento das informações entre os registros de compras realizadas para descobrir potenciais problemas. Assim que os logs são salvos no banco de dados com sucesso, eles são deletados do sistema embarcado para otimizar a memória interna.

O Protocolo de Atributo (ATT) do BLE define como é feita a conexão e transmissão de dados entre cliente e servidor. Neste protocolo também é definido o tamanho máximo do pacote, também chamado de Máxima Unidade de Transmissão, ou **MTU** (*Maximum Transmission Unit*), que é o tamanho do pacote permitido por cada envio de dados utilizando o *Bluetooth Low Energy*. Por padrão, esse tamanho limite é de no máximo 23 bytes (COLEMAN, 2019), o que é um limite baixo, considerando que a transmissão é feita em caracteres ASCII. Na figura 26 pode-se observar o formato do pacote de transmissão de dados via BLE.

Figura 26 – Pacote de transmissão de dados no BLE



Fonte: (COLEMAN, 2019)

Pela especificação do *Bluetooth Core*, o máximo tamanho do atributo é de 512 bytes. Esse aumento no tamanho padrão do MTU é negociado no início da conexão, dado que ambos os lados suportem este tamanho de pacote. Porém, mesmo com 512 bytes, ainda não seria suficiente para enviar todos os logs de uma só vez, já que existe a possibilidade de alguns terem sido acumulados. Por isso, foi desenvolvida uma forma de comunicação em que os logs são passados aos poucos ao aplicativo móvel.

Cada autenticação de usuário é feita utilizando o `userId` e uma `transactionId`, que são utilizados para o registro de cada um dos logs do que aconteceu no sistema. A primeira etapa da transmissão dos logs é enviar ao aplicativo quantos logs serão transmitidos no total e quais os `transactionId` dos logs que serão transmitidos:

```
{
    "size":3,
    "transactions":[ "0ewmD", "23kl4", "wer4t" ]
}
```

Cada log possui a identificação da transação que ocorreu, com os identificadores de usuário, transação e horário. Além disso, loga quais os estados foram acionados e a quantidade retirada de marmitas, caso ela tenha sido calculada. O formato de um log é, em JSON:

```
{
    "user":"EtvEzf1lGTSvVnI5kMaodqPRm8t2",
    "transaction":"0ewmD",
    "time":"12/04/2021 19:34",
    "quantity": 4,
    "states": [
        "authenticated",
        "ready",
        "open",
        "calculating",
        "waiting_payment",
        "received_payment"
    ]
}
```

Em seguida, os logs são enviados, sequencialmente, através da propriedade **notify** do Bluetooth. Essa propriedade é do tipo *fire-and-forget*, o que significa que o pacote é enviado mas não exite certeza se o aplicativo recebeu a informação. Dessa forma, os logs

são enviados um a um de forma contínua, e cabe ao aplicativo avisar que recebeu todas as informações, baseadas na primeira transmissão que diz exatamente quais logs são enviados. Isso torna a transmissão mais ágil, já que utilizar a propriedade de enviar mensagens exige um tempo maior entre cada mensagem diferente a ser enviada e, caso algum pacote seja perdido na transmissão, existe o risco dele ser deletado sem que tenha sido salvo no banco de dados.

Do lado do aplicativo, é utilizada uma estrutura de *Dictionary* (ou *HashMap*) para verificar quando todas as mensagens foram recebidas, conforme descrito no pseudo-código a seguir:

```
var allLogs: [String: Log] = [:]

func receivedLog(log: Log) {
    allLogs[log.transactionId] = log

    if allLogs.keys.count == allTransactions.count {
        // foram recebidos todos os logs, portanto pode-se
           enviar
        sendReadLogsAck()
    }
}
```

A partir dessas informações, é possível cruzar os dados com as informações salvas das transações concluídas no banco de dados na nuvem. Embora em algum momento o sinal de pagamento completo não tenha sido recebido pelo sistema embarcado, se o pagamento foi realmente recebido existirá o registro dele na nuvem, com o mesmo `userId` e `transactionId`. Assim, pode-se verificar quais são as transações que possivelmente foram fraudulentas. Junto com o horário e usuário cadastrado, seria possível cruzar com informações de câmera e cobrar diretamente ao cartão de crédito cadastrado inicialmente pelo usuário.

4.3 Aplicativo Móvel

O aplicativo iOS é a interface pela qual o usuário irá desbloquear o Freezer e realizar a compra de uma marmita. As funcionalidades do aplicativo podem ser resumidas nos pontos a seguir, que serão detalhados nas próximas seções.

- Cadastro, Login e recuperação de senha
- Tela principal
  - Mapa dos freezers disponíveis
  - Cardápio das opções de marmitas disponíveis

  - Botão para iniciar a compra com QRCode
  - Histórico de compras
  - Configurações
- Fluxo de Compra
  - Leitura do QRCode
  - Cadastro da forma de pagamento, caso o usuário não possua nenhuma cadastrada
  - Abertura do Freezer
  - Aguardando o usuário retirar as marmitas
  - Finalização da compra
- Configurações
  - Dados da Conta (telefone e dados pesssoais)
  - Gerenciar formas de pagamento
  - Contato
  - Sobre nós
  - Logout
- Configurações de administrador
  - Calibração da balança
  - Calibração das travas
  - Abrir as travas
  - Cadastrar Freezer
  - Geração dos QRCodes

4.3.1 Visão Geral

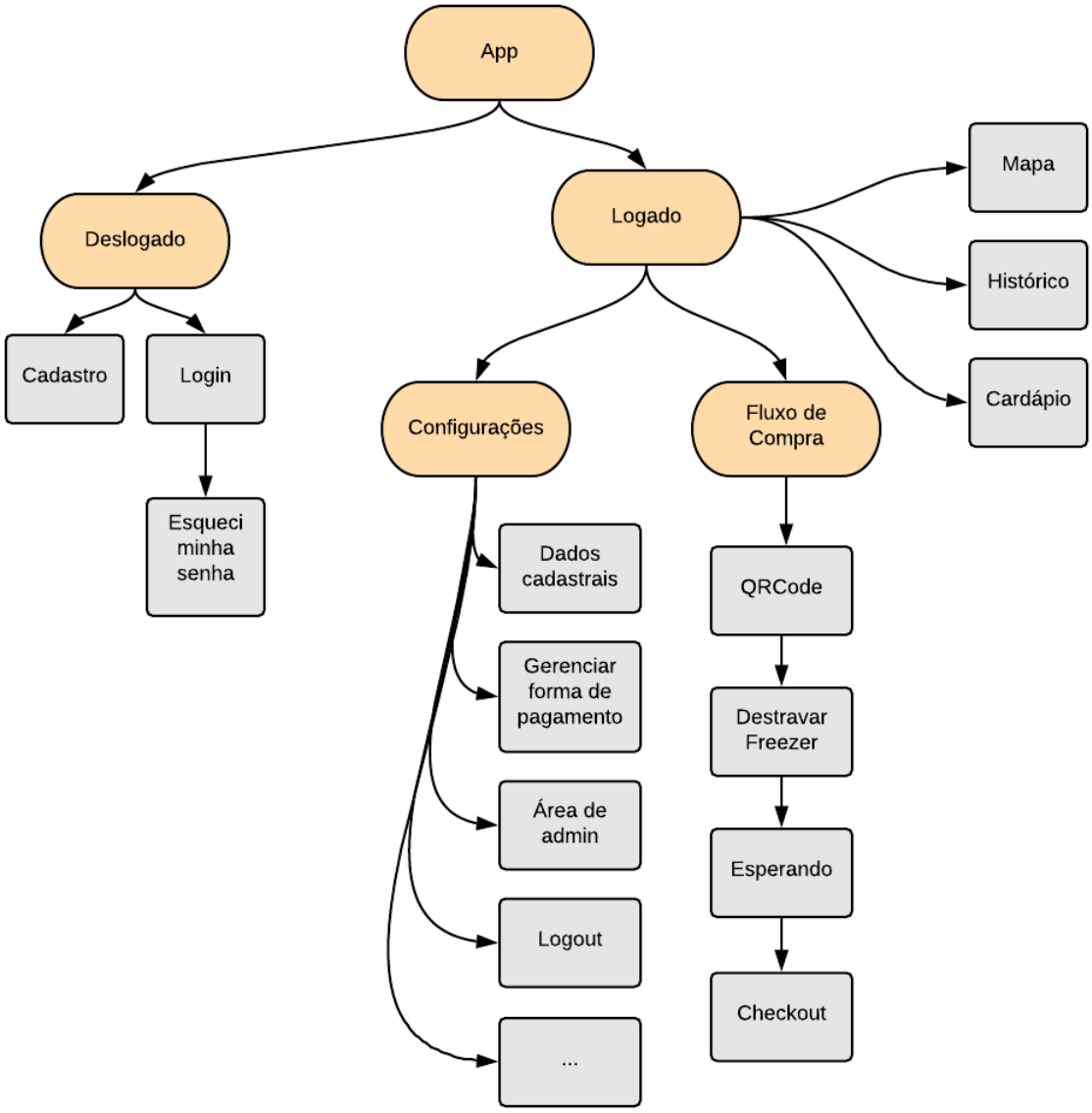
A navegação entre as telas são feitas como através de *Coordinators*. Cada *coordinator* gerencia as telas que fazem parte do contexto ou, quando necessário, instanciam outro *Coordinator* para que gerencie um fluxo interno. Na figura 27 pode-se observar, em formato de árvore, a estrutura simplificada do **app** em relação à navegação dos fluxos. Os pontos em laranja são os *Coordinators* e, em cinza, cada 'cena' de tela.

Cada uma dessas cenas é composta de 4 classes:

- `View`: Código apenas do layout da tela
- `ViewController`: Padrão do iOS, apresenta a View e faz ligações de ações para o Interactor.
- `Interactor`: Classe com a lógica de negócios. Faz a conexão com o Service e com o BLE.
- `Builder`: É a classe que constrói todas as outras classes e faz a ligação das referências entre elas. Conhece os comandos JSON a serem enviados e retorna os resultados recebidos.

Compartilhados entre algumas cenas existem as dependências. Por exemplo, ao

Figura 27 – Estrutura de árvore das telas do aplicativo.



Fonte: Autoria própria

invés de um `Interactor` chamar diretamente o banco de dados na nuvem ou do *Bluetooth*, são criadas classes intermediárias que abstraem a maior parte da lógica. Assim, segue-se o princípio de separação de responsabilidades e torna-se muito simples escrever testes unitários para cada uma dessas classes, pois estão atrás de uma interface. Alguns exemplos de classes de serviço são:

- `AuthService`: É responsável pela autenticação do usuário.
- `TropicaliaService`: É responsável pelo acesso ao banco de dados no *Firebase*; cria, modifica e obtém informações dos registros.
- `BluetoothWorker`: É responsável por manter a conexão Bluetooth. Conhece os

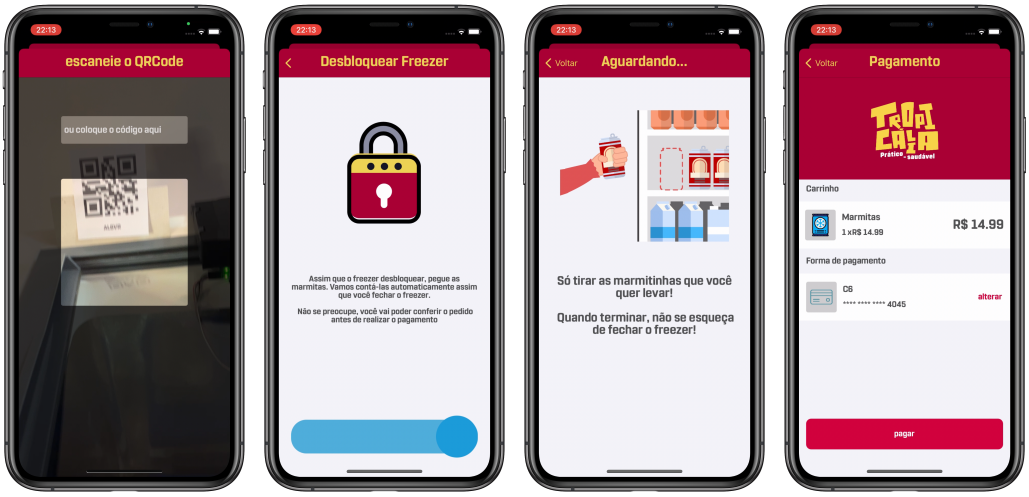
comandos a serem enviados e retorna as mensagens recebidas.

4.3.2 Fluxo de compra

Na figura 28 pode-se ver a captura das telas do aplicativo para o fluxo na realização de uma compra. Este fluxo faz o desbloqueio do freezer e permite ao usuário realizar a compra de marmitas, e tem 4 telas, cuja função principal está relacionada a seguir:

- **Leitura do QRCode**: Lê o QRCode (que é o ID único do freezer) e faz a requisição das informações completas das características bluetooth para realizar a conexão com o Freezer.
- **Destravar o freezer**: Tem um botão do tipo *slide* que envia o comando de destravar para o Bluetooth. Também recebe e envia os Logs para a nuvem.
- **Aguardando**: Aguarda o usuário fechar o freezer. Recebe essa informação (de aberto ou fechado) via propriedade NOTIFY. Quando fechado, envia um comando de *ack* de volta para o freezer.
- **Finalização do pagamento**: Recebe a quantidade de marmitas retirada pelo usuário. Ao finalizar a compra, chama o processamento de pagamento e salva o resultado final do processo.

Figura 28 – Captura de tela do fluxo de compra.



Fonte: Autoria própria

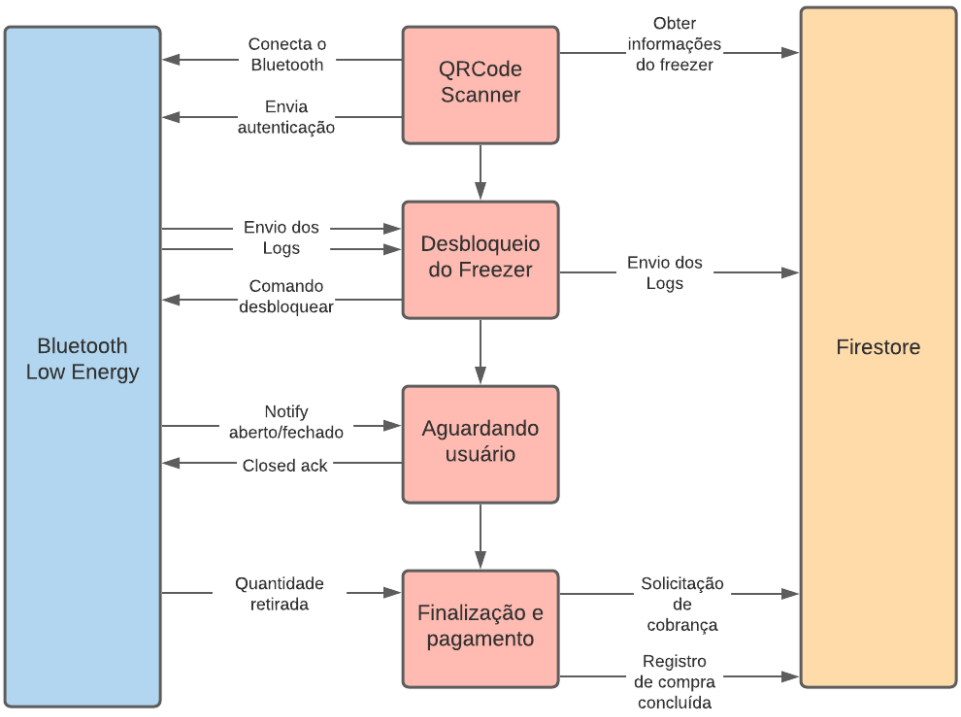
Para este protótipo, não foi feita a conexão com uma empresa que realiza as transações reais. Foi realizado um teste com um cartão teste. Para processamento de pagamento com cartão de crédito de verdade seria necessário a integração com um SDK de processamento de pagamentos. Neste caso, as informações de dados de cartão de crédito são guardadas apenas pela empresa que faz o processamento do pagamento e, no banco do **app**, apenas podem ser salvas algumas poucas informações sobre este método de pagamento.

Neste fluxo, logo após a leitura do QRCode, caso o usuário ainda não tenha um método de pagamento cadastrado, é mostrada a tela de "Gerenciador de métodos de pagamentos". Esta tela também é reutilizada nas configurações.

Uma conexão bluetooth não se mantém aberta por muito tempo se nenhuma troca de informações acontecer. A informação do freezer estar aberto ou fechado é enviado via propriedade *notify* justamente para mitigar este problema. Como é esperado que o usuário demore um pouco para escolher as marmitas, enviar de forma contínua o aberto/fechado mantém a conexão aberta durante este período.

Na figura 29 é possível observar, de forma visual, o que cada uma das telas faz em relação a sua conexão com o Banco de Dados na nuvem (*Firestore*) e ao sistema embarcado via *Bluetooth Low Energy*.

Figura 29 – Fluxo de informações nas telas de compra.



Fonte: Autoria própria

4.3.3 Configurações

A última página principal do aplicativo é a tela de "Configurações". Nela o usuário pode configurar sua conta, como alterar os seus dados pessoais, configurar métodos de pagamento ou entrar em contato com a equipe de suporte. Na figura 30 estão as capturas de tela da parte de configurações do aplicativo, tanto de um usuário comum quanto de um usuário administrador, que tem mais funcionalidades disponível.

Figura 30 – À esquerda, captura de tela das configurações comum e, à direita, de um usuário administrador.

Fonte: Autoria própria

Alguns usuários, denominados *Usuários administradores*, têm acesso a algumas funções a mais no aplicativo. No caso real do Freezer estar funcionando em locais distribuídos em uma cidade, embora não precise de um funcionário para realizar todas as vendas, seria necessário alguém para realizar a 'operação' dos freezers. Isso significa repor o estoque de marmitas no freezer, cadastrar novos freezer ou calibrar os sensores caso necessário.

As funcionalidades que um usuário adminstrador tem acesso são:

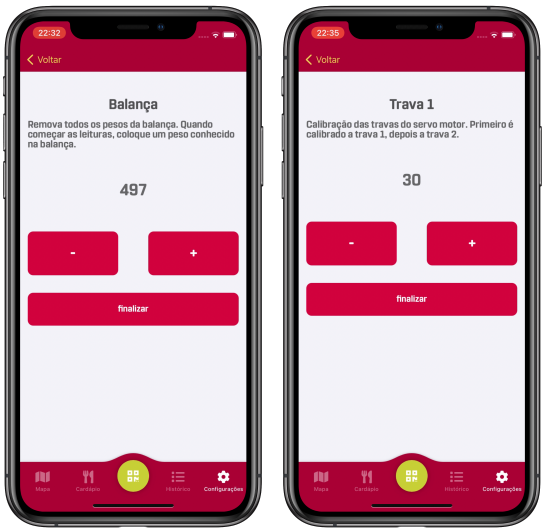
- **Cadastrar Freezer**: Quando é colocado um Freezer em um novo local, é preciso fazer o cadastro dele. Esta tela faz a geração automática do ID do freezer, possui um seletor de endereço para o local no mapa e cadastra o novo freezer no banco de dados na nuvem.
- **Geração dos QRCodes**: Utilizando um código nativo da linguagem Swift, esta tela gera os QRCodes que seriam impressos para cada freezer, que inicia o processo de compra. O QRCode é simplesmente o ID do freezer codificado.
- **Abrir as travas**: Como o circuito está dentro do próprio freezer, este comando permite abrir ou fechar as travas das portas.
- **Calibração da balança**: Embora a balança tenha um fator de calibração relativamente previsível, ela ainda pode ficar descalibrada. Está explicado em mais detalhes na seção 4.4.
- **Calibração das travas**: As travas tem o valor padrão em aberto, e a trava precisa fechar na posição certa para travar as portas do freezer. Também foi explicado em mais detalhes na seção 4.4.

4.4 Calibração da balança e das travas

A calibração tanto dos servos quanto da balança é acessível através do *usuário admistrador*, pois esse tem acesso aos estados de calibração `ScaleCalibration` e o `ServoCalibration`. Este usuário manda as informações para o sistema embarcado, que por sua vez controla as modificações. A calibração da balança é feita da seguinte forma:

- Retiram-se todas as cargas que estão em cima da balança.
- O usuário com esse nível solicita a calibração da balança via painel específico no aplicativo.
- É chamada uma função que tara a balança.
- Coloca-se uma carga conhecida sobre a balança.
- Modifica-se o valor do fator de calibragem da balança via aplicativo conforme a foto da tela à esquerda da figura 31. O valor da carga medida é mostrada em gramas.
- Quando o número medido pelo sistema embarcado, mostrado a esse usuário administrador se iguala ao real, previamente conhecido, a calibração foi realizada com sucesso.
- Ao clicar em finalizar, este número é guardado para ser utilizado em leituras posteriores.

Figura 31 – À esquerda, captura de tela da calibração da balança e, à direita, das travas.



Fonte: Autoria própria

A calibração dos servos é análoga:

- O usuário de nível específico solicita a calibragem das travas via aplicativo.
- O sistema embarcado chama a função que abre o freezer que coloca os servos na posição "0".
- Cada servo é calibrado por vez, conforme a tela à direita da figura 31.

- É ajustada a posição do servo, segundo os comandos "+"ou -", até que fisicamente ele impeça fisicamente a abertura da sua respectiva folha de vidro.
- Depois de finalizada a calibragem, a informação destas posições são guardadas, pelo sistema embarcado, para serem usadas toda vez que for chamada a função `closeFreezer()`.

4.5 Banco de dados na nuvem

Para o banco de dados na nuvem foi utilizado o *Cloud Firestore*. Por ser um banco de dados *NoSQL*, as tabelas não são relacionais, em que cada elemento tem um ID único que é utilizado para recuperar as informações.

Na nomenclatura do *Firestore*, um **Document** é o objeto em si - por exemplo, o Freezer ou o Usuário. Uma **Collection** é uma coleção, ou array, de documentos. Por exemplo, a coleção de todos os usuários, ou a coleção de todas as compras realizadas dentro de um usuário.

Os tipos de dados registrados no banco de dados no *Firestore* são:

- **Freezer**:
  - Freezer ID
  - BluetoothInfo (Informação dos UUIDs das características e como fazer a conexão com aquele Freezer específico)
  - Localização
  - Nome do Freezer
- **Usuários**:
  - User ID
  - Email
  - Telefone
  - Roles (Admin ou usuário)
  - ID do pagamento principal selecionado
  - (Collection) Formas de pagamento cadastradas
  - (Collection) Histórico de compras
- **Marmitas**:
  - Título
  - Descrição
  - URL da Imagem
- **Logs**:
  - ID
  - Timestamp
  - User ID
  - Transaction ID

- Quantidade retirada
- (Collection) Estados

5 GESTÃO

5.1 Custos

Aqui serão abordados os custos referentes ao projeto. Como um dos objetivos do projeto é oferecer uma solução alternativa às *Vending Machines* tradicionais, é importante que seja analisado os custos totais e parciais juntamente com os comparativos.

Tabela 6 – Custos do projeto.

Item	**Custo**
freezer 120l Ártico	R$ 1700,00
2 servo motor TowerPro	R$ 78,00
placa ESP32	R$ 39,00
2 módulos reed switch	R$ 10,00
2-6 balanças, dependendo da aplicação	R$ 180,00
2 ímãs	R$ 20,00
licença do aplicativo iOS	R$ 440,00
parafusos	R$ 25,00
suporte para as travas	R$ 150,00
fios para barramento	R$ 20,00
Custo total do projeto	R$ 2.662,00

Fonte: Autoria própria.

Tabela 7 – Custos de um freezer automatizado.

Item	**Custo**
freezer 120l Ártico	R$ 1700,00
2 servo motores TowerPro	R$ 78,00
placa ESP32	R$ 39,00
2 módulos reed switch	R$ 10,00
2-6 balanças, dependendo da aplicação	R$ 180,00
2 ímãs	R$ 20,00
parafusos	R$ 25,00
suporte para as travas	R$ 150,00
fios para barramento	R$ 20,00
Custo total aproximado do freezer automatizado	R$ 2.222,00

Fonte: Autoria própria.

A tabela 6 mostra o custo total que foi necessário para o projeto implementar o protótipo funcional. Já a tabela 7 é o custo de uma unidade de freezer automatizada. Por fim a tabela 8 mostra o custo que seria preciso para instalar a automatização, caso a pessoa já tivesse adquirido o freezer.

Tabela 8 – Custos de instalação.

Item	Custo
2 servo motores TowerPro	R$ 78,00
placa ESP32	R$ 39,00
2 módulos reed switch	R$ 10,00
2-6 balanças, dependendo da aplicação	R$ 180,00
2 ímãs	R$ 20,00
parafusos	R$ 25,00
Suporte para as travas	R$ 150,00
fios para barramento	R$ 20,00
Custo total aproximado do freezer automatizado	R$ 522,00

Fonte: Autoria própria.

É importante ressaltar que para o projeto virar um produto mínimo viável, é necessário algumas adaptações que irão encarecer o projeto. Também destaca-se a possibilidade de compra somente da instalação. Esta é uma opção para quem já tem o freezer e deseja fazer a instalação, somente com os outros componentes, deixando o produto final muito mais barato do que adquirindo o freezer novo, conforme a tabela 8.

6 RESULTADOS

Depois do protótipo pronto, foi filmada a interação completa desde o início com o usuário. O resultado foi satisfatório pois a equipe conseguiu fazer o protótipo ser funcional. Ainda que o *design* do aplicativo era um objetivo secundário, ficou atraente conforme exposto nas figuras das telas presentes neste documento. A empresa Tropicália cedeu as características de sua marca e identidade visual para que isso fosse possível.

Os custos gerais do projeto evidenciam a viabilidade financeira do projeto em face das opções presentes deste nicho de mercado, como visto na seção de custos.

O desempenho da placa foi satisfatório em relação ao controle das travas e seu processamento. Assim como o desempenho da leitura dos sensores. A precisão da medição da balança condizia sempre com a realidade depois de calibrada. A proposta da balança é para ser de alta precisão justamente porque serão elas que determinarão a quantidade de marmitas levadas pelo usuário.

O módulo bluetooth presente na placa conseguiu de forma satisfatória se comunicar com o aplicativo de celular, possibilitando que o sistema embarcado não necessitasse da comunicação direta com a internet. Isto permitiu que o projeto ficasse mais acessível financeiramente.

No caso de segurança e resiliência do projeto para evitar erros, pode ser mencionado o efeito surpresa. O usuário cliente não sabe como são contabilizadas a marmita retiradas. Se soubesse ou descobrisse, ela poderia tentar burlar o sistema colocando um outro peso por exemplo. Apesar disso, cabe ressaltar que o erro não seria propagado para os usuários seguintes, pois sempre é feito a contabilização do que tem dentro do freezer a cada nova abertura do mesmo.

7 CONSIDERAÇÕES FINAIS

Segundo o resultado obtido neste trabalho é possível perceber a possibilidade deste projeto evoluir para um produto mínimo viável conhecido no mercado como *Minimal Viable Product(MVP)* através de um custo relativamente baixo, frente às opções de mercado. Como o alvo da implementação foi para a empresa Tropicália, os equipamentos e aparência do aplicativo ficaram personalizados para a mesma. Porém é perfeitamente possível que a implementação sofra caráter híbrido e seja vendida também para outras empresas. Estas poderiam personalizar o projeto todo para sua própria marca, desde que as diferenças entre freezers, tipos de produtos, ou abordagem não mudem de tal forma a interferir diretamente na estrutura do que foi implementado. O próximo passo seria colocar em campo para verificar a aceitação dos usuários clientes e como estes responderiam. A equipe reconhece as limitações relacionadas ao escopo do projeto apresentado neste documento, e também que os objetivos propostos foram alcançados, mostrando a viabilidade do protótipo. Porém propõe algumas entregas futuras que fariam o projeto ganhar mais robustez.

7.1 Trabalhos futuros

Por ser um protótipo, para tornar-se um produto funcional algumas melhorias seriam interessantes para o produto. Na parte do aplicativo móvel, será essencial conectar o pagamento a uma empresa para o processamento real dos pagamentos, com a cobrança do cartão de crédito. Também seria importante melhorar a experiência do usuário em alguns casos de erro que poderiam acontecer, por exemplo:

- Se o cálculo da marmita não for contabilizado corretamente, por um erro do sistema, ter alguma maneira de abrir uma 'disputa' para que o usuário pague apenas pelo que realmente levou.
- Melhorar o gerenciamento de erros caso o usuário não feche o freezer corretamente. Ter alguma forma de *timeout* ou lembrete para o usuário fechar corretamente o freezer para concluir a compra.
- Ter um plano de risco de como lidar com possíveis usuários fraudadores, quando identificados. Como existe um cartão de crédito cadastrado, poderia ser automaticamente cobrado deste cartão, ou realizar uma tentativa de contato.
- Possuir um aplicativo Android para abranger mais usuários.
- Na parte jurídica, colocar um termo de consentimento de dados móveis, já que os dados móveis do usuário cliente vão ser usados para ter acesso ao banco de dados de forma transparente.

Na parte do sistema embarcado e do freezer físico, várias melhorias podem ser consideradas. A primeira seria tornar o sistema de travas e do harware mais robusto,

utilizando uma placa de circuito impresso e posicionando o circuito em um local não acessível para um usuário comum. Também seria essencial a existência de mais balanças, para que o freezer possa ser utilizado por completo. Cada balança teria, neste caso, uma pilha de marmitas e o sistema calcularia a quantidade retirada utilizando a informação de todas essas balanças. Esta e outras implementações são necessárias para que o projeto evolua para um produto mínimo viável(MVP). O próximo passo seria colocar o MVP em teste em um condomínio, onde tivesse camêras para avaliar as respostas dos usuários.

Referências

AIRBNB. **Lottie iOS**. 2021. <https://github.com/airbnb/lottie-ios>. Citado na página 24.

ALAMOFIRE. **AlamofireImage**. 2021. <https://github.com/Alamofire/AlamofireImage>. Citado na página 24.

ATANASOV, D. **Introducing Clean Swift Architecture (VIP)**. 2017. <https://hackernoon.com/introducing-clean-swift-architecture-vip-770a639ad7bf>. Citado na página 20.

BLUETOOTH. **Bluetooth Technology**. 2021. <https://www.bluetooth.com/learn-about-bluetooth/tech-overview/>. Citado na página 18.

BRAGA, N. C. **Funcionamento do Reed Switch**. 2010. <https://www.newtoncbraga.com.br/index.php/como-funciona/2462-art373.html>. Citado 2 vezes nas páginas 16 e 17.

CAFÉ, S. M. . **Article by Star Midia Café Company**. 2018. <https://starmidiaecafe.com.br/vending-machine/>. Citado 2 vezes nas páginas 10 e 25.

CIRCUITO, C. **Destrinchando a estrutura do ESP32 e suas funções importantes**. 2018. <https://www.curtocircuito.com.br/blog/Categoria%20IoT/conhecendo-esp32>. Citado na página 23.

CLOUD Firestore. 2021. <https://firebase.google.com/docs/firestore>. Citado na página 22.

COLEMAN, C. **A Practical Guide to BLE Throughput**. 2019. <https://interrupt.memfault.com/blog/ble-throughput-primer>. Citado na página 43.

CZL635. **3133 - Micro Load Cell (0-5kg) - CZL635 Datasheet**. 2011. <https://www.robotshop.com/media/files/PDF/datasheet-3133.pdf/>. Citado na página 16.

EPTV1, G. **São Carlos tem primeira loja com venda totalmente automatizada e sem funcionários**. 2020. <https://g1.globo.com/sp/sao-carlos-regiao/noticia/2020/01/30/sao-carlos-tem-primeira-loja-com-venda-totalmente-automatizada-e-sem-funcionarios-veja-o-video.ghtml>. Citado na página 25.

ESPRESSIF. **ESP32 Series Datasheet**. 2021. <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>. Citado 3 vezes nas páginas 22, 23 e 24.

FERRASOLI, D. **Lojas de conveniência sem funcionários crescem em condomínios na pandemia**. 2020. <https://www1.folha.uol.com.br/mpme/2020/11/lojas-de-conveniencia-sem-funcionarios-crescem-em-condominios-na-pandemia.shtml>. Citado na página 27.

FIREBASE. **Firebase**. 2021. <https://github.com/firebase/firebase-ios-sdk>. Citado na página 24.

FOWLER, M. **Domain-Specific Languages**. [S.l.]: Addison-Wesley Professional, 2010. ISBN 0321712943. Citado na página 35.

GARCIA, M. R. e. S. C. R. **Estudos de caso da interação universidade-empresa no Brasil**. [S.l.]: UFMG Cedeplar, 2018. ISBN 9788560500079. Citado na página 10.

GOIÁS, E. P. **Internet das Coisas: O Que é, Como Funciona e Aplicações**. 2020. <https://ead.pucgoias.edu.br/blog/internet-das-coisas>. Citado na página 10.

GOOGLE. **Google Maps SDK**. 2021. <https://developers.google.com/maps/documentation/places/ios-sdk/overview>. Citado na página 25.

HARRINGTON, J. K. B. K. **ESP32Servo library for Arduino**. 2020. <https://www.arduino.cc/reference/en/libraries/esp32servo/>. Citado 2 vezes nas páginas 13 e 23.

HASHIBA, S. **NewWorkAlert**. 2020. <https://github.com/shiba1014/NewYorkAlert>. Citado na página 24.

HOVHANNISYAN, A. **Finite State Machine (FSM) Tutorial: Implementing an FSM in C++**. 2019. <https://www.aleksandrhovhannisyan.com/blog/finite-state-machine-fsm-tutorial-implementing-an-fsm-in-c/>. Citado na página 35.

JSON - Introduction. 2021. <https://www.json.org/>. Citado na página 21.

MAGSWITCH. **Reed sensor module MagSwitch For Arduino**. 2020. <https://www.curtocircuito.com.br/datasheet/modulo/modulo_interruptor_magnetico.pdfl>. Citado 2 vezes nas páginas 17 e 18.

MANAGER, P. **What Is the Waterfall Methodology in Project Management?** 2021. <https://www.projectmanager.com/waterfall-methodology>. Citado na página 26.

MARKET4U. **Modelo de vendas autônomos baseado na honestidade do cliente**. 2020. <https://market4u.com.br/sobre/>. Citado na página 25.

MARTIN, R. C. **Arquitetura Limpa**. [S.l.]: Addison-Wesley Professional, 2019. ISBN 9788550804606. Citado na página 20.

MECHATRONICS, H. T. **How servo Motor Works How to Control Servos using Arduino**. 2018. <https://howtomechatronics.com/how-it-works/how-servo-motors-work-how-to-control-servos-using-arduino/>. Citado na página 13.

MONGODB. **What is NoSQL?** 2021. <https://www.mongodb.com/nosql-explained>. Citado na página 22.

MYBOTIC. **Tutorials site for Arduino: Instructables**. 2020. <https://www.instructables.com/Tutorial-How-to-Calibrate-and-Interface-Load-Cell-/>. Citado na página 15.

NECULA, B. **HX711 Arduino Library**. 2021. <https://github.com/bogde/HX711>. Citado na página 23.

PESSOA, L. **Visão Técnica do Bluetooth Smart**. 2016. <https://www.embarcados.com.br/bluetooth-smart-visao-tecnica/#Generic-Attribute-Profile-GATT>. Citado na página 18.

PLATFORMIO. **Platform IO**. 2021. <https://piolabs.com/>. Citado na página 23.

STEIN, A. **How QR Codes Work and Their History**. 2020. <https://www.qr-code-generator.com/blog/how-qr-codes-work-and-their-history/>. Citado na página 21.

TOWERPRO. **Datasheet TowerPro SG-5010 Servo**. 2020. <https://https://datasheetspdf.com/pdf-file/633270/TowerPro/SG-5010/1>. Citado na página 14.

TROPICALIA. ***Startup* Tropicalia Prático-Saudável**. 2020. <https://www.linkedin.com/company/tropic%C3%A1lia-pr%C3%A1tico-saud%C3%A1vel/about/>. Citado 2 vezes nas páginas e 10.

UBER. **RxCBCentral**. 2020. <https://github.com/uber/RxCBCentral>. Citado na página 24.

USINAINFO. **Módulo Conversor Amplificador HX711 24bit 2 canais**. 2021. <https://www.usinainfo.com.br/amplificadores-de-sinal/modulo-conversor-amplificador-hx711-24bit-2-canais-2818.html>. Citado na página 15.

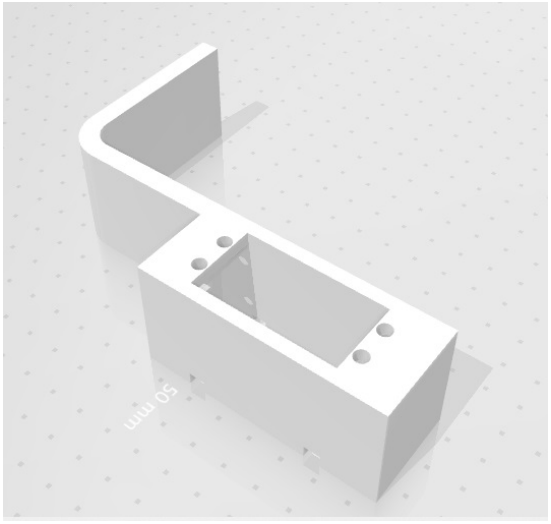
ZAHER, K.; MISHALI, S. **RxSwift**. 2021. <https://github.com/ReactiveX/RxSwift>. Citado na página 24.

ÁRTICO. **Especificações do freezer SC124**. 2021. <https://www.artico.com.br/pf/freezer-slim-sc124/>. Citado na página 12.

Apêndices

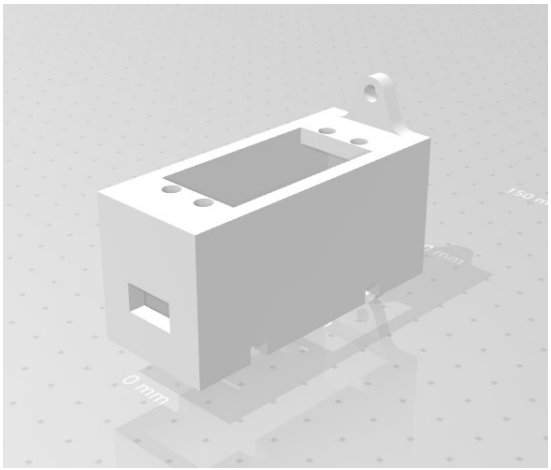
APÊNDICE A – Trabalhos auxiliares

Figura 32 – Foto do arquivo aberto do suporte da trava exterior no formato ".stl"



Fonte: Autoria própria

Figura 33 – Foto do arquivo aberto do suporte da trava interior no formato ".stl"



Fonte: Autoria própria