INSTITUTO FEDERAL DE SANTA CATARINA

LUIZA ALVES DA SILVA

Guarda-volumes inteligentes com auditoria

São José - SC

Março/2025

GUARDA-VOLUMES INTELIGENTES COM AUDITORIA

Monografia apresentada ao Curso de Engenharia de Telecomunicações do campus São José do Instituto Federal de Santa Catarina para a obtenção do diploma de Engenheira de Telecomunicações.

Orientador: Prof. Cleber Jorge Amaral, Dr.

São José - SC

Março/2025

Luiza Alves da Silva

Guarda-volumes inteligentes com auditoria

Este trabalho foi julgado adequado para obtenção do título de Engenheira de Telecomunicações, pelo Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, e aprovado na sua forma final pela comissão avaliadora abaixo indicada.

São José - SC, 21 de março de 2025:

Prof. Cleber Jorge Amaral, Dr.
Orientador
Instituto Federal de Santa Catarina

Prof. Clayrton Monteiro Henrique, Me.
Instituto Federal de Santa Catarina

Prof. Ederson Torresini, Me.
Instituto Federal de Santa Catarina

A mente que se abre a uma nova ideia
jamais voltará ao seu tamanho original.
A. Einstein

RESUMO

Este projeto desenvolve um guarda-volumes que integra um microcontrolador e um servidor para monitorar e registrar suas interações, como a identificação do usuário, compartimento ocupado e o tempo de uso. Utilizando tecnologia RFID, o sistema controla automaticamente a abertura e o fechamento das portas, melhorando a segurança e a eficiência na identificação dos usuários. O projeto engloba também uma adaptação mecânica para a abertura automatizada dos compartimentos por meio de sistemas eletrônicos. O microcontrolador integrado com uma interface de rede permite o acesso controlado ao armário, efetuando a autorização de usuários no sistema, enquanto o servidor visa simplificar a gestão com auditoria no sistema. Espera-se que o projeto facilite a operação tanto para os administradores do local quanto para os usuários dos guarda-volumes, reduzindo os inconvenientes associados à abertura e à segurança dos compartimentos.

Palavras-chave: armários inteligentes. automação. auditoria.

ABSTRACT

This project develops a locker system that integrates a microcontroller and a server to monitor and record its interactions, such as user identification, compartment occupancy, and usage time. Utilizing RFID technology, the system automatically controls the opening and closing of doors, enhancing security and efficiency in user identification. The project also includes a mechanical adaptation for the automated opening of compartments through electronic systems. The integrated microcontroller with a network interface allows controlled access to the locker, performing user authorization in the system, while the server aims to simplify management with system auditing. The project is expected to facilitate operations for both site administrators and locker users, reducing the inconveniences associated with opening and securing the compartments.

Keywords: smart lockers. automation. auditing.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AP *Acess Point*.

API Application Programming Interface.

ASK *Amplitude Shift Keying*.

BLE *Bluetooth Low Energy*.

BPSK *Binary Phase Shift Keying*.

BSS *Basic Service Set*.

CS Chip Select.

DSSS *Direct Sequence Spread Spectrum*.

EPC *Electronic Product Code*.

ESP-IDF *Espressif IoT Development Framework*.

ESP32 *Espressif Systems 32-bit Microcontroller*.

GPIO *General Purpose Input/Output*.

I2C *Inter-Integrated Circuit*.

IDE *Integrated Development Environment*.

IFSC Instituto Federal de Santa Catarina.

IHM Interface Homem-Máquina.

IoT *Internet of Things*.

LED Light Emitting Diode.

MIFARE *MIkron FARE Collection System*.

MIMO *multiple-input and multiple-output*.

MISO Master In Slave Out.

MOSI Master Out Slave In.

MQTT *Message Queuing Telemetry Transport*.

MVP Minimum Viable Product.

NFC Near Field Communication.

NRZ-L *Non-Return-to-Zero Level.*

NVS Non-Volatile Storage.

OOK *On-Off Keying.*

PAN *Personal Area Network.*

POC *Proof of Concept.*

QR Code Quick Response Code.

RF Radiofrequência.

RFID Radio Frequency Identification.

RGB Vermelho (Red), Verde (Green) e Azul (Blue).

SC-FDMA *Singe Carrier - Frequency Division Multiple Access.*

SCK Serial Clock.

SHF *Super High Frequency.*

SMS Serviço de Mensagens Curtas.

SoC *System-on-chip.*

SPI *Serial Peripheral Interface.*

SQL Structured Query Language.

TLS *Transport Layer Security.*

UART *Universal Asynchronous Receiver/Transmitter.*

UHF *Ultra High Frequency.*

UPC *Universal Product Code.*

USB *Universal Serial Bus.*

VSCode *VSCode Extension.*

WEP *Wired Equivalent Privacy.*

Wi-fi *Wireless Fidelity.*

WLAN *Wireless Local Area Network.*

WPA *Wi-Fi Protected Access.*

WPA2 *Wi-Fi Protected Access 2.*

WPA3 *Wi-Fi Protected Access 3.*

SUMÁRIO

1 INTRODUÇÃO

Em ambientes com grande fluxo de pessoas, nota-se a necessidade de guarda-volumes, seja para manter a organização do ambiente ou facilitar o dia a dia dos usuários. No entanto, esses armários podem apresentar vários problemas, como perda de chaves, cópias indevidas e falhas mecânicas que requerem manutenção frequente das fechaduras mecânicas. A automatização de guarda-volumes pode ser uma solução eficaz para simplificar o processo administrativo e reduzir significativamente os problemas mencionados, fornecendo relatórios em tempo real sobre a utilização e otimizando o uso desses espaços.

O Instituto Federal de Santa Catarina campus de São José é frequentado por aproximadamente 1,9 mil alunos no total dos diversos cursos ofertados IFSC (2022b). Um dos auxílios ofertados pelo campus é a disponibilização de 75 compartimentos de guarda-volumes para os discentes. Os interessados realizam sua inscrição através de um edital, podendo desfrutar deste benefício ao longo do semestre letivo atual. Os contemplados ficam sob posse da chave de um dos compartimentos do armário, devolvendo-a ao final do período vigente. Com o edital, pretende-se auxiliar a permanência do aluno na instituição e minimizar a mobilidade diária com os materiais (IFSC, 2022a).

Assim, observa-se que os guarda-volumes com controle de acesso através de chaves mecânicas podem ser pouco práticos quando problemas externos acontecem. Apesar de ser um serviço útil aos estudantes, o método mostra-se pouco eficiente. A chave do compartimento permanece com o mesmo usuário durante todo o semestre letivo, tendo o principal critério de escolha para a obtenção da chave a ordem de chegada no dia determinado pelo edital. Então, caso um discente não contemplado queira fazer uso esporádico, não terá essa possibilidade, mesmo que haja compartimentos em desuso. Além disso, o controle de acesso aos compartimentos possui alguns pontos críticos, que podem acabar trazendo sérios problemas. A segurança é um deles; caso ocorra a perda da chave, por exemplo, o usuário não poderá ter seus itens até que a situação seja resolvida. Pode vir a ocorrer cópias indevidas por algum usuário mal-intencionado, e este se sentir no direito de utilizar tal recurso enquanto outra pessoa está fazendo uso.

Uma alternativa moderna e eficiente é a implementação de tecnologia RFID nos guarda-volumes. A utilização de RFID permite um controle de acesso mais dinâmico, facilitando a gestão e permitindo o uso esporádico por parte de outros estudantes. Diferentemente das chaves mecânicas, a tecnologia RFID elimina problemas como perda ou cópia não autorizada de chaves, pois cada *tag* é única e pode ser facilmente desativada ou reconfigurada pelo sistema de gestão. Além disso, muitos dispositivos móveis modernos estão equipados com NFC, um subconjunto da tecnologia RFID, permitindo

que estudantes usem seus próprios *smartphones* como chaves. Esta integração aproveita a disponibilidade dos usuários com seus dispositivos móveis e simplifica a gestão dos acessos aos armários, eliminando as limitações e os riscos associados ao uso de chaves mecânicas tradicionais.

Com uma central inteligente e auditoria dos armários, a segurança dos itens armazenados nos compartimentos se torna menos vulnerável. Nos casos de arrombamento, o administrador recebe uma notificação de que há uma abertura indevida, possibilitando agir de maneira mais rápida diante desta situação. Quando um usuário esquece itens nos compartimentos, o responsável pelo sistema pode entrar em contato com o usuário informando o ocorrido. Isso também permite que a administração do ambiente tenha fácil acesso à quantidade de compartimentos disponíveis. Assim, observa-se que os guarda-volumes com controle de acesso através de chaves mecânicas podem ser pouco práticos quando problemas ocorridos por eventos externos acontecerem.

No mercado existem soluções de armários que utilizam uma central inteligente controlando os compartimentos. O *Amazon Hub Locker* é um armário eletrônico desenvolvido pela empresa *Amazon* que permite que seus clientes retirem suas encomendas através de um autoatendimento no local de instalação dos armários que acharem mais apropriado. Para o usuário pegar sua mercadoria, é preciso inserir no painel do *Locker* um código de 6 dígitos recebido por *e-mail*, então o compartimento com a respectiva mercadoria irá destravar (AMAZON, 2022). Também são encontradas outras empresas com ideias similares ampliando para diferentes aplicações, como condomínios, indústrias, shoppings, etc. (OIHANDOVER, 2022). Em suma, abertura de compartimentos através de carteira de estudante, cartão de crédito, aplicativo de celular, por exemplo, se torna uma solução muito atraente.

A proposta deste trabalho é desenvolver um MVP de guarda-volumes inteligentes com auditoria. O objetivo é integrar um dispositivo eletrônico que permita a abertura dos compartimentos de forma mais segura e eficiente, eliminando a dependência de chaves mecânicas. Com isso, pretende-se alcançar mais estudantes e otimizar o uso dos compartimentos, evitando a ociosidade e simplificando processos burocráticos, como o uso de editais.

1.1 Objetivo geral

O objetivo geral deste projeto é desenvolver um MVP de armários inteligentes com auditoria, visando automatizar e modernizar o sistema de guarda-volumes, melhorando a segurança, a eficiência e otimizando a gestão do recurso.

1.1.1 Objetivos específicos

Para alcançar o objetivo geral, definiram-se os objetivos específicos.

1. **Sistema de Abertura e Controle Eletrônico**: Desenvolver um sistema robusto para a abertura e o fechamento eletrônico dos compartimentos, visando automatizar o processo e aumentar a segurança dos armários.

2. **Autenticação de Usuários**: Implementar um mecanismo de autenticação que permita a abertura automatizada dos compartimentos apenas por usuários autorizados, utilizando tecnologias modernas como RFID para garantir a proteção contra acessos não autorizados.

3. **Sinalização Visual**: Desenvolver um sistema de sinalização simples que facilite a interação dos usuários com o sistema de armários, proporcionando feedback imediato e intuitivo durante o uso.

4. **Serviço e Aplicação de Gestão de Usuários**: Desenvolver um software para o cadastro e controle de usuários que integre funcionalidades de gerenciamento, auditoria e monitoramento, proporcionando uma administração eficiente e transparente do sistema.

5. **Integração Eletromecânica e Comunicação com Servidor**: Estabelecer uma comunicação eficaz entre a infraestrutura eletromecânica dos armários e o servidor central, assegurando que todas as operações sejam monitoradas e gerenciadas centralizadamente.

6. **Prototipagem e Validação com MVP**: Construir e testar um MVP para validar a funcionalidade e eficácia do sistema proposto, identificando oportunidades de melhoria e ajustando o projeto conforme necessário para atender às necessidades dos usuários finais.

1.2 Organização do texto

Este Capítulo 1 estabelece a base para a compreensão do projeto como um todo, abordando a introdução e os objetivos. Em seguida, o Capítulo 2 serve como a fundamentação teórica, provendo os conceitos essenciais que suportam o desenvolvimento e a compreensão da proposta deste projeto. O Capítulo 3 apresenta a proposta detalhada deste projeto. Os resultados são discutidos no Capítulo 4 e, por fim, no Capítulo 5 são apresentadas as conclusões sobre este trabalho.

2 REVISÃO BIBLIOGRÁFICA

2.1 Guarda-volumes convencionais

Em alguns locais onde há um grande fluxo de pessoas, pode ser encontrado guarda-volumes. Os guarda-volumes são móveis que possuem um conjunto de vários compartimentos, que servem para as pessoas armazenarem seus pertences durante algum tempo. Para a utilização desses compartimentos, os usuários precisam estar de acordo com as regras de negócios do ambiente onde se encontram (SCHIER, 2023).

Os guarda-volumes são fabricados em diversos materiais, como aço, madeira ou metal. Alguns estabelecimentos não possuem travas nos compartimentos, ficando a critério das pessoas que circulam naquele ambiente a confiança de não mexer em objetos pessoais de terceiros. Outros estabelecimentos possuem cadeados ou fechaduras mecânicas com chaves para trancar os compartimentos. Se o armário estiver equipado com uma chave, você abre o compartimento, armazena seus itens pessoais no interior, fecha o compartimento com segurança e mantém consigo a chave correspondente para garantir a segurança de seus pertences.

As aplicações dos guarda-volumes são amplas, visando principalmente a organização e segurança de objetos pessoais em diferentes ambientes. Nas academias, encontram-se comumente armários convencionais com cadeados ou até mesmo desprovidos de qualquer medida de segurança. Em locais como shoppings, supermercados e instituições de ensino, é comum encontrar armários convencionais que utilizam chaves, geralmente sob a supervisão de um funcionário responsável pela gestão e segurança dos itens armazenados. No entanto, esses sistemas podem apresentar alguns inconvenientes, como a necessidade de manutenção, o risco de perda de cadeados ou chaves, a possibilidade de cópias indevidas e outras circunstâncias adversas.

2.2 Armários eletrônicos

O interesse pelo uso de tecnologias novas para facilitar o dia a dia das pessoas está em constante crescimento. Os produtos a seguir oferecem soluções de armários eletrônicos para diferentes segmentos do mercado. Os armários eletrônicos podem ser personalizados de acordo com a solicitação do cliente, com compartimentos de tamanhos variados e um sistema modular, o que possibilita sua expansão. As soluções permitem integração com o sistema do cliente, possuem também a possibilidade de abertura de compartimentos através de aplicativo e outras tecnologias de acesso.

A Tabela 1 analisa três empresas de armários eletrônicos sob cinco critérios: empresa, solução, método de abertura, gestão online e IHM. Esse levantamento oferece uma noção do mercado para armazenamento seguro e eficiente. As empresas analisadas são: Inteligentes (2023), Oppitz (2023) e Duratta (2023).

Em relação às soluções disponíveis, os guarda-volumes são comuns a todas as empresas citadas. No entanto, a empresa Armários Inteligentes oferece também outras duas soluções distintas: Armário autônomo e Armário conectado. Os métodos de abertura dos compartimentos variam e incluem senhas, biometria, códigos de barras, RFID, QR Code, além de acessos via SMS, *e-mail* e aplicativos de celular. Todas as empresas proporcionam a opção de biometria em alguns de seus produtos. Entre as empresas mencionadas, a Opptiz não oferece RFID e código de barras, mas é a única que disponibiliza o acesso via SMS/*e-mail*. Por outro lado, a Duratta não inclui QR Code, senha, nem aplicativo de celular em suas soluções.

Quanto à gestão online, a empresa Armários Inteligentes oferece essa funcionalidade de forma opcional, enquanto a Opptiz possui gestão online em todos os seus modelos. A Duratta, por sua vez, não possui essa funcionalidade. Em relação às interfaces homem-máquina, observa-se que apenas a Duratta opta por uma tela sensível ao toque, em contraste com as outras empresas que utilizam telas LCD ou outras interfaces mais simples. Este aspecto indica um esforço das empresas para facilitar a interação do usuário e aprimorar a experiência de uso.

Tabela 1 – Comparação de empresas de armários eletrônicos

Empresa	**Solução**	**Método de abertura**	**Gestão online**	**Interface homem-máquina**
Armários inteligentes	Guarda-volumes	Senha	Opcional	Sem informações
		Código de barras		
		QR Code		
		RFID		
		NFC		
	Armário autônomo	Senha numérica + biometria	Não	LCD e teclado numérico
		Senha numérica + RFID		
		Senha numérica + código de barras		
	Armário conectado	Comando recebido via aplicativo	Sim	Aplicativo de celular
Opptiz	Guarda-volumes	Senha numérica	Sim	LCD
		Biometria		
		QR Code		
		SMS ou e-mail		
		Aplicativo de celular		
Duratta	Guarda-volumes	Código de barras	Não	Tela sensível ao toque
		RFID		
		Biometria		

Fonte: Autoria própria a partir dos dados de Inteligentes (2023), Oppitz (2023) e Duratta (2023)

Das empresas citadas, o grande diferencial é a gestão online que torna o produto mais caro e a interface homem-máquina, tornando o produto mais custoso, sujeito a ma-

nutenções mais frequentes. Como por exemplo, o uso de tela sensível ao toque está sujeito a danos, muitas vezes fora do controle administrativo, o que pode comprometer a durabilidade do equipamento. O mesmo pode ser observado com teclados numéricos sensíveis ao toque, amplamente adotados por muitas empresas citadas anteriormente. Problemas recorrentes com a manta de contato desses teclados podem levar a uma manutenção mais frequente, impactando a eficiência e a confiabilidade desses dispositivos.

Uma análise breve de mercado revela que certos dispositivos e materiais adotados por algumas dessas empresas apresentadas possuem um custo elevado, resultando em um preço final do produto que ultrapassa significativamente os preços típicos do segmento. Os leitores biométricos e armários fabricados com aço contribuem para o aumento dos custos; ambos não são materiais e dispositivos de baixo custo. Da mesma forma, a inclusão de câmeras de monitoramento não apenas encarece o produto, mas também implica em uma necessidade frequente de manutenção em caso de falhas. Outra desvantagem é a abertura dos compartimentos por aplicativo. Dessa forma, torna-se obrigatório o uso de um celular com acesso a *internet* na utilização do benefício, limitando o uso de usuários. Assim como a necessidade de manutenção e custos de implementação frequentes, já que atualizações ocorrem para garantir a eficiência do sistema.

A título de exemplo, para a contratação de seus serviços, a empresa Armários Inteligentes conta com uma mensalidade. Nela está inclusa a manutenção e atualização do sistema. O valor mais barato de assinatura é o armário autônomo, que custa uma mensalidade em torno de 8 a 10 mil reais por armário. Apesar de ser o produto mais em conta da empresa, dentro desse nicho é um produto de alto valor no mercado. Cada armário possui em torno de 25 compartimentos, ou seja, uma única central controla apenas um armário. Caso o cliente queira, por exemplo, 50 compartimentos, ele precisa solicitar duas centrais, que irão se integrar e controlar todos os compartimentos.

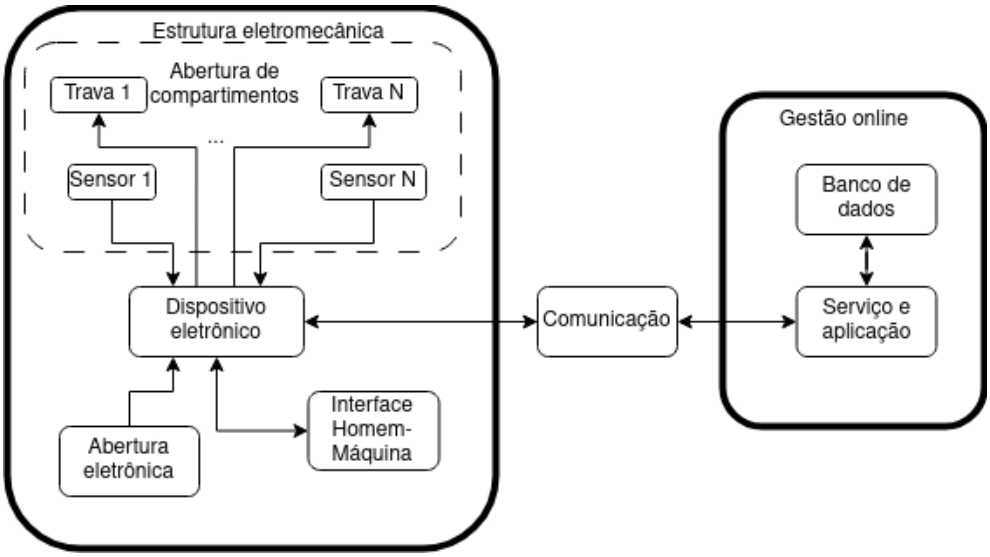
2.3 Componentes para a construção de armários eletrônicos

A transição de guarda-volumes convencionais que utilizam fechadura mecânica para uma versão eletrônica não é simplesmente a remoção de chaves físicas, há uma complexidade maior nessa integração. Muitos componentes conectados entre si são necessários para que o controle desses armários ocorra de maneira eficiente. Quando o usuário se autentica e solicita a abertura de um compartimento, ao remover a fechadura mecânica e substituí-la por uma trava automática, deve-se fazer uma verificação para detectar se a porta foi aberta e fechada corretamente. Assim, o controle eletrônico avisará o usuário através de uma interface o status do seu atual compartimento. Para permitir somente usuários autorizados a acessar os gabinetes, o sistema precisa ter conectividade com a *internet* em algum momento, para levar as informações para o servidor através de algum

protocolo de comunicação, viabilizando-se, assim, uma gestão online.

Na Figura 1, é apresentado um diagrama destacando os principais componentes necessários para a construção de armários eletrônicos. O componente principal é o dispositivo eletrônico, que integra todos os outros componentes, permitindo assim o funcionamento correto do sistema. Para permitir a abertura dos compartimentos, é necessário integrar os componentes de trava com os sensores de detecção de estado do compartimento (aberto ou fechado). A abertura ocorre quando o dispositivo eletrônico recebe os dados relativos do componente de abertura eletrônica, solicitando a abertura de um compartimento, desencadeando a operação das travas. A partir disso, a trava irá atuar de acordo com os dados recebidos do sensor. A comunicação com o sistema de gestão online para relatar eventos e incorporar novos usuários ao sistema também acontece através do dispositivo eletrônico. Além disso, ele envia informações para a interface homem-máquina, aprimorando a interação do usuário com o sistema.

Figura 1 – Componentes principais para construção de armários eletrônicos



Fonte: Autoria própria

2.3.1 Estrutura eletromecânica

Sistema eletromecânico consiste na integração de componentes elétricos e mecânicos para criar sistemas com funções específicas. A integração desses componentes tem papel fundamental na operação e no controle do armário. Os principais componentes da parte eletromecânica incluem as travas, os sensores e o dispositivo eletrônico central. Esses elementos trabalham em conjunto para garantir o funcionamento adequado do sistema.

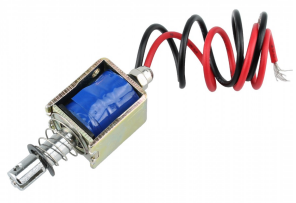
2.3.1.1 Fechadura eletrônica - Trava elétrica solenoide

A necessidade de manter um compartimento fechado é para evitar que pessoas não autorizadas tenham acesso a tais locais guardados. Dependendo do ambiente em que se encontra esse compartimento, seja ele uma porta, um armário ou um objeto, nota-se a necessidade de aumentar sua segurança, onde apenas uma fechadura sem nenhuma trava não é suficiente. O modelo mais comum são as fechaduras mecânicas, compostas por cilindro interno com uma chave no engate, que, ao girar, permite o destravamento da fechadura.

Os modelos de fechaduras eletrônicas têm-se tornado cada vez mais populares com a crescente evolução da tecnologia, muito comum em locais que necessitam de um controle maior de pessoas que os acessam. As fechaduras eletrônicas são controladas por uma tensão de alimentação, onde, ao receber um pulso elétrico, a trava realiza uma ação de abertura ou de fechamento. Além disso, os sistemas de fechaduras eletrônicas operam baseados em diferentes princípios de funcionamento, que definem como a trava é acionada. Esses sistemas podem ser acionados por diversos métodos, como inserção de senha, leitura de identificação biométrica, uso de cartões magnéticos ou ativação por solenoides. Cada método de acionamento oferece vantagens específicas em termos de segurança e controle de acesso, permitindo uma adaptação às necessidades particulares de cada aplicação.

Na Figura 2 pode ser visualizada a parte interna de uma trava elétrica solenoide. Para o seu funcionamento correto, na sua composição possui uma bobina de fios com n espiras, onde, ao ser aplicada uma corrente elétrica no fio condutor, cria-se um campo magnético. Essa força aplicada, então, recolherá o pino da fechadura através de uma mola, fazendo com que seja possível a abertura da trava. Com isso, pode-se perceber que, quando não há fonte de energia, o local guardado pela fechadura permanecerá fechado até que tudo se normalize (CARDOSO, 2010).

Figura 2 – Trava elétrica solenoide internamente



Fonte: Usinainfo (2023)

2.3.1.2 Sensor de detecção - *Reed Switch*

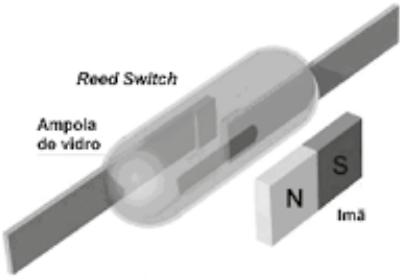
Ao utilizar uma trava eletrônica em cada compartimento, exige-se um monitoramento da porta para que, ao travá-la, tenha-se a certeza de que a porta esteja fechada corretamente. Dessa forma, nota-se a necessidade de utilização de sensores para realizar

essa detecção e garantir o funcionamento correto em todos os casos de uso. Dentre todos os sensores existentes que realizam essa tarefa, os sensores de proximidade magnéticos são comumente utilizados devido ao seu baixo custo e alta durabilidade.

O princípio de funcionamento dos *Reed Switch* é baseado em converter um campo magnético em sinais digitais ou analógicos, enviando esse sinal ao sistema de controle. O componente *Reed Switch*, que pode ser visto na Figura 3, possui uma ampola de vidro e internamente possui duas lâminas, que em seu estado natural não se encostam. Quando estão sob a presença de um campo magnético, o ímã, as lâminas fecham contato, permitindo a passagem de corrente elétrica entre seus terminais (THOMAZINI; ALBUQUERQUE, 2020).

As hastes do sensor de *reed switch* são normalmente feitas de material ferromagnético. Quando um campo magnético externo é aplicado ao sensor, as hastes são atraídas uma em direção à outra, fechando o circuito interno. Esse movimento magnético é o que faz as hastes se moverem. Quando o campo magnético é removido, as hastes retornam à sua posição original devido à força de mola dentro do sensor. Assim, o *reed switch* funciona como um interruptor controlado magneticamente, abrindo ou fechando o circuito conforme a presença ou ausência de um campo magnético externo.

Figura 3 – Sensor de Proximidade Magnético



Fonte: Thomazini e Albuquerque (2020)

Segundo Moraes e Castrucci (2006), esses sensores de proximidade possuem vantagens e desvantagens relevantes que devem ser levadas em conta dependendo de sua aplicação. Esses sensores são particularmente valorizados pela sua resposta rápida, baixo custo, vida útil prolongada e facilidade de instalação. No entanto, eles também apresentam limitações, como a sensibilidade a interferências eletromagnéticas, que podem levar a falsos positivos, e a falta de seletividade na detecção de diferentes fontes magnéticas, o que pode ser um desafio em aplicações que exigem maior precisão. Uma visão detalhada dessas características pode ser vista na Tabela 2, que compara as vantagens e desvantagens dos *reed switches*. Por isso, esses sensores de proximidade magnéticos são muito utilizados para detecção de abertura de portas e janelas.

Tabela 2 – Comparação das Vantagens e Desvantagens do Sensor Reed Switch

Vantagens	**Desvantagens**
Baixo custo de aquisição.	Sensível a interferências eletromagnéticas, podendo resultar em falsos positivos.
Resposta rápida na detecção.	Reage a qualquer campo magnético sem distinguir a fonte.
Vida longa do componente.	
Fácil instalação.	

2.3.2 Controlador de acesso - ESP32

Microcontroladores são compostos por unidades de processamento, memórias e periféricos de entrada e saída em um único *chip*. Normalmente, utiliza-se a linguagem de programação C para o desenvolvimento de sistemas, onde o compilador C abstrai os processadores, enquanto os periféricos de entrada e saída são configurados pelo programador, que variam acordo com a arquitetura de cada *chip*. Por isso, é importante analisar o funcionamento de cada chip, normalmente encontrado no site do fabricante (ALMEIDA, 2016).

Segundo a empresa Espressif (2023), os microcontroladores da *Serie* ESP32 foram projetados para se obter robustez, versatilidade e confiabilidade em diversas aplicações *IoT* distintas como, por exemplo, *smart home*, agricultura inteligente, assistência médica robótica, etc. O *SoC* da família ESP32 é a base do *kit* de desenvolvimento *ESP32-DevKitC*, que pode ser visto na Figura 4. Destacam-se alguns componentes principais nos itens a seguir:

- Processador *single/dual core*, cada um com velocidades de *clock* de até 240 MHz;
- Módulo RF receptor e transmissor de 2.4GHz;
- Gerador de *clock*;
- Conectividade *Wi-fi* modo Estação, modo *AP* e Estação/*AP*;
- Conectividade *Bluetooth* Clássico e *BLE*;
- *GPIO* para conexão com periféricos externos;
- Conversores analógico-digital de 12 bits e digital-analógico de 8 *bits*;
- Interfaces de comunicação como *SPI*, *I2C* e *UART*.

Figura 4 – Kit de desenvolvimento ESP32-DevKitC

Fonte: Autoria própria

2.3.2.1 Ambiente de desenvolvimento - ESP-IDF

O ambiente de desenvolvimento utilizado para o hardware desta aplicação será baseado no chip da Espressif, o ESP32. Para o desenvolvimento, será empregada a *IDE VSCode*, disponível gratuitamente no *VSCode Marketplace*. Esta IDE será complementada pela extensão *ESP-IDF*, que facilita a compilação e gravação do *firmware* nos *SoCs* da Espressif. É recomendável utilizar um kit de desenvolvimento específico da Espressif, como o mostrado na Figura 4, para garantir compatibilidade e eficiência durante as fases de teste e desenvolvimento. Após a instalação da extensão *ESP-IDF* no *VSCode*, a placa do kit *ESP-IDF* deve ser conectada ao computador usando um cabo *USB* - USB A / micro USB B. Este *setup* é compatível com os sistemas operacionais *Windows*, *Linux* e *MacOS*, proporcionando versatilidade e acessibilidade aos desenvolvedores.

2.3.3 Sistema de autenticação - RFID

Segundo Júnior e Farinelli (2018), sistemas de RFID são uma tecnologia para comunicações sem fio, que permite a conectividade entre periféricos. Algumas aplicações como, por exemplo, rastreamento de objetos, identificação, controle de acesso, facilitam o dia a dia de pessoas e empresas a níveis organizacionais. Além disso, o RFID possui uma implementação simples e de baixo custo.

2.3.3.1 Componentes do Sistema RFID

O sistema RFID consiste em três componentes principais: o leitor, a etiqueta e o software de gerenciamento. O leitor é um dispositivo que emite sinais para ler as informações da etiqueta, enquanto a etiqueta representa o objeto a ser identificado, autorizando sua requisição ao sistema. O software de gerenciamento processa e interpreta os dados coletados pelo leitor. O leitor emite um sinal RF através da sua antena; a energia transmitida é captada pela etiqueta. Dessa forma, o circuito interno da etiqueta recebe esse

sinal, essa energia é convertida para alimentá-lo e transmite de volta os dados solicitados pelo leitor. A alimentação das etiquetas pode ocorrer de duas maneiras: etiquetas ativas possuem fonte de energia própria, e para etiquetas passivas a energia é captada do sinal transmitido pelo leitor (JÚNIOR; FARINELLI, 2018).

2.3.3.2 Etiquetas RFID Passivas

Em etiquetas passivas é criada uma PAN, que é uma rede privada entre uma etiqueta e o leitor, normalmente tem uma distância de aproximadamente 10 a 30 centímetros, devido ao fato de que a alimentação da etiqueta passiva é recebida pelo leitor. As etiquetas com alcance de 10 centímetros para a faixa de frequência de 125 a 134 kHz são de simples implementação e baixo custo, tornando possível o envio de dados a uma baixa taxa de transmissão e o contato com água não ter influência no seu funcionamento. Para as etiquetas com alcance de 30 centímetros para a faixa de frequência de 13.56 MHz, as aplicações mais comuns são para cartões de acesso, nessa frequência de operação, a água influencia no seu funcionamento. Por sua vez, nas etiquetas ativas é criada uma rede pessoal PAN entre o leitor e várias etiquetas, formando uma topologia estrela, sua maior vantagem é o maior alcance de leitura de aproximadamente 30 a 100 metros, dependendo também da frequência de operação variando entre 433 MHz e 856 a 960 MHz. Além de transferir grande quantidade de dados de maneira rápida, por operar na região espectral de frequência UHF (CALPOLY, 2016). A Tabela 3 mostra as especificações de algumas etiquetas comerciais.

Tabela 3 – Comparativo entre diversas tecnologias e padrões RFID

Tipo de etiqueta	Faixa de frequência	Alcance
Passiva	125 a 134 kHz	10 cm
Passiva	13.56 MHz	30 cm
Ativa	433 MHz e 856 a 960 MHz	30 a 100 m

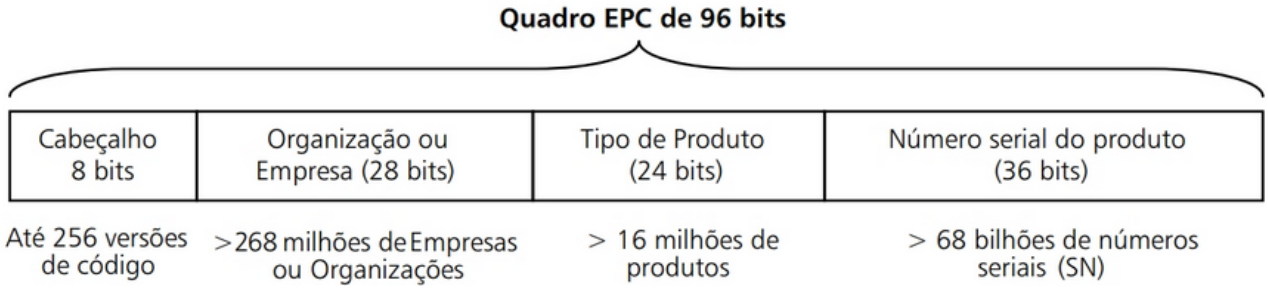
Fonte: Autoria própria a partir dos dados de CALPOLY (2016)

As etiquetas possuem uma memória interna padronizada pelo EPC. Essa memória pode permitir tanto a leitura quanto a escrita de dados, dependendo de suas características. O EPC, sucessor do identificador único UPC que é proveniente do código de barras, codifica essas informações através de um identificador único. Sua função é identificar exclusivamente um objeto por conter informações suficientes para que o objeto seja direcionado corretamente CALPOLY (2016). A Figura 5 a seguir pode ser vista a estrutura de um protocolo EPC Gen2 Class 1, que possui 96 bits e é dividido em cinco campos.

2.3.3.3 MIFARE

A tecnologia MIFARE, desenvolvida pela *NXP Semiconductors*, é amplamente utilizada em soluções de comunicação por radiofrequência para controle de acesso, transporte

Figura 5 – Estrutura de um quadro EPC



Fonte: Rochol (2018)

público e sistemas de pagamento sem contato. O funcionamento da tecnologia MIFARE é baseado na comunicação via campo eletromagnético, operando na frequência de 13,56 MHz. (NXP Semiconductors, 2011)

Os cartões e *tags* MIFARE possuem setores de 16 bytes cada, que podem armazenar dados do usuário ou informações de acesso. A memória total depende da versão do cartão, com modelos que variam entre 1 KB e 4 KB. Cada setor é protegido por chaves de autenticação (A e B) que controlam o acesso à leitura e escrita.

O leitor *MFRC522* é compatível com a tecnologia MIFARE. Ele utiliza a modulação ASK para comunicação entre o leitor (PCD) e o cartão (PICC), com codificação *Modified Miller*. Para a comunicação do cartão para o leitor, a modulação é OOK com codificação Manchester a 106 kbit/s. Já para taxas de 212 kbit/s a 848 kbit/s, utiliza-se a modulação BPSK e codificação NRZ-L. (NXP Semiconductors, 2016)

2.3.4 Interface de usuário

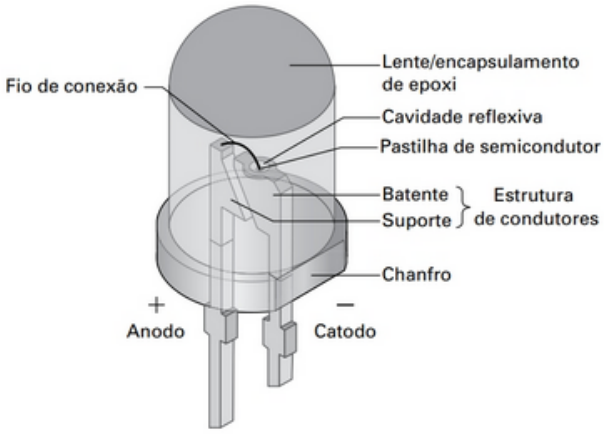
A IHM desempenha um papel crucial na interação entre os usuários e o sistema, tornando-se um dos componentes principais do sistema. A IHM deve ser intuitiva, agradável, minimizando erros durante o uso. Uma interface bem projetada contribui significativamente para a melhoria da experiência do usuário.

Sob o ponto de vista econômico, a interface gráfica nem sempre é a escolha mais apropriada. A interação com o usuário pode se manifestar por meio de diversos elementos visuais e sonoros intuitivos, direcionados a evitar equívocos. A inclusão de um *buzzer* para alertar sobre a abertura ou travamento de portas apresenta-se como uma opção interessante. Da mesma forma, LEDs com diferentes cores podem ser empregados para indicar o *status* dos armários, tais como a disponibilidade de compartimentos, o estado das portas, a ocupação total dos compartimentos, entre outras informações relevantes. Essas abordagens visam não apenas a eficácia operacional, mas também aprimoram a comunicação efetiva entre o usuário e o sistema.

2.3.4.1 LED

Os LEDs são dispositivos semicondutores que emitem luz quando estão diretamente polarizados. Possuem dois terminais, positivo e negativo, também conhecidos, respectivamente, como ânodo e cátodo. Normalmente, o componente possui uma lente de plástico colorido; entre as cores mais comuns encontradas nesses componentes estão o vermelho, amarelo e verde. Além disso, a intensidade da luz emitida varia proporcionalmente à corrente que percorre o circuito, proporcionando controle preciso sobre a luminosidade do LED (ALCIATORE; HISTAND, 2014).

Figura 6 – Partes constituintes de um LED



Fonte: MALVINO e BATES (2016)

Dentre as cores emitidas por esse tipo de LED, as mais usuais para serem utilizadas em uma sinalização visual são as cores vermelho e verde. No senso comum, essas cores têm um impacto positivo para o verde e negativo para o vermelho. Visualmente, o usuário poderá saber se ocorreu tudo certo na abertura/fechamento de um compartimento através dos LEDs.

Por exemplo, ao tentar abrir um compartimento, um LED verde pode acender para indicar que a abertura foi realizada com sucesso. Inversamente, se houver algum problema no processo, como uma falha de autenticação, um LED vermelho seria acionado para alertar o usuário de que a operação não foi concluída corretamente. Essa sinalização direta e intuitiva ajuda os usuários a compreenderem rapidamente o status do sistema sem a necessidade de interações adicionais ou verificações complexas.

2.3.5 Sistemas de comunicação

2.3.5.1 *Comunicação sem fio - Wi-Fi*

O *Wi-fi*, é uma tecnologia de comunicação sem fio que segue as especificações de normas determinadas pelo padrão IEEE 802.11. Os componentes básicos para a criação de uma rede *wireless* são conhecidos como *BSS* formados por várias estações (STAs) e

podem ter ou não um ponto de acesso. No caso de uma rede com um AP, é conhecida como rede de infraestrutura; nesse caso, as estações se conectam no ponto de acesso, se autenticam e trocam informações criptografadas (FOROUZAN, 2010).

As faixas de frequência principais do *Wi-fi* são de 2,4 GHz e 5 GHz. A faixa de frequência utilizada pela *WLAN* de 2,400 GHz até 2,4835 GHz, está no canal de radiofrequências na faixa *UHF*, possuindo uma largura de 83,5 MHz. O padrão utilizado nessa faixa é IEEE 802.11b eg e utiliza a modulação DSSS. Já a faixa de frequência de 5,725 GHz até 5,850 GHz, está no canal de radiofrequências na faixa SHF, possuindo uma largura de 125 MHz. O padrão utilizado nessa faixa é IEEE 802.11a/ac e utiliza as modulações SC-FDMA e MIMO 8 (ROCHOL, 2018).

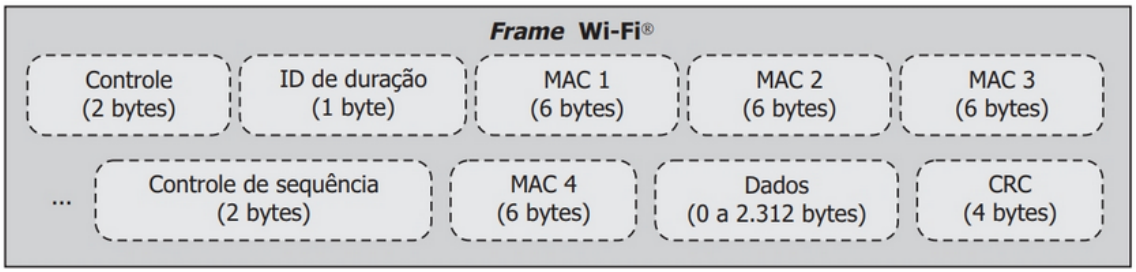
Segundo Júnior e Farinelli (2018), a arquitetura básica do Wi-fi possui três camadas:

- **LLC (*Logical link control*)**: Responsável por carregar um protocolo de alto nível (TCP, UDP e IP).
- **MAC (*Media Acess Control*)**: Responsável por montar o quadro de transmissão.
- **Física**: Responsável por controlar o envio das informações.

O pacote(*frame*) de transmissão Wi-Fi pode ser visto na Figura 7, possui nove campos, descritos a seguir. É importante notar que a estrutura desse quadro pode apresentar variações quando mais de uma estação (STA) está envolvida na transmissão.

- **Controle**: contém a versão do protocolo, total de 2 *bytes*. O segundo campo é de ID *Identity* de duração, que contém um valor configurado automaticamente pelo *hardware*, total de 1 *byte*.
- **ID de duração**: contém um valor configurado automaticamente pelo *hardware*, total de 1 *byte*.
- **MAC 1 a 4**: contém respectivamente endereço do destinatário, de origem, do transmissor, do receptor e alcance máximo, total de 6 *bytes*.
- **Controle de sequencia**: contém o número do pacote caso tenha sido dividido em vários pacotes, para saber a ordem dos pacotes, total de 2 *bytes*.
- **Dados**: contém a informação a ser transmitida, total de 0 a 2.132 *bytes*.
- **CRC(*Cyclic Redundancy Check*)**: contém um calculo para detecção de erros do pacote, para verificar sua integridade, total de 4 *bytes*.

Figura 7 – Quadro de transmissão Wi-Fi



Fonte: Júnior e Farinelli (2018)

O pacote de transmissão pode ser enviado com criptografias diferentes, depende da configuração da rede *Wi-fi* adotada pelo usuário. Ao utilizar a segurança WPA, é um protocolo que possui uma chave de acesso de 256 *bits*. O WPA2 é uma evolução do WPA e é considerado o mais seguro entre os mencionados. Mais recentemente, o WPA3 foi introduzido como o padrão de segurança mais avançado e seguro, oferecendo melhor proteção contra ataques externos, como tentativas de decifração por força bruta e ataques de canal lateral. WPA3 utiliza um novo método de handshake chamado Simultaneous Authentication of Equals (SAE), que proporciona uma segurança mais forte e é projetado para prevenir ataques comuns que exploravam falhas nos protocolos anteriores. Vale mencionar que o WEP e WPA já foram quebrados por força bruta (JÚNIOR; FARINELLI, 2018).

2.3.5.2 *Protocolo de comunicação - MQTT*

Protocolo de comunicação é um conjunto de regras e formatos a fim de padronizar a comunicação entre dois dispositivos, mesmo que tenham linguagens diferentes. É importante especificar o formato dos dados nas mensagens e a sequência das mensagens trocadas (COULORIS et al., 2013).

Segundo IBM (2023), os *message brokers* são protocolos de comunicação orientados a mensagens. São responsáveis por garantir que a mensagem chegue corretamente ao seu destino sem se importar onde estão e quem são esses destinatários. Eles utilizam fila de mensagens para garantia de entrega e confiabilidade, por isso a mensagem é retirada da fila apenas após sua confirmação. Como mencionado acima, eles não se importam se o destinatário está disponível; isso ocorre por utilizarem o sistema de mensagens assíncronas. Dessa forma, garantem a eficiência mesmo em ambientes hostis. Por ambientes hostis, entende-se situações de rede onde há problemas comuns como latência elevada, taxas de erro altas, intermitência na disponibilidade dos serviços, que podem prejudicar a comunicação regular. É possível encontrar dois modelos de *message broker*:

- **Sistema de mensagens ponto a ponto**: Cada mensagem na fila é enviada uma por vez apenas para um destinatário. Transações financeiras, por exemplo, é um

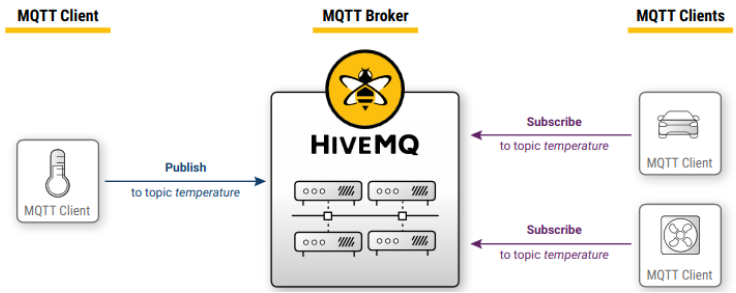
dos cenários de uso, onde precisa-se garantir que a transação ocorreu corretamente e apenas entregue ao destino certo.

- **Sistemas de mensagens de publicação/assinatura**: um produtor publica em um tópico e os consumidores inscritos naquele tópico receberão a mensagem. Uma plataforma de *streaming*, por exemplo, onde um criador de conteúdos publica seus vídeos e apenas inscritos receberão a notificação.

O MQTT é um protocolo de mensagens do modelo *publish/subscribe* orientado a eventos voltado para aplicações IoT por possuir uma implementação simples e de baixo consumo, adequado para troca de mensagens entre dispositivos com a mínima largura de banda. O mecanismo de autenticação deste protocolo é através de usuário/senha, por isso faz-se necessário o uso de TLS (criptografia da conexão TCP/IP) (MORAES; HAYASHI, 2021).

Como pode ser visto na Figura 8, na arquitetura *publish/subscribe* os clientes não trocam mensagens diretamente entre si, como no Sistema de mensagens ponto a ponto. Através do *Broker* MQTT, clientes publicam em determinados tópicos e outros clientes se inscrevem nesses tópicos para serem notificados quando ocorrer um evento. No caso da Figura 8, um cliente responsável pela temperatura publica no tópico temperatura e outros dois clientes se inscrevem nesse tópico para receber mensagens quando ocorrer um evento. Os clientes não precisam se conhecer, tudo ocorre através do MQTT *broker*.

Figura 8 – Arquitetura *Publish/Subscribe*



Fonte: HiveMQ (2020)

2.3.6 Plataforma de Gestão online

A administração e controle eficazes de um serviço disponibilizado para os usuários são de extrema importância para assegurar o seu uso otimizado. Nesse contexto, a gestão online se torna uma ferramenta essencial, possibilitando o acompanhamento em tempo real de compartimentos livres, a análise de uso e a ciência pela parte administrativa em casos de esquecimento de objetos, podendo assim, avisar o usuário. A capacidade de gerenciamento online não apenas aprimora a eficiência operacional, mas também oferece

uma abordagem proativa para atender às necessidades dinâmicas dos usuários, resultando em uma experiência mais ágil, transparente e adaptável.

2.3.6.1 Banco de dados - *MySQL*

Banco de dados é uma coleção de dados que são gerenciados por um Sistema de Gerenciamento de Banco de Dados (SGBDs). Os SGBDs armazenam e recuperam esses dados que são valiosos, grandes volumes de dados e normalmente acessados por várias aplicações ao mesmo tempo. Dentre os diferentes modelos de dados existentes, neste projeto será utilizado um tipo de banco de dados SQL, que é baseado no modelo de dados relacional. Banco de dados relacional possui sua estrutura em forma de coleção de tabelas. Cada tabela do banco de dados possui um nome único e cabeçalhos que variam de acordo com a modelagem. Esses sistemas são muito utilizados em aplicações web. O banco de dados utilizado neste projeto é o *MySQL*, que armazena dados em formato de tabelas relacionais (SILBERSCHATZ, 2020).

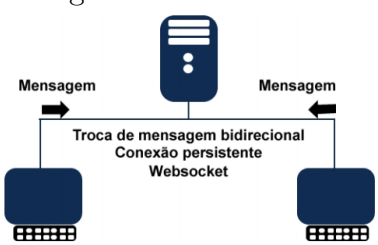
Para utilizar o banco de dados *MySQL*, é necessário instalá-lo e configurá-lo corretamente no sistema. Na API do *MySQL*, inicialmente é preciso criar tabelas para armazenar dados estruturados. Cada tabela é composta por colunas e linhas, onde as colunas definem os campos e as linhas representam as tuplas armazenadas. Também é possível realizar consultas a esses dados armazenados utilizando funções específicas para inserir, consultar, alterar e excluir (SILBERSCHATZ, 2020).

2.3.6.2 Servidor

Ao desenvolver um armário inteligente, é necessário fazer uma *interface* administrativa para gerenciar os usuários e monitorar os compartimentos. Por isso, é importante desenvolver um servidor na nuvem para que, remotamente, o administrador dos armários possa fazer esse gerenciamento e monitoramento do sistema geral.

Segundo Ferreira (2021), os protocolos *Websockets* são muito utilizados em servidores *web* para transferir dados entre cliente e servidor. Como mostra a Figura 9, o *WebSocket* realiza uma comunicação bidirecional em tempo real. Neste projeto, o *Websocket* tem o objetivo de encapsular o protocolo MQTT.

Figura 9 – Websocket



Fonte: Ferreira (2021)

2.3.6.3 API REST

Uma API é um protocolo que facilita a troca de dados, recursos e funcionalidades entre aplicativos de software, oferecendo uma interface amigável para desenvolvedores utilizarem o HTTP. APIs são *stateless*, ou seja, não mantêm informações de estado entre solicitações, garantindo que os dados do cliente não sejam salvos de uma interação para outra, tornando o sistema mais escalável e flexível. As APIs REST utilizam solicitações como GET, POST, PUT e DELETE para interagir com recursos (GOODWIN, 2024).

- **GET**: Utilizado para recuperar informações de um recurso existente.
- **POST**: Usado para criar novos recursos.
- **PUT**: Utilizado para atualizar um recurso existente.
- **DELETE**: Remove um recurso especificado.

3 DESENVOLVIMENTO

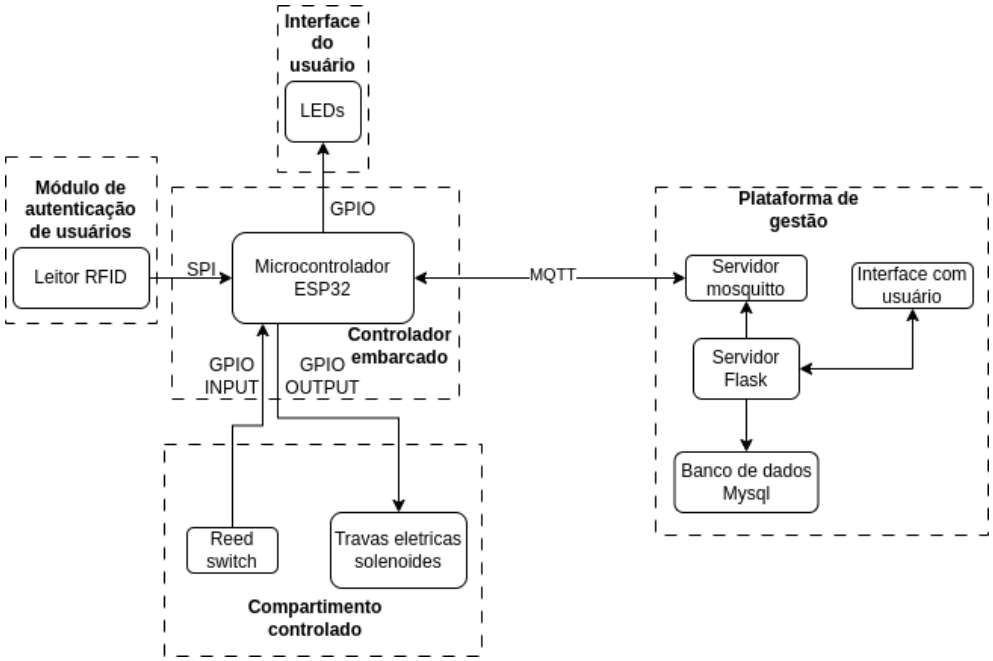
A proposta deste projeto consiste na construção de um MVP de guarda-volumes inteligentes com auditoria, destinado a demonstrar a viabilidade e a eficácia de um sistema controlado com um desenvolvimento minimalista. Esse projeto teve início no "Desafio IFSC de Ideias Inovadoras 2021"(documentado em comunicações internas), onde uma POC foi realizada utilizando, para a estrutura eletromecânica, um sensor de luminosidade e uma trava elétrica solenoide. A autenticação de usuários foi simulada por meio de dois botões, representando duas autenticações distintas. A POC permitiu validar conceitos iniciais e serviu como base para o desenvolvimento do MVP atual. Na Figura 10, apresenta-se a topologia do sistema proposto, detalhando a comunicação entre os componentes e a arquitetura geral da solução.

O projeto utiliza um microcontrolador ESP32, da Espressif, como controlador embarcado, responsável por receber informações do módulo de autenticação de usuários, que é um leitor RFID com tecnologia MIFARE, permitindo o acesso aos compartimentos. No MVP, os LEDs funcionam como a interface do usuário, e o controlador embarcado gerencia essa interface, controlando o estado dos LEDs. A comunicação entre o controlador embarcado e o servidor é realizada através do protocolo MQTT, permitindo o acesso aos compartimentos, o rastreamento do uso e garantindo a auditoria das operações realizadas.

Cada compartimento será controlado por uma estrutura eletromecânica composta por uma trava elétrica solenoide e sensor de detecção, *Reed Switch*, permitindo a abertura e fechamento automáticos. Para maior conforto e segurança, o sistema utiliza uma chave de autenticação personalizada para cada usuário, substituindo as chaves físicas tradicionais de armários convencionais. Como chave de acesso, podem ser utilizados cartões de transporte, cartões de crédito, celulares com tecnologia NFC ou qualquer objeto com tecnologia RFID embutida, operando na frequência de 13.56 MHz.

Todas as operações realizadas nos armários serão monitoradas pelos administradores do sistema, e os usuários poderão visualizar seu próprio histórico de uso. A plataforma de gestão será desenvolvida utilizando o *framework Flask*, e incluirá um servidor MQTT, um banco de dados *MySQL* e uma API REST. O servidor MQTT trocará informações com o controlador embarcado, enquanto as informações auditadas serão armazenadas no banco de dados *MySQL*. A API REST permitirá o cadastro de usuários e a visualização do histórico de uso dos compartimentos.

Figura 10 – Componentes da solução de guarda-volumes inteligentes com auditoria



Fonte: Autoria própria

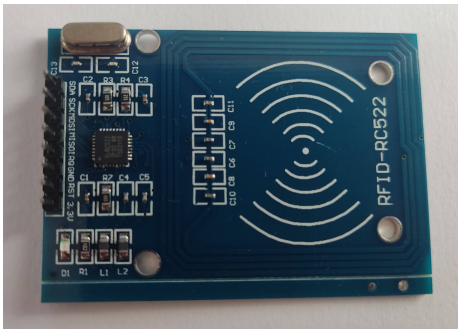
3.1 Conectividade dos Componentes

Nesta seção, serão abordadas as interconexões entre os principais componentes do sistema, destacando como eles se comunicam e interagem para garantir o funcionamento eficaz e coordenado da solução proposta. A seguir, cada componente será detalhado em termos de sua integração e conectividade com o microcontrolador e outros elementos do sistema. Esta análise detalhada mostra como as diferentes partes do projeto trabalham em conjunto para atingir os objetivos estabelecidos.

3.1.1 Leitor RFID

A comunicação entre o leitor RFID e o microcontrolador é realizada via protocolo SPI. No contexto deste projeto, o leitor RFID utiliza o SPI para transmitir as informações de autenticação dos usuários ao microcontrolador. A configuração do SPI no sistema envolve a conexão dos pinos MISO, MOSI, SCK e CS, garantindo a sincronização correta na transferência de dados. Na Figura 11 é apresentado o leitor RFID que será utilizado no projeto, mostrando a sua disposição e os pinos de conexão.

Figura 11 – Leitor RFID



Fonte: Autoria própria

3.1.2 *Reed Switch*

A comunicação entre o *Reed Switch* e o microcontrolador é realizada através de pinos GPIO configurados como entrada, permitindo que o microcontrolador leia diretamente o estado do sensor. O funcionamento do *Reed Switch* depende da presença de um ímã permanente instalado em cada porta do compartimento. Quando o ímã está próximo do sensor, como quando a porta está fechada, o sensor ativa e muda o estado do pino GPIO para indicar que o compartimento está seguro e fechado. Por outro lado, quando o ímã se afasta do sensor, como ao abrir a porta, o sensor desativa e muda o estado do pino GPIO para indicar que o compartimento está aberto. Essas mudanças de estado alteram o estado do pino GPIO, enviando um sinal ao microcontrolador para que as ações apropriadas sejam executadas.

3.1.3 Trava Elétrica Solenoide

A comunicação entre o microcontrolador e a trava elétrica solenoide é realizada através de pinos GPIO configurados como saída. Os pinos de saída permitem ao microcontrolador controlar diretamente a trava, enviando sinais elétricos que ativam ou desativam a solenoide. A trava solenoide é projetada para ser normalmente fechada, o que significa que na ausência de sinal elétrico, a solenoide mantém a trava fechada como posição de segurança padrão. Quando o pino GPIO é configurado para *HIGH*, a solenoide é energizada, acionando a abertura da trava. Quando configurado como *LOW*, desenergiza a solenoide, mantendo a trava fechada. Esses acionamentos são realizados com base nas informações recebidas do sensor *Reed Switch* associado àquela trava, certificando-se de que o compartimento se feche corretamente.

3.1.4 LEDs

A comunicação entre o microcontrolador e os LEDs de sinalização é realizada por meio de pinos GPIO configurados como saída. Essa configuração permite ao microcon-

trolador controlar diretamente os LEDs, ligando-os ou desligando-os conforme necessário para fornecer informações visuais ao usuário do comportamento do sistema, para que o usuário entenda o que está acontecendo. O MVP consiste em um total de 5 LEDs, divididos em duas categorias: status do compartimento, composto por um vermelho e um verde; e status da comunicação com o servidor, composto por um amarelo, um vermelho e um verde. A seguir pode-se visualizar o status dos LEDs.

Status do compartimento:

- Led vermelho: ligado continuamente por 2 segundos, sinaliza que a identificação do usuário não está cadastrada no sistema.
- Led verde: ligado continuamente até o final da operação, sinaliza que usuário está guardando pertences no armário. Piscando, sinaliza que o usuário está retirando pertences no armário.

Status da comunicação com o servidor:

- Led amarelo: processando alguma informação, o microcontrolador iniciou uma solicitação e está aguardando a finalização da operação.
- Led vermelho: comunicação com o servidor não estabelecida.
- Led verde: Indica prontidão para iniciar operações.

3.1.5 ESP32

O microcontrolador da ESP32 foi escolhido por suas capacidades avançadas que vão além das funções básicas de controle de dispositivos periféricos mencionados anteriormente, como SPI e GPIO. Uma das principais vantagens do ESP32 é sua conectividade Wi-Fi integrada, que permite a comunicação sem fio. Além disso, o ESP32 utiliza *NVS Flash* para armazenar dados críticos de forma persistente. Essa funcionalidade é essencial para manter as configurações do sistema, chaves de autenticação e estados operacionais, mesmo após reinicializações, garantindo a continuidade e segurança do sistema. A manutenção desses estados é realizada através do armazenamento de informações críticas em setores de memória não volátil, que não são apagados durante o ciclo de reinicialização. O sistema operacional em tempo real (RTOS) presente no ESP32 permite o gerenciamento eficiente de múltiplas tarefas, assegurando que todas essas tarefas sejam executadas de forma coordenada e em tempo real, o que é crucial para o bom funcionamento do sistema.

3.1.6 *Broker MQTT*

O sistema utiliza o protocolo MQTT, que opera com base no modelo publish/-subscribe, garantindo uma distribuição eficaz de mensagens entre dispositivos. Uma consideração importante é que, embora o MQTT facilite a comunicação em tempo real, ele requer que os dispositivos estejam online para receber as mensagens, pois não há uma cópia local das tags armazenada nos dispositivos. Isso implica que, em caso de perda de conexão, os dispositivos não poderão acessar informações atualizadas até que a conexão seja restaurada. Além disso, todas as mensagens são distribuídas para os dispositivos inscritos nos tópicos específicos, implica que todos os dispositivos embarcados recebam todas as mensagens.

A comunicação entre o controlador embarcado e o servidor é realizada utilizando diferentes tópicos para a troca de informações. Vale mencionar que todas as mensagens são distribuídas para os dispositivos inscritos nos tópicos específicos. No entanto, isso não implica que todos os dispositivos embarcados recebam todas as mensagens. Na Tabela 4, estão listados os tópicos utilizados no sistema, acompanhados de suas descrições e do conteúdo das mensagens (*payload*).

No tópico *door_command/request*, o *payload* no primeiro campo corresponde à identificação única do usuário que está solicitando a abertura do compartimento; o segundo campo contém o nome do armário registrado no sistema; e o terceiro campo indica o número total de compartimentos disponíveis naquele armário.

No tópico *door_command/response*, o *payload* no primeiro campo informa o número do compartimento do usuário; o segundo campo informa o nome do armário; e o terceiro campo confirma a identificação única do usuário autorizado.

No tópico *door/info*, o *payload* no primeiro campo registra o *timestamp* da abertura do compartimento, o segundo campo registra o *timestamp* do fechamento, o terceiro campo contém a identificação única do usuário autorizado, e o quarto campo especifica o número do compartimento utilizado. O registro dos horários ocorrem no próprio microcontrolador, já que um dos requisitos para o funcionamento é a conectividade com a internet.

Finalmente, o tópico *weird_activity* é usado para alertar sobre atividades suspeitas. Quando movimentos inesperados são detectados em compartimentos que deveriam estar estáticos, este tópico é acionado, enviando um alerta, informando imediatamente os administradores do sistema para que possam agir rapidamente em resposta ao incidente suspeito. O seu *payload* no primeiro campo informa o nome do armário; o segundo campo informa o número do compartimento do usuário.

Tabela 4 – Tópicos Estabelecidos para Mensagens Utilizando o Protocolo MQTT

Tópico	**Descrição**	***Payload***
door_command/request	Comando para solicitar abertura de compartimento	Identificação RFID Nome do armário Total de compartimentos
door_command/response	Resposta à solicitação de abertura de compartimento	Número do compartimento Nome do armário Identificação RFID
door/info	Registro de eventos de uso do compartimento	*Timestamp* *Timestamp* Identificação RFID Número do compartimento
weird_activity	Alerta de atividade suspeita	Nome do armário Número do compartimento

3.1.7 Modelagem do banco de dados *MySQL*

Para suportar as funcionalidades do sistema de gerenciamento da plataforma de gestão e auditoria, foi desenvolvida uma modelagem de banco de dados *MySQL*, composta por seis tabelas. Na Figura 12 pode ser visualizada a modelagem completa do banco de dados. Na tabela *user*, são armazenadas as informações básicas dos usuários, como nome, *e-mail*, *tag* RFID, (único para cada usuário), senha criptografada e o tipo de usuário (*admin* ou usuário comum). Essa tabela é consultada pela plataforma de gestão para fins de auditoria, troca de senha e uso dos compartimentos.

A tabela *locker* registra os armários, onde o nome de cada armário deve ser único dentro do sistema. Isso facilita a gestão de múltiplos armários de forma organizada. A tabela *compartment* armazena os compartimentos, associando cada um a um armário específico e garantindo, por meio de uma restrição de unicidade, que não exista compartimentos com números duplicados no mesmo armário.

A tabela *compartment_usage* é utilizada para rastrear quais compartimentos estão em uso, assegurando que cada usuário possa acessar apenas um compartimento por vez. A tabela *locker_schedule* armazena as programações de uso dos compartimentos, incluindo horários de abertura, fechamento, retirada e término da retirada dos itens, informações essenciais para auditoria.

Por fim, a tabela *forgot_password* armazena um código único gerado para cada usuário, a data em que o código foi gerado e o usuário associado, para que possa efetuar a troca de senha em casos de esquecimento.

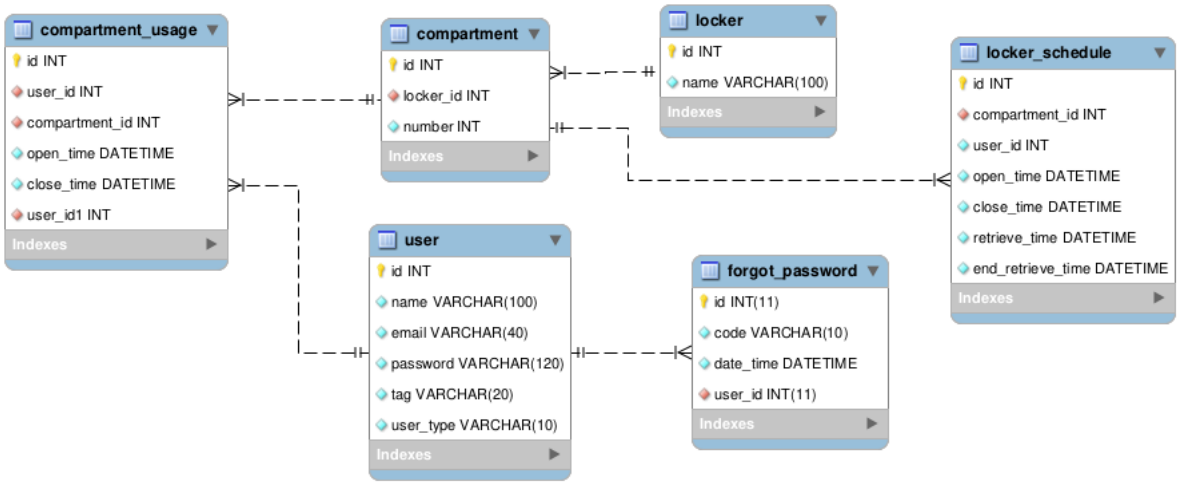


Figura 12 – Modelagem do Banco de Dados da aplicaçao.

Fonte: Autoria própria

3.1.8 API REST

A Tabela 5 apresenta as rotas da API REST disponíveis para interação com a plataforma de gerenciamento do sistema. A tabela inclui rotas para operações essenciais, como autenticação de usuários, gerenciamento de sessões e manutenção de perfil. A rota / renderiza a página principal de navegação, exibindo o histórico de utilização dos armários para usuários logados. A rota /login usa o método HTTP GET para exibir o formulário de login e o método POST para processar a autenticação do usuário. A rota /sair realiza o logout do usuário, encerrando a sessão atual. A rota */register* exibe o formulário de registro com o método GET e processa o cadastro de novos usuários com o método POST. A rota */update-email* permite a atualização do e-mail do usuário com o método GET para exibir o formulário e o método POST para processar a atualização. Por fim, a rota */update-password* exibe o formulário de atualização de senha com o método GET e permite que os usuários atualizem suas senhas com o método POST.

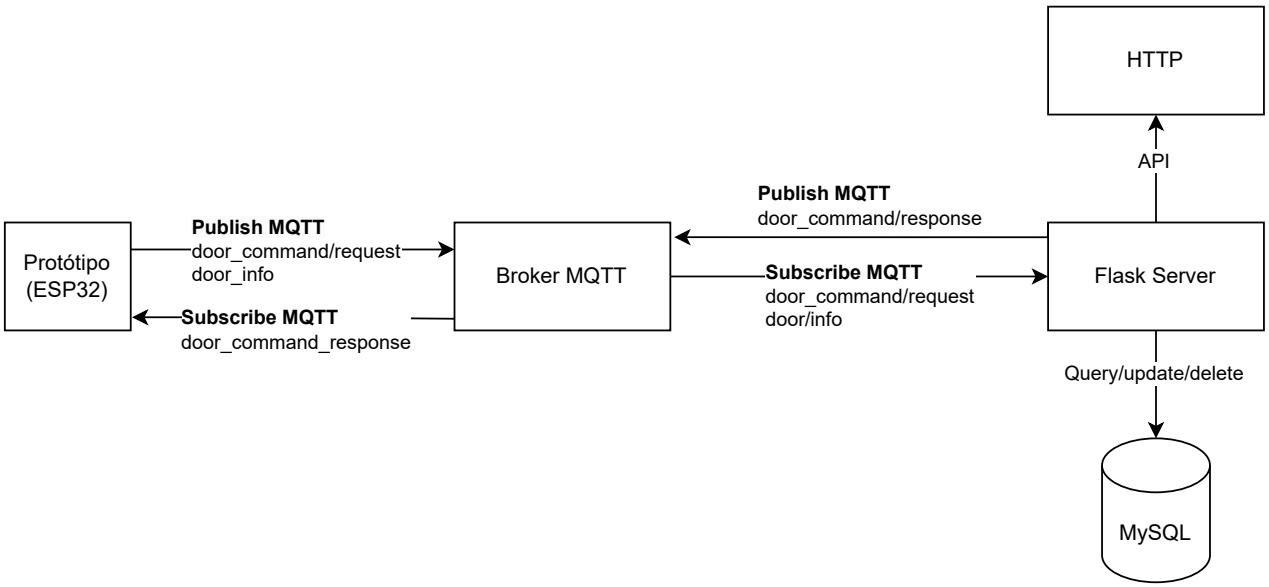
Tabela 5 – Rotas da API REST

Rota	**Método HTTP**
/	GET
/login	GET, POST
/sair	GET
/register	GET, POST
/update-email	GET, POST
/update-password	GET, POST
/edit_number	GET, POST
/forgot_password	GET, POST
/reset_password	GET, POST
/check_email	GET, POST
/update	GET, POST

3.2 Fluxo de comunicação no Sistema

O fluxo da comunicação do protocolo MQTT entre o servidor e o controlador embarcado é ilustrado na Figura 13. A comunicação é iniciada pelo controlador embarcado, que, após a autenticação de um usuário, publica uma mensagem no tópico *door_command/request*. O servidor, que deve ter previamente realizado a assinatura (*subscribe*) desse tópico durante sua inicialização, recebe a mensagem e, se tudo estiver correto, publica uma resposta no tópico *door_command/response*. O controlador embarcado, que também deve ter assinado esse tópico durante sua inicialização, recebe a resposta. Após a finalização da interação do usuário com o compartimento, o controlador embarcado publica uma mensagem no tópico *door/info*, ao qual o servidor também deve estar inscrito.

Figura 13 – Fluxo da comunicação no servidor



Fonte: Autoria própria

3.3 Requisitos do sistema

Nesta seção, são detalhados os requisitos essenciais para o desenvolvimento do MVP de guarda-volumes inteligentes com auditoria. Os requisitos aqui descritos abrangem as funcionalidades essenciais e as características de qualidade necessárias para garantir o funcionamento eficiente e eficaz do sistema. Incluem-se tanto os requisitos funcionais, não funcionais, regras de negócio e casos de uso que orientam o comportamento do sistema e suas interações.

Os requisitos funcionais do sistema estão apresentados na Tabela 6, que detalha as principais funções que o sistema deve desempenhar. O RF01 garante que apenas usuários autorizados possam acessar o sistema. O RF02 e o RF05 asseguram que os administradores tenham total controle sobre quem pode acessar os compartimentos. O RF03 motiva os usuários a manterem suas chaves de acesso em segurança, uma vez que se trata de um objeto de uso pessoal. Por fim, o RF04 assegura que o usuário esteja sempre ciente do status atual do sistema e do compartimento com o qual está interagindo.

Tabela 6 – Requisitos funcionais do sistema

Código	**Funções**
RF01	Realizar o login usando e-mail e senha;
RF02	Permitir que administradores criem perfil de usuários;
RF03	Permitir que usuários guardem seus objetos utilizando como autenticação seu cartão RFID e retirem seus objetos utilizando o mesmo cartão;
RF04	Permitir que o status do sistema possa ser monitorado através de LEDs;
RF05	Registrar histórico de utilização dos compartimentos e permitir auditoria.
RF06	Permitir que usuários consigam trocar suas senhas do acesso ao servidor.

Na Tabela 7, encontram-se os requisitos não funcionais, que especificam as características de qualidade e as restrições técnicas do sistema. A tabela mostra, na categoria de segurança, a importância do armazenamento de senhas criptografadas que seguem as boas práticas de segurança cibernética. Além disso, restringir a criação e atualização de perfis de usuário apenas a administradores e garantir acesso à auditoria protege o sistema contra ações indevidas ou mal-intencionadas, possibilitando uma resposta rápida a quaisquer incidentes de segurança.

Nas categorias seguintes de confiabilidade e usabilidade, com a capacidade do sistema de detectar falhas na leitura de cartões RFID ou outras falhas do sistema e notificar o usuário através de LEDs, aborda-se a melhoria da eficiência do processo de acesso aos compartimentos. Na categoria de disponibilidade, a eficácia do gerenciamento dos compartimentos é assegurada pela capacidade de conectar múltiplas centrais ao mesmo servidor. Cada controlador embarcado envia seu identificador junto com as informações via MQTT, o que permite ao servidor reconhecer e gerenciar diferentes centrais. Isso evita a duplicação de uso, garantindo que os usuários tenham acesso a apenas um compartimento, mesmo que haja várias centrais no estabelecimento.

Já na categoria de compatibilidade, o uso de cartões do tipo MIFARE aumenta a acessibilidade, pois esses cartões, amplamente utilizados em diversas aplicações do dia a dia, permitem que os usuários utilizem seus próprios cartões, que já possuem outras

funcionalidades, para acessar os compartimentos. Como esses cartões são objetos de uso pessoal, os usuários tendem a cuidar melhor deles e a evitar emprestá-los a terceiros. Por fim, a modularidade do sistema permite sua expansão conforme a necessidade dos clientes.

Tabela 7 – Requisitos Não Funcionais do Sistema

Categoria	**Descrição**
Segurança	Senha dos usuários armazenadas no banco de dados criptografada;
	Apenas usuários com privilégios de administrador devem ter permissão para criar e atualizar perfis de usuários;
	O sistema deve registrar todas as operações de autenticação e transações de guarda e retirada de objetos para fins de auditoria e segurança;
Confiabilidade	O sistema deve detectar de forma eficaz falhas de leitura de cartões RFID ou erros no processo de autenticação, fornecendo *feedback* ao usuário;
Disponibilidade	O armário precisa estar conectado no *Wi-Fi*, para permitir que os usuários acessem seus objetos a qualquer momento;
Compatibilidade	O sistema utiliza apenas o cartão RFID do tipo MIFARE para autenticação, o qual muitos usuários já possuem por conta de cartão de crédito, cartão de ônibus, entre outros;
Usabilidade	O sistema deve fornecer *feedback* visual para indicar o sucesso ou falha das operações de autenticação e acesso ao compartimento;
	O sistema deve atualizar o status dos LEDs em tempo real ou quase em tempo real, proporcionando *feedback* imediato sobre o estado atual do sistema;
	Interface intuitiva de usar;
Modularidade	O sistema pode ser expandido para mais compartimentos por central;

Já na Tabela 8, são listadas as regras de negócio que orientam o funcionamento do sistema, garantindo que as operações realizadas estejam em conformidade com as políticas e diretrizes estabelecidas.

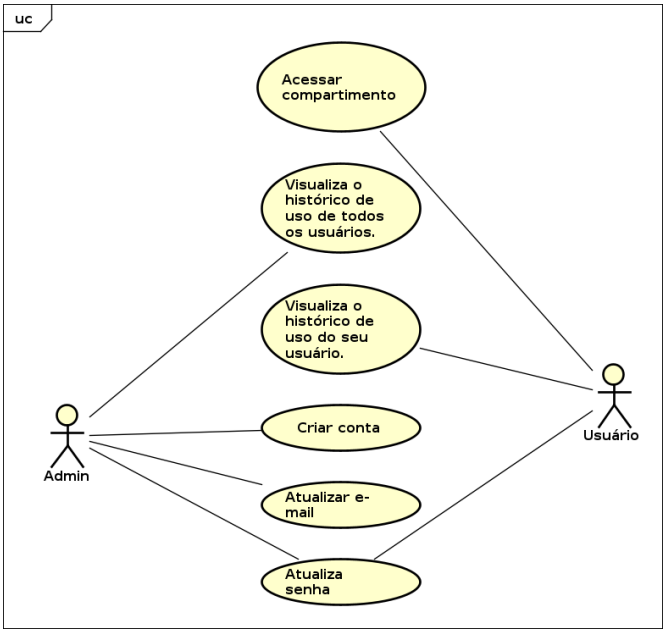
Os casos de uso do servidor estão ilustrados na Figura 14, onde dois atores principais interagem com o sistema: o administrador e o usuário comum. O diagrama destaca as principais funcionalidades do sistema. O usuário comum pode atualizar sua senha e visualizar o seu próprio histórico de uso. Já o administrador tem permissões mais amplas, como a criação de novas contas, a visualização do histórico de uso de todos os usuários, além de atualizar *e-mails* e senhas.

O caso de uso do armário ilustrado na Figura 15. Assim como no servidor, os

Tabela 8 – Regras de Negócio do Sistema

Regra	Descrição
1	Cartão RFID exclusivamente para autenticação; caso seja um cartão de crédito, não irá consumir o saldo do cartão.
2	O usuário deve utilizar o armário apenas quando necessário.
3	O usuário deve garantir que o compartimento está fechado ao fim do uso para finalizar sua sessão no sistema.

Figura 14 – Casos de Uso do servidor



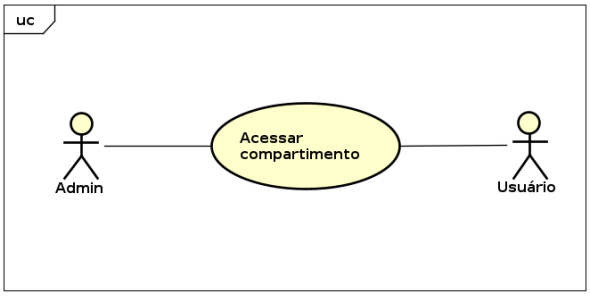
Fonte: Autoria própria

atores principais são o administrador e o usuário comum. O usuário pode acessar diretamente um compartimento para armazenar seus pertences. O administrador, por sua vez, pode acessar compartimentos em uso, especialmente em situações onde, por exemplo, um usuário precisa recuperar seus pertences, mas por algum motivo não está sob posse de sua autenticação. Nesse caso, o administrador se associa ao compartimento através do servidor para permitir o acesso necessário.

3.4 Diagrama de blocos do hardware

O diagrama de blocos do *hardware* ilustra a disposição e a funcionalidade dos principais componentes do sistema. No diagrama, que pode ser visto na Figura 16, é possível entender como o sistema opera. Um leitor RFID, conectado via SPI ao microcontrolador ESP32, gerencia dois compartimentos. Conforme a interação entre os componentes do sistema e dos usuários, o estado de cada compartimento é indicado visualmente por LEDs.

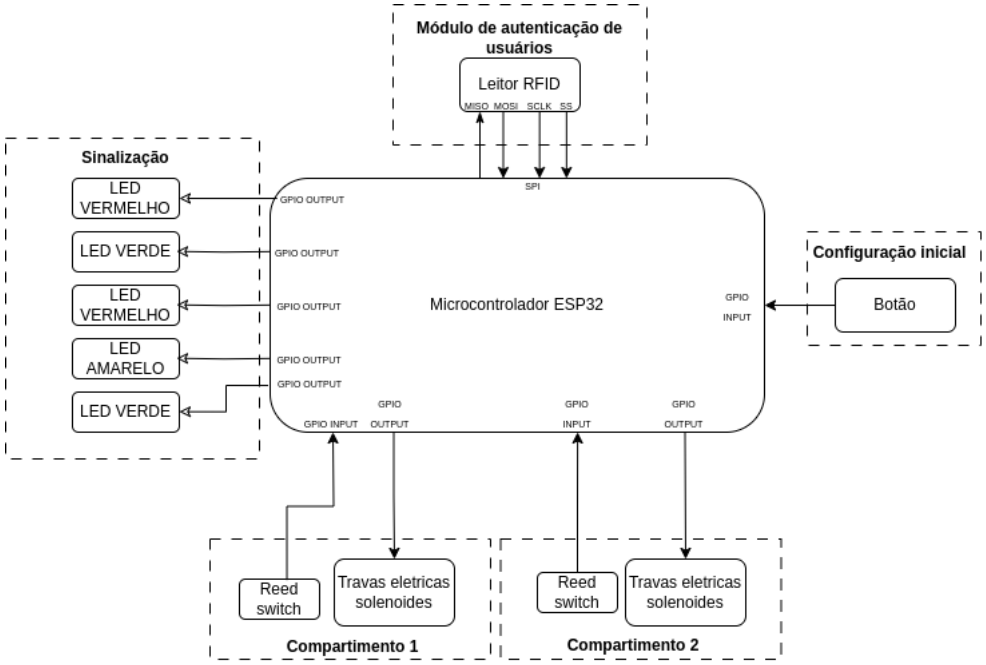
Figura 15 – Caso de Uso do armário



Fonte: Autoria própria

Além disso, um botão é utilizado para a configuração inicial do dispositivo, facilitando a preparação e personalização do sistema antes do uso.

Figura 16 – Diagrama de blocos de hardware



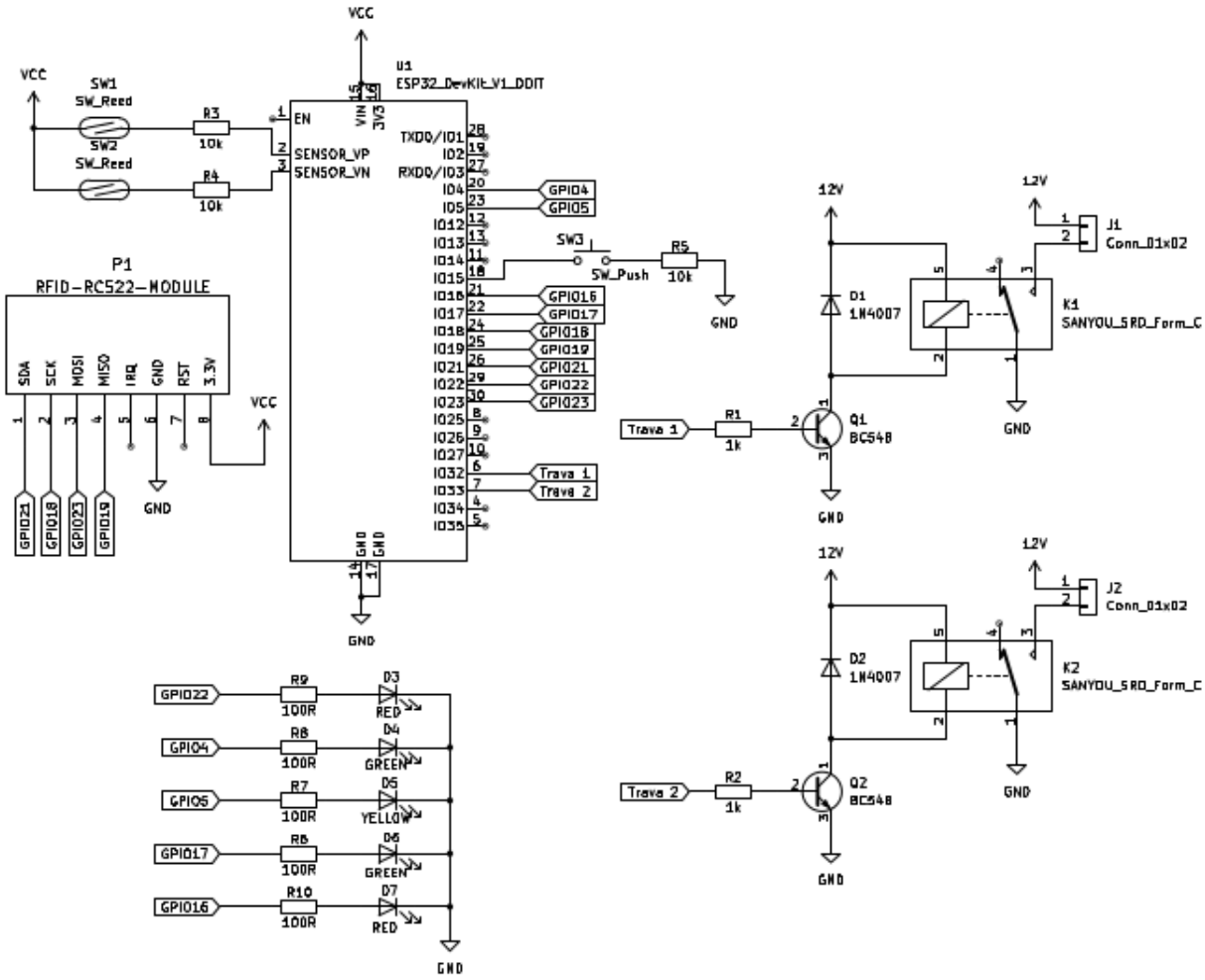
Fonte: Autoria própria

3.5 Esquemático eletrônico do protótipo

O diagrama esquemático eletrônico do circuito representa como os componentes eletrônicos do sistema estão conectados entre si. Ele é usado para planejar e entender o funcionamento do circuito antes de construir o protótipo físico. No diagrama apresentado na Figura 17, é possível observar a interação entre os principais componentes do sistema. O leitor RFID é responsável por capturar a identificação do usuário, que é enviada ao microcontrolador ESP32. Este, por sua vez, processa as informações e comunica-se com o servidor por meio do protocolo MQTT para validar a autenticação e liberar o acesso

ao armário. A trava eletromecânica é controlada diretamente pelo ESP32, enquanto os LEDs fornecem *feedback* visual ao usuário, indicando o status do armário (aberto, fechado ou erro).

Figura 17 – Esquemático eletrônico



Fonte: Autoria própria

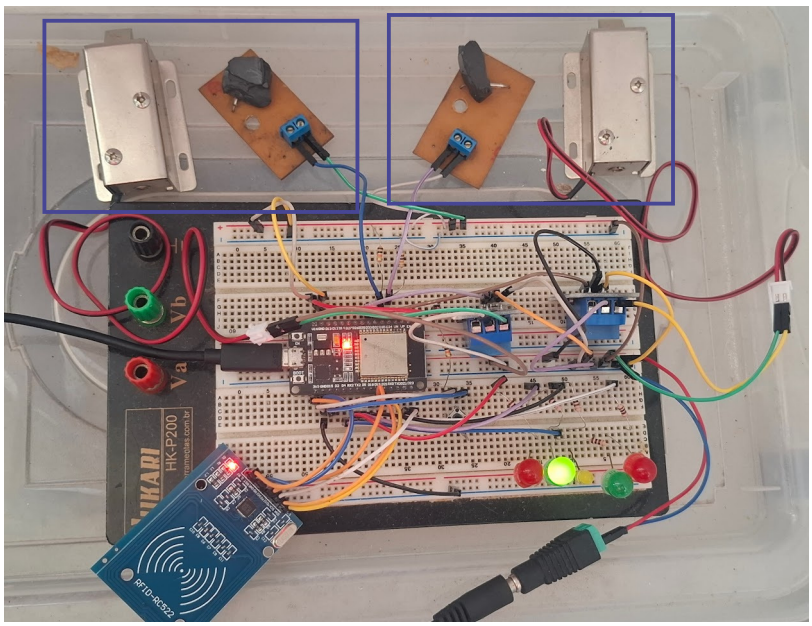
4 RESULTADOS

Esta seção apresenta os resultados obtidos com o desenvolvimento e a implementação do MVP de armários inteligentes, analisando o desempenho das funcionalidades propostas. Os resultados incluem a validação dos componentes eletromecânicos integrados no hardware, o sistema de autenticação, a integração da comunicação com o servidor e a usabilidade da interface administrativa e de usuários comuns.

4.0.1 Validação dos componentes eletromecânicos

A Figura 18 ilustra o protótipo da placa, que foi desenvolvido em uma *protoboard* para validar seu funcionamento. Na imagem, são destacados dois compartimentos na *protoboard*, indicando a organização e a disposição dos componentes utilizados no protótipo. Esta montagem inicial foi crucial para testar as funcionalidades individuais da placa antes da etapa de desenvolvimento, integrando-as[1].

Figura 18 – Protótipo



Fonte: Autoria própria

A implementação das travas elétricas solenoides e operando com uma fonte externa com tensão de 12V com consumo de corrente de 600mA, mostrou-se altamente eficaz na proteção dos itens armazenados. Este sistema de segurança é projetado para manter os compartimentos normalmente fechados, mesmo durante interrupções no fornecimento de

[1] O código-fonte está disponível em https://github.com/luizaalves/Smart-Lockers-firmware.

energia, assegurando que os itens fiquem protegidos até que as condições operacionais normais sejam restauradas.

A utilização de um sensor magnético provou ser crucial para a segurança do sistema. Esse sensor auxilia a trava elétrica a determinar o momento exato para ativar e travar o compartimento, assegurando que este seja fechado adequadamente. Além disso, o sensor é fundamental para a segurança proativa, pois está configurado para enviar notificações por *e-mail* em situações de arrombamento para todos os administradores do sistema. Esse mecanismo é ativado quando há movimentação inesperada em um compartimento que deveria estar estático, enviando imediatamente um alerta de atividade suspeita. Esse método de alerta garante que os responsáveis sejam prontamente informados sobre quaisquer incidentes, permitindo uma resposta rápida e eficaz.

Importante destacar que o sistema de alerta depende da conectividade de rede para funcionar. Em caso de falha na rede, os e-mails de alerta não serão enviados, pois o sistema não tem capacidade de operar offline. Esta limitação não está no escopo do projeto atual, que visa apenas validar a funcionalidade do sistema sob condições operacionais normais. Assim, a confiabilidade do sistema em situações de falha de rede não é garantida.

Durante os testes, o armário foi equipado com dois compartimentos; no entanto, para a implementação de sistemas com um número maior de compartimentos, recomenda-se a utilização de um expansor de portas GPIO. Além disso, a escolha de um microcontrolador da série ESP32 provou ser benéfica devido ao seu kit robusto que facilita a integração com *Wi-Fi*. As bibliotecas disponíveis, acompanhadas de exemplos na documentação da Espressif, auxiliaram o processo de desenvolvimento.

4.0.2 Validação do sistema de autenticação

O leitor RFID se mostrou eficaz para a autenticação dos compartimentos do armário inteligente. Ele permitiu o cadastro de *tags* RFID, cartão de crédito, cartões de ônibus e NFC do celular. Abrindo assim uma gama maior de possibilidades de chaves de acesso que os usuários possam usar. Vale lembrar que o servidor está autorizado a passar algum compartimento em uso para o nome de algum administrador do sistema, em casos de perda de autenticação.

4.0.3 Validação da comunicação com o servidor

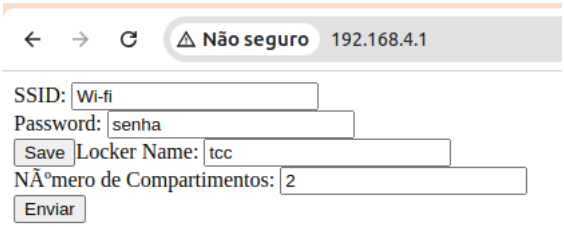
A implementação de um servidor MQTT como intermediário na troca de mensagens entre o armário e o servidor central demonstrou eficácia. Esse protocolo oferece respostas rápidas e uma comunicação fluida e em tempo real, essenciais para a operação eficaz do sistema. A facilidade de implementação é outro ponto forte, beneficiada pela ampla disponibilidade de documentação detalhada, que auxilia significativamente no

desenvolvimento. Além disso, o MQTT garante a entrega das mensagens, um aspecto crucial para manter a confiabilidade do sistema. A arquitetura baseada no modelo *publish/subscribe* do protocolo facilita a expansão do sistema, permitindo a adição de novos tópicos de forma simples e eficiente, o que é ideal para sistemas em escala ou que preveem atualizações e ampliações futuras.

4.0.4 Validação da usabilidade da interface homem-máquina - Protótipo

O protótipo inclui um botão para ativar o modo *Access Point* (AP), que é essencial para configurar o armário. Ao ativar este modo, é necessário conectar-se à rede *Wi-Fi* gerada pelo armário usando o SSID: LOCKER. Após estabelecer a conexão, o usuário deve abrir um navegador de internet e digitar o endereço IP 192.168.4.1 para acessar a interface de configuração, que pode ser vista na Figura 19. Esta funcionalidade é projetada especificamente para facilitar a configuração inicial do sistema, permitindo ao usuário ajustar as configurações necessárias diretamente através de uma interface gráfica amigável e intuitiva.

Figura 19 – Modo *Acess Poin*t para configuração do protótipo



Fonte: Autoria própria

Os LEDs na placa sinalizam o status dos compartimentos do armário e a comunicação com o servidor. Eles indicam claramente se o armário está funcionando corretamente e disponível para uso, se está processando alguma informação, se não há nenhum compartimento livre ou se um compartimento está aberto. No entanto, uma possível melhoria para este sistema de sinalização poderia ser a implementação de um LED RGB. O uso de um LED RGB teria a vantagem de utilizar menos pinos do microcontrolador, mantendo a mesma funcionalidade.

4.0.5 Validação da usabilidade da interface homem-máquina - Servidor

A implementação do servidor é multifuncional, validando a abertura dos compartimentos, no armazenamento de dados e na gestão da interface de usuário para administradores e usuários comuns. O desenvolvimento do servidor incluiu a criação de um histórico de uso dos armários e o registro de novos usuários e chaves de acesso, utilizando-se de tecnologias como banco de dados e servidor MQTT. Para a interface das telas, optou-se

pelo uso do *framework Flask* em Python, garantindo uma interação eficaz e intuitiva. Além disso, o sistema é equipado com funcionalidades de segurança avançadas, como o envio de alertas por *e-mail* em casos de arrombamento dos compartimentos e recuperação de senhas, onde um código é enviado por *e-mail* para o cadastro de novos usuários e a geração de novas chaves de acesso para os compartimentos. Essas características asseguram uma plataforma segura, confiável e de fácil uso para todos os envolvidos[2].

4.0.5.1 Interfaces do Sistema servidor

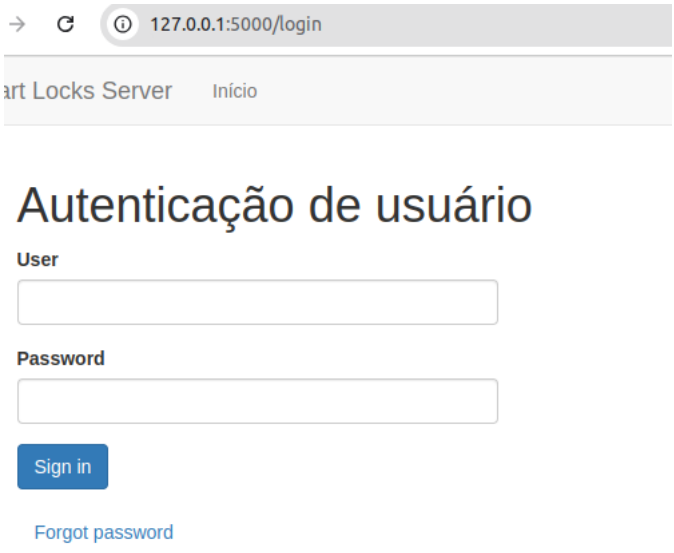
A seção a seguir detalha as interfaces do sistema servidor, destacando as funcionalidades essenciais disponíveis para usuários e administradores através do *front-end* desenvolvido.

A interação com o sistema inicia-se na tela de login, ilustrada na Figura 20, onde o usuário precisa autenticar-se usando um *e-mail* e senha válidos. Caso o usuário esqueça sua senha, pode acessar a tela de recuperação de senha, como mostrado na Figura 21, onde um código de recuperação é enviado por *e-mail*. Uma vez autenticado, o usuário é direcionado para a tela principal, onde as informações variam conforme o tipo de usuário. Usuários administradores visualizam o histórico de uso de todos os armários e os compartimentos em uso, conforme representado na Figura 22. Usuários comuns veem apenas as informações relacionadas aos seus próprios compartimentos, como visto na Figura 23.

Além dessas funcionalidades, os administradores têm a capacidade de registrar novos usuários através da tela mostrada na Figura 24, atualizar o *e-mail* de um usuário na tela exibida na Figura 25, e alterar a senha de qualquer usuário, conforme ilustrado na Figura 26. Usuários comuns também têm a possibilidade de alterar suas próprias senhas.

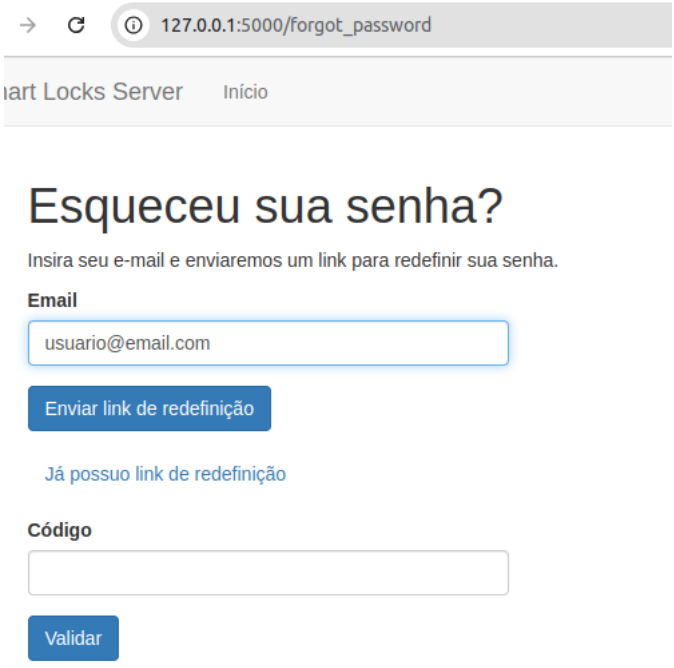
[2] O código-fonte está disponível em https://github.com/luizaalves/Smart-Lockers-server.

Figura 20 – Tela de login



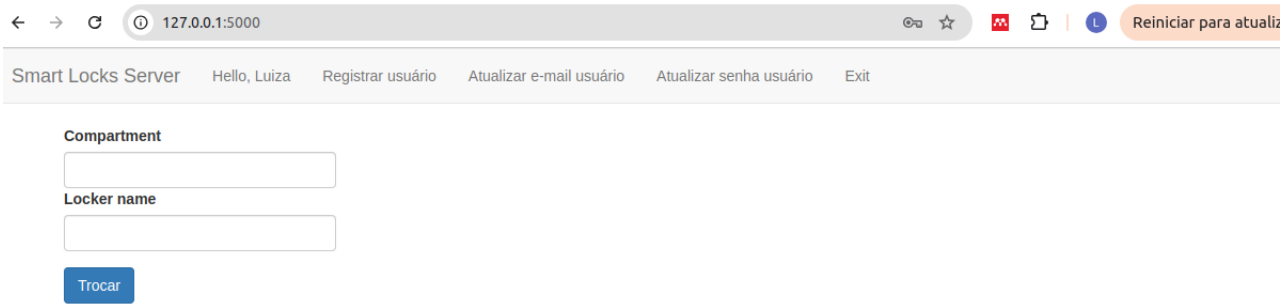
Fonte: Autoria própria

Figura 21 – Tela de recuperação de senha



Fonte: Autoria própria

Figura 22 – Tela principal



Compartment in Use

User name	Stored Items Date		Compartment Number	Locker name
	Open Time	Close Time		
Roseli	2025-01-18 09:15:05	2025-01-18 09:15:06	2	tcc

Locker Schedule

User name	Stored Items Date		Removed Items Date		Compartment Number	Locker name
	Open Time	Close Time	Retrieve Time	End Retrieve Time		
Yan	2024-07-31 14:05:03	2024-07-31 14:05:11	2024-07-31 14:08:53	2024-07-31 14:08:57	1	tcc
Luiza	2024-07-31 14:07:34	2024-07-31 14:07:45	2024-07-31 14:09:16	2024-07-31 14:09:20	2	tcc
Luiza	2024-07-31 14:09:36	2024-07-31 14:09:39	2024-07-31 14:10:01	2024-07-31 14:10:03	1	tcc

Fonte: Autoria própria

Figura 23 – Tela de principal de usuário comum

Compartment in Use

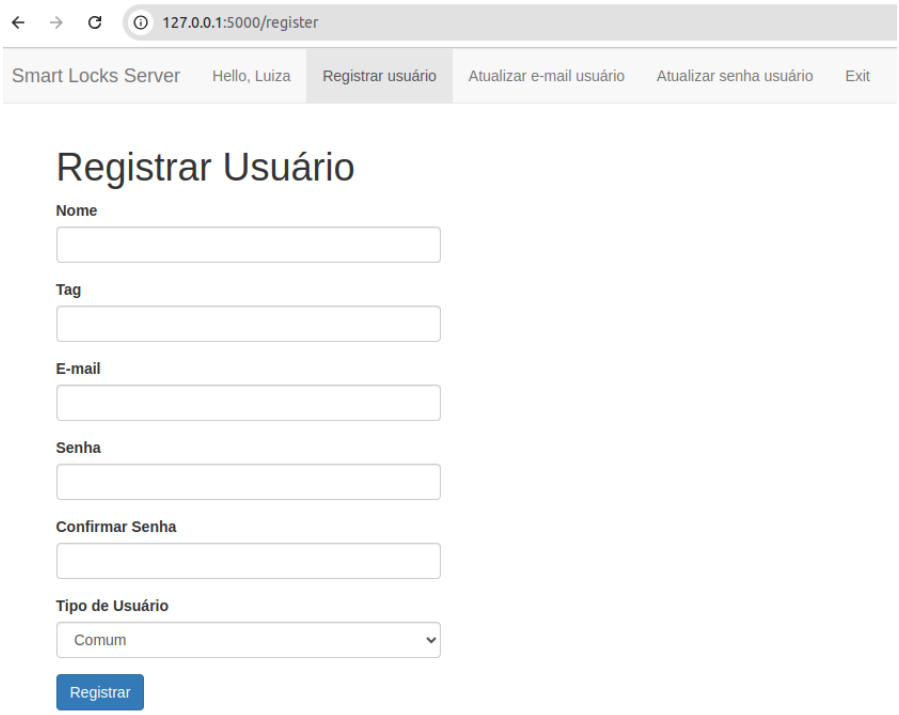
User name	Stored Items Date		Compartment Number	Locker name
	Open Time	Close Time		
Roseli	2025-01-18 09:15:05	2025-01-18 09:15:06	2	tcc

Locker Schedule

User name	Stored Items Date		Removed Items Date		Compartment Number	Locker name
	Open Time	Close Time	Retrieve Time	End Retrieve Time		
Roseli	2024-08-07 14:22:08	2024-08-07 14:22:11	2024-08-07 14:23:22	2024-08-07 14:23:23	2	tcc
Roseli	2024-08-07 14:25:04	2024-08-07 14:25:05	2024-09-14 16:18:31	2024-09-14 16:18:34	2	tcc
Roseli	2024-09-14 16:20:44	2024-09-14 16:20:44	2024-09-14 16:20:54	2024-09-14 16:20:57	2	tcc

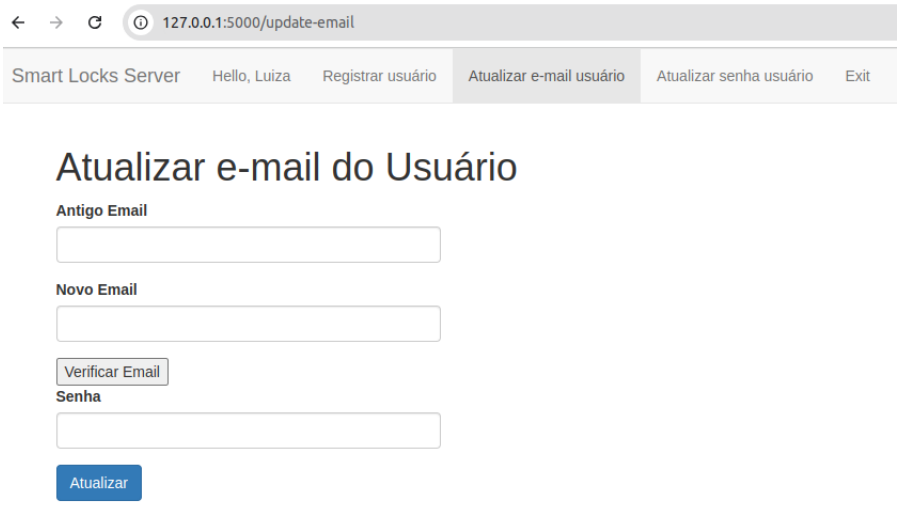
Fonte: Autoria própria

Figura 24 – Tela de registro de usuário



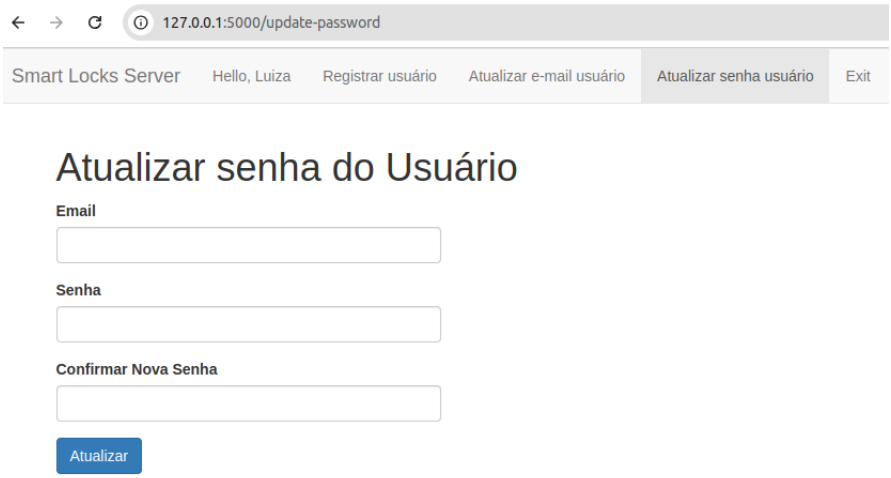
Fonte: Autoria própria

Figura 25 – Tela de atualização de *e-mail* do usuário



Fonte: Autoria própria

Figura 26 – Tela de atualização de senha do usuário



Fonte: Autoria própria

5 CONCLUSÕES

A construção e implementação deste MVP de armários inteligentes com auditoria alcançaram o objetivo de automatizar e modernizar o sistema de guarda-volumes. As travas elétricas solenoides, operando a 12V e com um consumo de corrente de 600mA, demonstraram robustez, mantendo a segurança dos itens armazenados, inclusive durante quedas de energia. No entanto, uma limitação observada é a dependência de energia elétrica para operação, que poderia ser mitigada com a integração de um sistema de *nobreak*, bateria e rede TCP/IP, aspecto não explorado neste trabalho.

Outra questão relacionada ao protótipo diz respeito à configuração inicial, durante a qual, por meio de um botão, cria-se uma rede Access Point que permite ao usuário atribuir nomes ao armário e aos compartimentos. Apesar de essa funcionalidade proporcionar flexibilidade, ela pode acarretar complicações, como a inconsistência nos nomes devido à falta de restrições, o que pode dificultar a gestão e o rastreamento eficazes dos armários em ambientes maiores. Além disso, o sistema presume que o usuário respeite as regras de negócio estabelecidas, permitindo apenas uma única abertura. Caso o usuário esqueça de guardar algum objeto e necessite abrir o compartimento novamente, ele perderá aquele compartimento; este aspecto também não foi explorado neste trabalho. Outro ponto não abordado neste MVP é a dependência de conectividade com a internet: sem ela, o armário não funciona offline. Uma possível melhoria seria implementar um cache local, salvando os dados na memória flash.

A autenticação de usuários foi implementada com sucesso através de tecnologia RFID e NFC, permitindo o acesso automatizado aos compartimentos. Utilizar dispositivos pessoais como cartões ou celulares para a autenticação mostrou-se uma estratégia eficaz, reduzindo a necessidade de distribuir chaves físicas e minimizando o risco de cópias indevidas. Este sistema não só reforça a segurança como também aprimora a conveniência para os usuários, embora não tenham sido feitos testes suficientes para garantir que não gerem sinais NFC aleatórios.

A interface do usuário desenvolvida para o armário, baseada em LEDs, foi projetada para validar a funcionalidade do protótipo sem focar em aspectos estéticos ou de usabilidade avançada. Embora cada LED indicasse o status do sistema, a falta de sinalização clara nos compartimentos se mostrou uma limitação, especialmente em armários com muitos compartimentos. Esta interface simplificada cumpriu seu papel inicial de validação, mas sua expansão para testes mais abrangentes exigirá melhorias significativas em design e funcionalidade.

O desenvolvimento de um serviço e aplicativo para o cadastro e controle de usuários

foi vital para o gerenciamento eficaz do sistema. A robustez da comunicação foi assegurada pelo uso do protocolo MQTT, enquanto o servidor, desenvolvido em Python com Flask e *MySQL*, provou ser adequado para as necessidades atuais do MVP. O banco de dados poderia permitir a implementação de mais de uma tag por usuário, tornando assim o sistema com mais flexibilidade para o usuário. Se houver uma expansão dos guarda-volumes, o uso da implementação de topicos através de publish/subscribe se torna inviavel pelo fato de que pode causar um atraso no recebimento das mensagens. Futuras iterações do projeto poderão incluir a integração do armário em um ambiente real, expandindo a capacidade de compartimentos para testes mais extensivos e aplicação prática.

O desenvolvimento e teste do MVP confirmaram a viabilidade técnica e operacional do projeto de armários inteligentes. Estes testes foram cruciais para identificar áreas de melhoria e ajustes necessários para futuras iterações do produto. Em resumo, o projeto cumpriu seus objetivos, evidenciando que a integração de controle eletrônico e sistemas de auditoria pode elevar significativamente a segurança e a experiência do usuário.

5.1 Trabalhos futuros

Considerando algumas funcionalidades do projeto que poderiam ter de maneira mais eficiente, abre um vasto campo para futuras pesquisas e desenvolvimentos que podem enriquecer e estender a funcionalidade do sistema. Para trabalhos futuros, recomenda-se explorar as seguintes melhorias e expansões, criando assim um sistema mais completo e eficiente. Algumas áreas recomendadas para a continuação do projeto incluem:

- Em cada compartimento uma sinalização para o usuário poder saber onde guardar seus objetos em casos de muitos compartimentos associados a aquele armário;
- LED RGB para interface do usuário dos armários inteligentes ao invés de vários LEDs de diferentes cores;
- Expansor de portas GPIO para ter a possibilidades de incluir mais compartimentos, já que somente com os pinos da ESP32, há limitação um número limitado;
- Implementação de um protótipo para testes em campo, em um cenário de uso real para validação do produto.

REFERÊNCIAS

ALCIATORE, D. G.; HISTAND, M. B. *Introdução à mecatrônica e aos sistemas de medições*. [S.l.]: Grupo A, 2014. ISBN 9788580553413. 35

ALMEIDA, R. de. *Programação de Sistemas Embarcados - Desenvolvendo Software para Microcontroladores em Linguagem C*. [S.l.]: Grupo GEN, 2016. ISBN 9788595156371. 31

AMAZON. *ULP*. 2022. https://www.amazon.com/ulp/view. Acesso em: 14 nov. 2022. 22

CALPOLY. *Center for Global Automatic Identification Technologies*. Poly Gait, 2016. (Acesso em: 11 set. 2023). Disponível em: https://polygait.calpoly.edu/. 33

CARDOSO, J. *Engenharia Eletromagnética*. [S.l.]: Grupo GEN, 2010. ISBN 9788595156975. 29

COULORIS, G. et al. *Sistemas distribuídos*. [S.l.]: Grupo A, 2013. ISBN 9788582600542. 37

DURATTA. *Armário Biométrico*. 2023. https://www.duratta.com.br/produto/armario-biometrico/. Acesso em: 03 dez. 2023. 26

ESPRESSIF. *ESP32 Series Datasheet*. [S.l.], 2023. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. Acesso em: 10 out. 2023. 31

FERREIRA, A. G. *Interface de programação de aplicações (API) e web services*. [S.l.]: Editora Saraiva, 2021. ISBN 9786553560338. 39

FOROUZAN, B. A. *Comunicação de dados e redes de computadores*. [S.l.]: Grupo A, 2010. ISBN 9788563308474. 36

GOODWIN, M. *O que é uma API?* 2024. Publicado em: 9 abr. 2024, Acesso em: 9 set. 2024. Disponível em: https://www.ibm.com/br-pt/topics/api. 40

HIVEMQ. *MQTT MQTT 5 Essentials*. [S.l.]: HiveMQ GmbH, 2020. 38

IBM. *O que são message brokers?* 2023. (Acesso em: 13 out. 2023). Disponível em: https://www.ibm.com/br-pt/topics/message-brokers. 37

IFSC. *Edital 005/2019: Chave dos armários em 2019.2*. 2022. https://www.ifsc.edu.br/documents/35941/1078237/EDITAL+005-2019+chave+dos+arm%C3%A1rios+em+2019.2.pdf. Acesso em: 14 nov. 2022. 21

IFSC. *Estudantes*. 2022. https://www.ifsc.edu.br/web/campus-sao-jose/estudantes. Acesso em: 14 nov. 2022. 21

INTELIGENTES, A. *Soluções para guarda de volumes*. 2023. https://www.armariosinteligentes.com.br/solucoes/guarda-volumes-inteligente/. Acesso em: 07 abril 2023. 26

JÚNIOR, S. L. S.; FARINELLI, F. A. *DOMÓTICA - AUTOMAÇÃO RESIDENCIAL E CASAS INTELIGENTES COM ARDUINO E ESP826.* [S.l.]: Editora Saraiva, 2018. ISBN 9788536530055. 32, 33, 36, 37

MALVINO, A. P.; BATES, D. J. *Eletrônica. v.1.* Grupo A, 2016. ISBN 9788580555776. Disponível em: https://app.minhabiblioteca.com.br/#/books/9788580555776/. 35

MORAES, A. de; HAYASHI, V. T. *Segurança em IoT.* [S.l.]: Alta Books, 2021. ISBN 9788550816548. 38

MORAES, C. C. de; CASTRUCCI, P. de L. *Engenharia de Automação Industrial.* [S.l.]: Grupo GEN, 2006. ISBN 9788521619765. 30

NXP Semiconductors. *AN10833 - MIFARE and contactless cards in access management.* [S.l.], 2011. Acesso em: 25 agosto 2024. Disponível em: https://www.nxp.com/docs/en/application-note/AN10833.pdf. 34

NXP Semiconductors. *MFRC522 - Standard performance MIFARE and NTAG frontend.* [S.l.], 2016. Acesso em: 25 agosto 2024. Disponível em: https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf. 34

OIHANDOVER. *Oihandover - Soluções.* 2022. https://oihandover.com/#solucoes. Acesso em: 14 nov. 2022. 22

OPPITZ. *Armário inteligente para guarda volumes / capacetes em shoppings.* 2023. https://oppitz.com.br/produtos/armarios-inteligentes/. Acesso em: 7 nov. 2023. 26

ROCHOL, J. *Sistemas de comunicação sem fio.* [S.l.]: Grupo A, 2018. ISBN 9788582604564. 34, 36

SCHIER. *Armario guarda-volume.* [S.l.], 2023. Disponível em: [https://schiermoveis.com.br/armario-guarda-volume-o-que-e. Acesso em: 19 fev. 2024. 25

SILBERSCHATZ, A. *Sistema de Banco de Dados.* [S.l.]: Grupo GEN, 2020. ISBN 9788595157552. 39

THOMAZINI, D.; ALBUQUERQUE, P. U. B. D. *Sensores industriais.* [S.l.]: Saraiva, 2020. 30

USINAINFO. *Mini Solenóide 12V NF Tipo Tranca JF-0520B.* 2023. (Acesso em: 15 set. 2023). Disponível em: https://www.usinainfo.com.br/mini-fechadura-eletrica-solenoide/mini-solenoide-12v-nf-tipo-tranca-jf-0520b-5246.html. 29

Apêndices

APÊNDICE A – CÓDIGO-FONTE DO FIRMWARE

Código A.1 – main.c

```c
#include <stdio.h>
#include "esp_log.h"
#include "srv_communication.h"

static const char TAG[] = "main";

void app_main(void)
{
    ESP_LOGI(TAG, "HELLO WORLD");
    srv_comm_init();
    // srv_comm_config_wifi();
}
```

Código A.2 – srvcommunication.c

```c
#include "srv_communication.h"

#include "srv_mqtt.h"
#include "srv_wifi.h"
#include "srv_rfid.h"
#include "drv_nvs.h"
#include "esp_log.h"
#include "drv_nvs.h"
#include "srv_button.h"
#include <time.h>
#include <stdio.h>
#include "esp_sntp.h"
#include "esp_netif_sntp.h"
#include "srv_hmi.h"

static const char *TAG = "SRV_COMM";

static TaskHandle_t door_handle=NULL;
hmi_cb led_cb = handler_hmi;
rc522_tag_t *tag = {0};
uint8_t break_in = 0;
SemaphoreHandle_t reed_sem = NULL;
uint16_t state=0;
```

```
24 bool rfid_requested_open = false;
25 static TimerHandle_t rfid_timer;
26
27 static void handler_on_sta_got_ip(void *arg, const char* event_base, int32_t
       event_id, void *event_data);
28 static void mqtt_event_handler(void *handler_args, const char* base, int32_t
       event_id, void *event_data);
29 static void rfid_handler(void* arg, esp_event_base_t base, int32_t event_id, void*
       event_data);
30 static void open_door(void *arg);
31 static uint8_t check_num_from_gpio(uint8_t num);
32 static uint8_t check_num_from_db(uint8_t num);
33 static uint8_t check_sensor_from_num(uint8_t num);
34 static uint8_t check_compartment_broken_from_gpio(uint8_t num);
35 static void split_data(const char* data, int* num, char* locker_name, char *
       tag_rfid) ;
36 static void IRAM_ATTR button_isr_handler(void* arg) ;
37 static void invasion_compartment(void *arg);
38 static void rfid_request_open(bool open);
39 static void rfid_timer_callback(TimerHandle_t xTimer);
40
41 void srv_comm_init(void)
42 {
43     drv_nvs_init();
44     srv_button_init();
45     srv_hmi_init();
46     char *ssid=malloc(32);
47     char *password=malloc(32);
48     size_t len_ssid;
49     size_t len_pass;
50
51     drv_nvs_err_et err = drv_nvs_get_str("storage", WIFI_SSID_KEY, ssid, &len_ssid)
       ;
52     if(err != DRV_NVS_OK) return;
53     err = drv_nvs_get_str("storage", WIFI_PASS_KEY, password, &len_pass);
54     if(err != DRV_NVS_OK) return;
55
56     wifi_config_st config =
57     {
58     .ssid = ssid,
59     .password = password,
60     .mode = APP_WIFI_MODE,
61     .interface = APP_WIFI_INTERFACE,
62     .interface_description = APP_WIFI_IF_DESC
63     };
64
65     rfid_config_st rfid_config =
```

```
66      {
67          .host = VSPI_HOST,
68          .miso = 19,
69          .mosi = 23,
70          .sck = 18,
71          .sda = 21
72      };
73
74      if(srv_wifi_start(&config) != DRV_WIFI_OK) return;
75      srv_wifi_connect();
76      srv_wifi_set_callback(handler_on_sta_got_ip, APP_EVENT_IP_STA);
77      srv_rfid_start(rfid_config);
78      srv_rfid_set_callback(rfid_handler);
79      free(ssid);
80      free(password);
81
82      drv_gpio_set_direction(APP_COMPARTMENT_1, GPIO_MODE_OUTPUT);
83      drv_gpio_set_direction(APP_COMPARTMENT_2, GPIO_MODE_OUTPUT);
84
85      drv_gpio_set_direction(APP_SENSOR_1, GPIO_MODE_INPUT);
86      // drv_gpio_set_pull(39);
87      drv_gpio_set_direction(APP_SENSOR_2, GPIO_MODE_INPUT);
88
89      drv_gpio_set_intr_type(APP_SENSOR_1, GPIO_INTR_NEGEDGE);
90      drv_gpio_set_intr_type(APP_SENSOR_2, GPIO_INTR_NEGEDGE);
91      gpio_isr_handler_add(APP_SENSOR_1, button_isr_handler, (void*) APP_SENSOR_1);
92      gpio_isr_handler_add(APP_SENSOR_2, button_isr_handler, (void*) APP_SENSOR_2);
93
94      led_cb(SRV_LED_FREE_TO_USE,0);
95      led_cb(SRV_LED_NO_FREE_DOORS,1);
96      if(reed_sem == NULL) reed_sem = xSemaphoreCreateBinary();
97      xSemaphoreGive(reed_sem);
98      return;
99  }
100
101 static bool debounce(uint8_t btn)
102 {
103     state = (state<<1) | !drv_gpio_get_level(btn) | 0xfe00;
104     return (state == 0xfe01);
105 }
106
107 static bool task_state(void)
108 {
109     eTaskState state_task = door_handle != NULL ? eTaskGetState(door_handle):
        eInvalid;
110     return (state_task!=eBlocked && state_task!= eRunning);
111 }
```

```
112
113 static void IRAM_ATTR button_isr_handler(void* arg)
114 {
115     uint8_t num = arg;
116     if(!rfid_requested_open && !drv_gpio_get_level(num)  && break_in != num &&
    task_state() && debounce(num))
117     {
118         state=0;
119         break_in = num;
120         ESP_DRAM_LOGI("","Interrupção detectada no botão: %u", num);
121         if(xSemaphoreTakeFromISR(reed_sem, NULL))
122             xTaskCreate(invasion_compartment, "invasion_compartment", 1024*3, NULL,
     10, NULL);
123     }
124 }
125
126 static void invasion_compartment(void *arg)
127 {
128     for(;;)
129     {
130         char str[25];
131         char locker_name[20] = "";
132         size_t locker_name_len = sizeof(locker_name);
133
134         drv_nvs_get_str("storage", LOCKER_NAME_KEY, locker_name, &locker_name_len);
135
136         sprintf(str, "%u:%s", check_num_from_gpio(break_in), locker_name);
137         srv_mqtt_publish("/weird_activity", str, strlen(str));
138         xSemaphoreGive(reed_sem);
139         vTaskDelete(NULL);
140     }
141 }
142
143 static void handler_on_sta_got_ip(void *arg, const char* event_base, int32_t
    event_id, void *event_data)
144 {
145     ip_event_got_ip_t *event = (ip_event_got_ip_t *)event_data;
146     ESP_LOGI(TAG, "Got IPv4 event: Interface \"%s\" address: " IPSTR,
    esp_netif_get_desc(event->esp_netif), IP2STR(&event->ip_info.ip));
147     ESP_LOGI(TAG, "- IPv4 address: " IPSTR ",", IP2STR(&event->ip_info.ip));
148     srv_mqtt_start("mqtt://192.168.1.7:1883");
149     srv_mqtt_set_callback(mqtt_event_handler);
150     esp_sntp_config_t config = ESP_NETIF_SNTP_DEFAULT_CONFIG("pool.ntp.org");
151     esp_netif_sntp_init(&config);
152 }
153
154 static void mqtt_event_handler(void *handler_args, const char* base, int32_t
```

```
    event_id, void *event_data)
{
    esp_mqtt_event_handle_t event = event_data;
    //esp_mqtt_client_handle_t client = event->client;
    //int msg_id;
    switch (event->event_id) {
        case MQTT_EVENT_CONNECTED:
            ESP_LOGI(TAG, "MQTT_EVENT_CONNECTED");
            int response = srv_mqtt_subscribe("/door_command/response", 2);
            //liga led "ja da pra usar"
            led_cb(SRV_LED_FREE_TO_USE,1);
            led_cb(SRV_LED_NO_FREE_DOORS,0);

            break;
        case MQTT_EVENT_DISCONNECTED:
            ESP_LOGI(TAG, "MQTT_EVENT_DISCONNECTED");
            //liga led "nao da pra usar!!!"
            led_cb(SRV_LED_FREE_TO_USE,0);
            led_cb(SRV_LED_NO_FREE_DOORS,1);

            break;
        case MQTT_EVENT_SUBSCRIBED:
            ESP_LOGI(TAG, "MQTT_EVENT_SUBSCRIBED, msg_id=%d", event->msg_id);
            break;
        case MQTT_EVENT_UNSUBSCRIBED:
            ESP_LOGI(TAG, "MQTT_EVENT_UNSUBSCRIBED, msg_id=%d", event->msg_id);
            break;
        case MQTT_EVENT_PUBLISHED:
            ESP_LOGI(TAG, "MQTT_EVENT_PUBLISHED, msg_id=%d", event->msg_id);
            break;
        case MQTT_EVENT_DATA:
            ESP_LOGI(TAG, "MQTT_EVENT_DATA");
            printf("TOPIC=%.*s\r\n", event->topic_len, event->topic);
            printf("DATA=%.*s\r\n", event->data_len, event->data);
            if(strcmp(event->topic,"/door_command/response")==0)
            {
                int compartment = 0;
                char locker_name_received[20];
                char tag_received[20]="";
                char locker_name[20] = "";
                size_t locker_name_len = sizeof(locker_name);

                drv_nvs_get_str("storage", LOCKER_NAME_KEY, locker_name, &
    locker_name_len);

                split_data(event->data, &compartment, locker_name_received, &
    tag_received);
```

```
199                 ESP_LOGI(TAG, "NUM: %d", compartment);
200 
201                 if(compartment == 0)
202                 {
203                     // led_cb(SRV_LED_STORE_OBJ,0);
204                     led_cb(SRV_LED_TAG_NOT_FOUND_IN_DB,1);
205                     vTaskDelay(pdMS_TO_TICKS(2000));
206                     led_cb(SRV_LED_TAG_NOT_FOUND_IN_DB,0);
207                     led_cb(SRV_LED_FREE_TO_USE,1);
208                     led_cb(SRV_LED_WAITING_ANSWER,0);
209                     return;
210                 }
211                 else if(compartment == -1)
212                 {
213                     led_cb(SRV_LED_FREE_TO_USE,0);
214                     led_cb(SRV_LED_WAITING_ANSWER,0);
215                     led_cb(SRV_LED_NO_FREE_DOORS,1);
216                     led_cb(SRV_LED_STORE_OBJ,0);
217 
218 
219                     return;
220                 }
221                 char str[20];
222                 sprintf(str, "%llu", tag->serial_number);
223                 tag_received[strlen(str)] = '\0';
224                 if((strcmp(tag_received,str)!=0)||(strcmp(locker_name_received,
    locker_name)!=0)) return;
225                 compartment = check_num_from_db(compartment);
226                 eTaskState state_task = door_handle != NULL ? eTaskGetState(
    door_handle): eInvalid;
227                 rfid_request_open(true);
228                 if(state_task!=eRunning) xTaskCreate(open_door, "open_door",
    1024*5, (void *)compartment, 10, &door_handle);
229 
230             }
231             break;
232         case MQTT_EVENT_ERROR:
233             ESP_LOGI(TAG, "MQTT_EVENT_ERROR");
234             break;
235         default:
236             ESP_LOGI(TAG, "Other event id:%d", event->event_id);
237             break;
238     }
239 }
240 
241 static void rfid_handler(void* arg, esp_event_base_t base, int32_t event_id, void*
    event_data)
```

```
242 {
243     rc522_event_data_t* data = (rc522_event_data_t*) event_data;
244 
245     switch(event_id) {
246         case RC522_EVENT_TAG_SCANNED: {
247             //liga led "tao usando, pera ai"
248                 tag = (rc522_tag_t*) data->ptr;
249                 ESP_LOGI(TAG, "Tag scanned (sn: %" PRIu64 ")", tag->serial_number);
250                 //manda pro mqtt
251                 char str[20];
252                 uint8_t num_compartments = 0;
253                 char locker_name[10] = "";
254                 size_t locker_name_len = sizeof(locker_name);
255                 drv_nvs_get_str("storage", LOCKER_NAME_KEY, locker_name, &
    locker_name_len);
256                 drv_nvs_get_u8("storage", LOCKER_NUM_KEY, &num_compartments);
257 
258                 snprintf(str, sizeof(str), "%llu:%s:%u",(unsigned long long)tag->
    serial_number, locker_name, num_compartments);
259                 int msg_id = srv_mqtt_publish("/door_command/request", str, strlen(
    str));
260                 led_cb(SRV_LED_FREE_TO_USE,0);
261                 led_cb(SRV_LED_NO_FREE_DOORS,0);
262                 led_cb(SRV_LED_WAITING_ANSWER,1);
263                 ESP_LOGI(TAG, "sent subscribe successful, msg_id=%d; value: %s",
    msg_id, str);
264                 vTaskDelay(pdMS_TO_TICKS(1000));
265             }
266             break;
267     }
268 }
269 
270 static uint8_t check_num_from_db(uint8_t num)
271 {
272     if(num == 1)
273         return APP_COMPARTMENT_1;
274     if(num == 2)
275         return APP_COMPARTMENT_2;
276     return 0;
277 }
278 
279 static uint8_t check_sensor_from_num(uint8_t num)
280 {
281     if(num == APP_COMPARTMENT_1)
282         return APP_SENSOR_1;
283     if(num == APP_COMPARTMENT_2)
284         return APP_SENSOR_2;
```

```
285     return 0;
286 }
287 
288 static uint8_t check_num_from_gpio(uint8_t num)
289 {
290     if(num == APP_COMPARTMENT_1 || num == APP_SENSOR_1)
291         return 1;
292     if(num == APP_COMPARTMENT_2 || num == APP_SENSOR_2)
293         return 2;
294     return 0;
295 }
296 
297 static uint8_t check_compartment_broken_from_gpio(uint8_t num)
298 {
299     if(num == APP_SENSOR_1)
300         return 1;
301     if(num == APP_SENSOR_2)
302         return 2;
303     return 0;
304 }
305 
306 static void open_door(void *arg)
307 {
308     uint8_t compartment = arg;
309     uint8_t sensor = check_sensor_from_num(compartment);
310     gpio_intr_disable(APP_SENSOR_2);
311     gpio_intr_disable(APP_SENSOR_1);
312     drv_gpio_set_direction(sensor, GPIO_MODE_INPUT);
313     state_door state = STATE_WAIT_DOOR_OPEN;
314     time_t timestamp = 0;
315     time_t timestamp2 = 0;
316     //verifica de qual gpio esse compartimento esta associado 15
317     //destrava tal gpio na condição imã fechado
318     drv_gpio_set_level(compartment, 1);
319     led_cb(SRV_LED_TAG_NOT_FOUND_IN_DB,0);
320 
321     for(;;)
322     {
323         if(state == STATE_WAIT_DOOR_OPEN)
324         {
325             if(drv_gpio_get_level(sensor)==0)
326             {
327                 ESP_LOGI(TAG, "> afastou o imã");
328 
329                 time(&timestamp);
330 
331                 state = STATE_WAIT_DOOR_CLOSE;
```

```
332             }
333             vTaskDelay(pdMS_TO_TICKS(100));
334 
335             //quando detectar que o ima afastou, começa a contar o tempo
336 
337         }
338         else if(state == STATE_WAIT_DOOR_CLOSE)
339         {
340             if(drv_gpio_get_level(sensor)==1)
341             {
342                 time(&timestamp2);
343 
344                 drv_gpio_set_level(compartment, 0);
345                 char locker_name[10] = "";
346                 size_t locker_name_len = sizeof(locker_name);
347                 drv_nvs_get_str("storage", LOCKER_NAME_KEY, locker_name, &
    locker_name_len);
348 
349                 char str[50];
350                 snprintf(str, sizeof(str), "%lld:%lld:%llu:%u:%s",timestamp,
    timestamp2, (unsigned long long)tag->serial_number, check_num_from_gpio(
    compartment), locker_name);
351 
352                 int msg_id = srv_mqtt_publish("/door/info", str, strlen(str));
353                 ESP_LOGI(TAG, "sent subscribe successful, msg_id=%d; value: %s",
    msg_id, str);
354 
355                 tag = NULL;
356                 //liga led "liberou pra usar"
357                 led_cb(SRV_LED_WAITING_ANSWER,0);
358                 led_cb(SRV_LED_NO_FREE_DOORS,0);
359                 led_cb(SRV_LED_FREE_TO_USE,1);
360                 led_cb(SRV_LED_REMOVE_OBJ,0);
361                 led_cb(SRV_LED_STORE_OBJ,0);
362                 gpio_intr_enable(APP_SENSOR_1);
363                 gpio_intr_enable(APP_SENSOR_2);
364                 rfid_timer = xTimerCreate("RFID_Timer", pdMS_TO_TICKS(2000),
    pdFALSE, (void*)0, rfid_timer_callback);
365                 xTimerStart(rfid_timer, 0);
366                 vTaskDelete(NULL);
367             }
368             //quando detectar que o ima afastou, começa a contar o tempo
369 
370         }
371         vTaskDelay(pdMS_TO_TICKS(100));
372 
373         //quando detectar que o ima aproximou, para de contar o tempo e trava
```

```
374             //envia via mqtt
375
376     }
377     gpio_intr_enable(sensor);
378     vTaskDelete(NULL);
379 }
380
381 static void split_data(const char* data, int* num, char* locker_name, char *
        tag_rfid)
382 {
383     // Use a cópia da string para preservá-la, pois strtok modifica a string
        original.
384     char data_copy[50];
385     strncpy(data_copy, data, sizeof(data_copy));
386
387     // Use strtok para dividir a string
388     char* token = strtok(data_copy, ":");
389
390     if (token != NULL)
391     {
392         // Primeiro valor: num (uint8_t)
393         *num = (int)atoi(token);
394
395         // Segundo valor: locker_name (string)
396         token = strtok(NULL, ":");
397         if (token != NULL)
398         {
399             strcpy(locker_name, token);
400
401             // Terceiro valor: tag_rfid (string)
402             token = strtok(NULL, ":");
403             if (token != NULL)
404             {
405                 strcpy(tag_rfid, token);
406             }
407             token = strtok(NULL, ":");
408             if (token != NULL)
409             {
410                 char led[6];
411                 strcpy(led, token);
412                 // ESP_LOGI(TAG, "LED: %d - %d", (int)strcmp(led, "store"), (int)
        strcmp(led, "remove"));
413
414                 if(strcmp(led, "store")>=0)
415                 {
416                     ESP_LOGI(TAG, "STORE");
417                     led_cb(SRV_LED_STORE_OBJ, 1);
```

```
418                 }
419                 else if(strcmp(led, "remove")>=0)
420                 {
421                     ESP_LOGI(TAG, "REMOVE");
422 
423                     led_cb(SRV_LED_REMOVE_OBJ, 1);
424                 }
425             }
426         }
427     }
428 }
429 
430 static void rfid_request_open(bool open)
431 {
432     rfid_requested_open = open;
433 }
434 
435 static void rfid_timer_callback(TimerHandle_t xTimer)
436 {
437     rfid_requested_open = false;
438 }
```

APÊNDICE B – CÓDIGO-FONTE DO SERVIDOR

Código B.1 – flaskmqtt.py

```python
from flask_mqtt import Mqtt
import time
from .queries import *
import random
from flask import current_app
from flask_mail import Mail, Message

mqtt = Mqtt()
topic_request = "/door_command/request"
topic_response = "/door_command/response"
topic_send_info = "/door/info"
topic_notify_error = "/weird_activity"
global_uuid = 0

def init_mqtt(app):
    mqtt.init_app(app)
    mail = Mail(app)

    @mqtt.on_connect()
    def handle_connect(client, userdata, flags, rc):
        if rc == 0:
            print('Conectado ao broker MQTT com sucesso')
            client.subscribe(topic_request, qos=2)
            client.subscribe(topic_send_info, qos=2)
            client.subscribe(topic_notify_error, qos=2)
        else:
            print(f'Falha ao conectar ao broker MQTT, código: {rc}')

    @mqtt.on_message()
    def handle_mqtt_message(client, userdata, msg):
        with app.app_context():
            global global_uuid
            if msg.retain:
                return
            time.sleep(1)

            message = msg.payload.decode('utf-8', errors='ignore')
            print(f"message: {message}")
```

```
39            if msg.topic == topic_notify_error:
40                parts = msg.payload.decode('utf-8', errors='ignore').split(':')
41                if len(parts) == 2:
42                    compartment = parts[0]
43                    locker_name = parts[1]
44                print(f"Possível arrombamento no compartimento {message}")
45                users_admins = get_admins()
46
47                msg = Message(
48                    'Security alert',  sender='smart.lockers.notification@gmail.com
    ',
49                    recipients = users_admins
50                )
51                msg.body = f"Possible invasion in compartment {compartment}. \
    nLocker name: {locker_name}"
52
53                with app.app_context():
54                    mail.send(msg)
55
56                    return "E-mail enviado com sucesso!"
57
58            if msg.topic == topic_request:
59                parts = msg.payload.decode('utf-8', errors='ignore').split(':')
60                if len(parts) == 3:
61                    tag = parts[0]
62                    locker_name = parts[1]
63                    num_compartments = parts[2]
64
65                    print(f"Mensagem recebida: {msg.topic} -> {msg.payload.decode('
    utf-8', errors='ignore')}")
66                    try:
67                        num_compartments = int(num_compartments)
68                    except ValueError:
69                        print("Erro: número de compartimentos inválido")
70                        return
71                    locker = get_locker(locker_name)
72                    if locker is None:
73                        set_locker(locker_name)
74
75                    locker_obj = get_locker(locker_name)
76
77                    for i in range(1, num_compartments+1):
78                        if check_compartment_by_num(i, locker_obj.id) is None:
79                            set_compartment(locker_obj.id, i)
80
81                    user = get_user_by_tag(tag)
82                    if(user is None):
```

```
83                          print(f"Tag não cadastrada: {tag}")
84                          buffer = ':'.join(['0' , locker_name, tag])
85                          client.publish(topic_response, payload=buffer, qos=2)
86
87                          return
88                      compartment_usage = get_compartment_usage(user)
89                      if compartment_usage is not None:
90                          print(f"user: {user.name} is using a compartment already,
    user is removing objects")
91                          compartment = get_compartment_by_id(compartment_usage.
    compartment_id)
92                          buffer = ':'.join([str(compartment.number) , locker_name,
    tag,'remove'])
93                          client.publish(topic_response, payload=buffer, qos=2)
94                      else:
95                          print(f"user: {user.name} isn't using a compartment, user
    is keeping objects")
96
97                          available_numbers = get_available_compartments(get_locker(
    locker_name).id)
98
99                          if available_numbers:
100                             chosen_number = random.choice(available_numbers)
101                             print(f"Chosen number: {chosen_number}")
102                             buffer = ':'.join([str(chosen_number) , locker_name,
    tag, 'store'])
103                             client.publish(topic_response, payload=buffer, qos=2)
104                         else:
105                             buffer = ':'.join([str(-1) , locker_name, tag])
106                             client.publish(topic_response, payload=buffer, qos=2)
107                             print("No available compartments")
108
109                 else:
110                     return
111             elif msg.topic == topic_send_info:
112
113                 parts = msg.payload.decode('utf-8', errors='ignore').split(':')
114                 if len(parts) == 5:
115                     time1 = parts[0]
116                     time2 = parts[1]
117                     tag = parts[2]
118                     num_compartment = parts[3]
119                     locker_name = parts[4]
120
121                     try:
122                         num_compartment = int(num_compartment)
123                     except ValueError:
```

```
                        print("Erro: número do compartimento inválido")
                        return
                    locker = get_locker(locker_name)
                    if locker is None:
                        print("Erro: Nome do locker inválido")
                        return
                    user = get_user_by_tag(tag)
                    if(user is None):
                        print(f"Tag não cadastrada: {tag}")
                        return
                    compartment_usage = get_compartment_usage(user)
                    if compartment_usage is None:
                        set_compartment_usage(time1, time2, user.id, locker.id,
    num_compartment)
                    else:
                        set_locker_schedule(time1, time2, user.id, locker.id,
    num_compartment)

                    print(f"Mensagem recebida: {msg.topic} -> {msg.payload.decode('
    utf-8', errors='ignore')}")

    @mqtt.on_disconnect()
    def handle_disconnect(client, userdata, rc):
        print('Desconectado do broker MQTT')

    @mqtt.on_subscribe()
    def handle_subscribe(client, userdata, mid, granted_qos):
        print('Assinatura MQTT realizada')
```

Código B.2 – routes.py

```
import flask, logging, random, string, pytz
from flask import render_template, session, jsonify, redirect, url_for, request
from flask_login import current_user, login_required, login_user, logout_user
from flask_nav.elements import Navbar, View, Text
from sqlalchemy import select
from datetime import datetime, timedelta
from .queries import *
from flask_mail import Mail, Message

from . import nav,db
from .forms import *

def init_routes(app):

    @app.route('/edit_number', methods=['GET', 'POST'])
    def edit_number():
        """
```

```
18          Associa algum compartimento já em uso ao admin logado
19          """
20          if session.get('logado'):
21              user_id = session['usuario']
22              usuario = get_user_by_id(user_id)
23              if(usuario.user_type=='admin'):
24                  compartment = get_compartment_by_compartment_in_use_user(user_id)
25
26                  form = CompartmentAdmin()
27
28                  if form.validate_on_submit():
29                      novo_valor = form.compartment.data
30
31                  if compartment:
32                      form.compartment.data = compartment.number
33                  return render_template('locker_schedule.html', form=form)
34
35      @nav.navigation()
36      def menu():
37          """
38          Cria o menu de navegação para o Smart Locks Server.
39
40          Esta função define a barra de navegação usando a biblioteca Flask-Nav.
41          O menu muda dinamicamente com base no estado de login do usuário e seu
42          tipo (admin ou comum).
43
44          Se o usuário estiver logado:
45              - Exibe uma mensagem de saudação com o nome do usuário.
46              - Adiciona opções de menu específicas para administradores, como
47              'Registrar usuário' e 'Atualizar e-mail/senha do usuário'.
48              - Adiciona a opção 'Exit' para permitir que o usuário saia.
49
50          Se o usuário não estiver logado:
51              - Exibe apenas a opção de menu 'Início'.
52
53          Returns:
54              Navbar: O objeto Navbar configurado com os itens de menu apropriados.
55          """
56          menu = Navbar('Smart Locks Server')
57          if session.get('logado'):
58              user_id = session['usuario']
59              usuario_logado = get_user_by_id(user_id)
60              menu.items.append(Text('Hello, '+usuario_logado.name))
61              if(usuario_logado.user_type=='admin'):
62                  menu.items.append(View('Registrar usuário', 'register'))
63                  if(usuario_logado.email!='admin'):
64                      menu.items.append(View('Atualizar e-mail usuário', '
```

```
update_email'))
65              menu.items.append(View('Atualizar senha usuário', 'update_password'))
66              menu.items.append(View('Exit', 'sair'))
67              return menu
68          menu.items = [View('Início', 'inicial')]
69          return menu
70
71      @app.route('/', methods=['GET', 'POST'])
72      def inicial():
73          """
74          Renderiza a página de navegação para o Smart Locks Server.
75
76          Esta função define a página de navegação da rota principal usando a
    biblioteca Flask-session.
77          A página muda dinamicamente com base no estado de usuário logado ou não.
78
79          Se o usuário estiver logado:
80              - Renderiza a página 'index.html', exibindo informações perminentes ao
    usuário logado.
81              - O menu exibido muda dinamicante de acordo com o tipo de usuário.
82
83          Se o usuário não estiver logado:
84              - Renderiza a página 'login.html'.
85
86          Returns:
87              Redirect: O redirecionamento para a página.
88          """
89          if session.get('logado'):
90              form = CompartmentAdmin()
91              user_id = session['usuario']
92              usuario = get_user_by_id(user_id)
93              locker_schedules = None
94              if(usuario.user_type == 'admin'):
95                  compartment = get_compartment_by_compartiment_in_use_by_user(
    user_id)
96
97                  if compartment is not None:
98                      form.compartment.data = compartment.number
99                      form.locker_name.data = compartment.locker.name
100
101                 if request.method == 'POST':
102                     form = CompartmentAdmin()
103                     if form.validate_on_submit():
104                         locker = get_locker(form.locker_name.data)
105                         if locker is not None:
106                             compartment_to_choose = get_compartment_by_number(form.
    compartment.data, locker.id)
```

```
107                         if compartment is None and compartment_to_choose is not
    None:
108                             set_compartment_in_use_now(user_id,
    compartment_to_choose.id, datetime.now(), datetime.now())
109                             return redirect(url_for('inicial'))
110
111             locker_schedules = get_all_lockers()
112             compartment_in_use = get_all_compartiment_in_use()
113         else:
114             locker_schedules = get_all_lockers_schedules_by_id_user(user_id)
115             compartment_in_use = get_all_compartment_in_use_by_id_user(user_id)
116         return render_template('locker_schedule.html', locker_schedules=
    locker_schedules, compartment_in_use=compartment_in_use, type = usuario.
    user_type, form=form)
117     session['usuario'] = None
118     session['logado'] = False
119     return flask.redirect(flask.url_for('login'))
120
121 @app.route('/sair')
122 @login_required
123 def sair():
124     logout_user()
125     session['logado'] = False
126     session['usuario'] = None
127     logging.info('Você foi desconectado com sucesso.', 'success')
128     return redirect(url_for('login'))
129
130 @app.route('/login', methods=['GET', 'POST'])
131 def login():
132     form = LoginForm()
133     if form.validate_on_submit():
134         usuario = get_user_by_email(form.email.data)
135         set_password_admin_user(usuario)
136
137         if usuario is not None:
138             if usuario.check_password(form.password.data):
139                 session['logado']=True
140                 session['usuario']=usuario.id
141                 print(f'Usuário logado com ID: {session["usuario"]}, nome: {
    usuario.name}')
142                 login_user(usuario)
143                 if(usuario.email=='admin'):
144                     return flask.redirect(flask.url_for('update'))
145                 return flask.redirect(flask.url_for('inicial'))
146             else:
147                 return render_template('login.html', form=form)
148     elif form.forgot_password.data:
```

```
149             return redirect(url_for('forgot_password'))
150         session['usuario'] = None
151         session['logado'] = False
152         return render_template('login.html', form=form)
153 
154     @app.route('/forgot_password', methods=['GET', 'POST'])
155     def forgot_password():
156         form = ForgotPasswordForm()
157         timezone = pytz.timezone("America/Sao_Paulo")
158         if form.validate_on_submit():
159             if form.submit.data:
160                 email = form.email.data
161                 date_time = datetime.now(timezone)
162                 code=random_string(6)
163                 set_code_validation_password(code=code, email=email, time_generated
    =date_time)
164                 mail = Mail(app)
165                 msg = Message(
166                     'Smart Lockers - Reset Password',  sender='smart.lockers.
    notification@gmail.com',
167                     recipients = str(email).split()
168                 )
169                 msg.body = f"Code to reset password {code}. \nValid for 15 minutes.
    "
170 
171                 mail.send(msg)
172 
173             elif form.submit_code.data:
174                 any_date = get_date_from_code_validation_password(form.code.data)
175 
176                 if any_date is not None and any_date.tzinfo is None:
177                     any_date = timezone.localize(any_date)
178 
179                 date_time = datetime.now(timezone)
180                 if date_time - any_date < timedelta(minutes=15):
181                     return redirect(url_for('reset_password', email=
    get_email_from_code_validation_password(form.code.data)))
182                 else:
183                     return redirect(url_for('login'))
184         return render_template('forgot_password.html', form=form)
185 
186     @app.route('/reset_password', methods=['GET','POST'])
187     def reset_password():
188         form = ResetPasswordForm()
189         email = request.args.get('email') or session.get('email')
190         if request.args.get('email') is not None:
191             session['email'] = request.args.get('email')
```

```
192         form.email.data = email
193         if form.validate_on_submit():
194             form.new_password.data
195             update_password_user(email, form.new_password.data)
196             session['email'] = None
197             return redirect(url_for('login'))
198         return render_template('reset_password.html', form=form, email=email)
199 
200     @app.route('/register', methods=['GET', 'POST'])
201     def register():
202         form = RegisterForm()
203         if request.method == 'POST':
204             logging.info('Form submitted')
205             if form.validate_on_submit():
206                 set_user(form.nome.data, form.email.data, form.senha.data, form.tag
    .data, form.tipo_usuario.data)
207                 logging.info('User registered successfully')
208                 return redirect(url_for('inicial'))
209             else:
210                 logging.error('Form validation failed')
211                 logging.error(form.errors)
212                 return render_template('register.html', form=form)
213         return render_template('register.html', form=form)
214 
215     @app.route('/check_email', methods=['POST'])
216     def check_email():
217         email = request.form.get('email')
218         user = get_user_by_email(email)
219         if user:
220             return jsonify({'exists': True})
221         return jsonify({'exists': False})
222 
223     @app.route('/update-email', methods=['GET', 'POST'])
224     @login_required
225     def update_email():
226         form = UpdateEmailForm()
227         if form.validate_on_submit():
228             update_email_user(form.old_email.data, form.email.data, form.senha.data
    )
229             print('Suas informações foram atualizadas com sucesso!')
230             return redirect(url_for('inicial'))
231         return render_template('update_email.html', form=form)
232 
233     @app.route('/update-password', methods=['GET', 'POST'])
234     @login_required
235     def update_password():
236         form = UpdatePasswordForm()
```

```
237         if form.validate_on_submit():
238             user_id = session['usuario']
239             usuario = get_user_by_id(user_id)
240             if usuario.email == form.email.data or usuario.user_type == "admin":
241                 update_password_user(form.email.data, form.confirmar_senha.data)
242 
243             print('Suas informações foram atualizadas com sucesso!')
244             return redirect(url_for('inicial'))
245         return render_template('update_password.html', form=form)
246 
247     @app.route('/update', methods=['GET', 'POST'])
248     @login_required
249     def update():
250         if current_user.email != 'admin':
251             print('Acesso negado: você não tem permissão para acessar esta página.'
    , 'danger')
252             return redirect(url_for('inicial'))
253         form = UpdateEmailPasswordForm()
254         if form.validate_on_submit():
255             update_admin_user(form.email.data, form.nome.data, form.tag.data, form.
    confirmar_senha.data)
256             print('Suas informações foram atualizadas com sucesso!')
257             return redirect(url_for('inicial'))
258         return render_template('update.html', form=form)
259 
260     def random_string(length):
261         characters = string.ascii_letters + string.digits
262         return ''.join(random.choice(characters) for _ in range(length))
```

Código B.3 – queries.py

```
1  from sqlalchemy import select, delete, update
2  from flask import current_app
3  from datetime import datetime
4  from . import db
5  from .models.user import User
6  from .models.compartment_usage import CompartmentUsage
7  from .models.compartment import Compartment
8  from .models.locker_schedules import LockerSchedules
9  from .models.locker import Lockers
10 from .models.forgot_password import ForgotPassword
11 
12 def timestamp_to_datetime(timestamp):
13     return datetime.fromtimestamp(int(timestamp))
14 
15 with current_app.app_context():
16     def get_user_by_tag(tag):
17         """Retorna um usuário pela tag."""
```

```
18          statement = select(User).filter_by(tag=tag)
19          user = db.session.execute(statement).scalars().first()
20          return user
21
22      def get_compartment_usage(user):
23          """Retorna um compartimento em uso pelo usuário."""
24          statement = select(CompartmentUsage).filter_by(user_id=user.id)
25          compartment_usage = db.session.execute(statement).scalars().first()
26          return compartment_usage
27
28      def set_compartment_usage(open_time, close_time, user_id, locker_id, num):
29          """Configura um compartimento em uso """
30          statement = select(Compartment).filter_by(locker_id=locker_id, number = num
    )
31          compartment = db.session.execute(statement).scalars().first()
32
33          compart_use = CompartmentUsage(
34              id_user = user_id,
35              id_compartment = compartment.id,
36              open_time= timestamp_to_datetime(open_time),
37              close_time = timestamp_to_datetime(close_time)
38          )
39
40          db.session.add(compart_use)
41          db.session.commit()
42
43      def set_locker_schedule(open_time, close_time, id_user, locker_id, num):
44          """Registra o uso de um compartimento"""
45          statement = select(Compartment).filter_by(locker_id=locker_id, number = num
    )
46          compartment = db.session.execute(statement).scalars().first()
47
48          statement = select(CompartmentUsage).filter_by(compartment_id=compartment.
    id)
49          compartment_usage_list = db.session.execute(statement).scalars().all()
50
51          for compartment_usage in compartment_usage_list:
52              statement = select(User).filter_by(id=compartment_usage.user_id)
53              user = db.session.execute(statement).scalars().first()
54
55              stmt = delete(CompartmentUsage).where(CompartmentUsage.user ==
    compartment_usage.user)
56              db.session.execute(stmt)
57              db.session.commit()
58
59              statement = select(Compartment).filter_by(locker_id=locker_id, number =
     num)
```

```
60              compartment = db.session.execute(statement).scalars().first()
61
62              locker_schedule = LockerSchedules(
63                  id_user = compartment_usage.user.id,
64                  id_compartment = compartment.id,
65                  open_time= compartment_usage.open_time,
66                  close_time = compartment_usage.close_time,
67                  retrieve_time = timestamp_to_datetime(open_time),
68                  end_retrieve_time = timestamp_to_datetime(close_time)
69              )
70
71              db.session.add(locker_schedule)
72              db.session.commit()
73
74      def get_locker(locker_name):
75          """Retorna o locker pelo nome"""
76          statement = select(Lockers).filter_by(name=locker_name)
77          locker = db.session.execute(statement).scalars().first()
78          return locker
79
80      def check_compartment_by_num(num, locker_id):
81          """Retorna o objeto compartimento a partir do nome e locker id"""
82          statement = select(Compartment).filter_by(number=num)
83          compartment = db.session.execute(statement).scalars().first()
84          if compartment is not None:
85              if compartment.locker_id == locker_id:
86                  return compartment
87          return None
88
89      def get_compartment_by_id(id):
90          """Retorna o objeto compartimento a partir do id"""
91          statement = select(Compartment).filter_by(id=id)
92          compartment = db.session.execute(statement).scalars().first()
93          return compartment
94
95      def set_locker(locker_name):
96          """Configura um compartimento a partir do nome do armário"""
97          locker = Lockers (
98              name= locker_name
99          )
100         db.session.add(locker)
101         db.session.commit()
102
103     def set_compartment(locker_id, num):
104         """Configura um compartimento a partir do número e locker id"""
105         statement = select(Compartment).filter_by(locker_id=locker_id, number=num)
106         existing_compartment = db.session.execute(statement).scalars().first()
```

```
107         if existing_compartment:
108             print("A combinação locker_id e number já existe.")
109             return
110         try:
111             compartment_obj = Compartment (
112                 locker_id = locker_id,
113                 number = num
114             )
115             db.session.add(compartment_obj)
116             db.session.commit()
117         except ValueError:
118             return
119 
120     def get_admins():
121         """Retorna uma lista com todos os e-mails dos administradores do sistema"""
122         users_admins = db.session.query(User.email).filter(User.user_type == "admin
    ").all()
123 
124         return [email[0] for email in users_admins]
125 
126     def get_available_compartments(locker_id):
127         """Retorna compartimentos disponiveis pelo locker id"""
128         used_compartments_subquery = (
129             db.session.query(CompartmentUsage.compartment_id)
130             .join(Compartment, Compartment.id == CompartmentUsage.compartment_id)
131             .filter(Compartment.locker_id == locker_id)
132             .subquery()
133         )
134 
135         available_compartments = (
136             db.session.query(Compartment.number)
137             .filter(
138                 Compartment.locker_id == locker_id,
139                 Compartment.id.notin_(used_compartments_subquery)
140             )
141             .all()
142         )
143 
144         available_compartments_numbers = [compartment.number for compartment in
    available_compartments]
145 
146         return available_compartments_numbers
147 
148     def set_code_validation_password(email, code, time_generated):
149         """Configura o código de validação para o 'Esqueceu sua senha'; com 15
    minutos de validade"""
150         statement = select(User).filter_by(email=email)
```

```
151         user = db.session.execute(statement).scalars().first()
152         if user is not None:
153             statement = select(ForgotPassword).filter_by(user=user)
154             forgot_password = db.session.execute(statement).scalars().first()
155             if forgot_password is None:
156                 password = ForgotPassword (
157                     code_generated= code,
158                     id_user=user.id,
159                     date_time = time_generated
160                 )
161                 db.session.add(password)
162                 db.session.commit()
163             else:
164                 stmt = (
165                     update(ForgotPassword).
166                     where(ForgotPassword.user_id == user.id).
167                     values(code=code, date_time=time_generated)
168                 )
169                 db.session.execute(stmt)
170                 db.session.commit()
171 
172     def get_date_from_code_validation_password(code):
173         """Retorna a data gerada do código para validação no 'esqueceu sua senha'""
    "
174         statement = select(ForgotPassword).filter_by(code=code)
175         forgot_password = db.session.execute(statement).scalars().first()
176 
177         if forgot_password is not None:
178             return forgot_password.date_time
179         return None
180 
181     def get_email_from_code_validation_password(code):
182         """Retorna o e-mail a partir do código para validação no 'esqueceu sua
    senha'"""
183         statement = select(ForgotPassword).filter_by(code=code)
184         forgot_password = db.session.execute(statement).scalars().first()
185 
186         if forgot_password is not None:
187             return forgot_password.user.email
188         return None
189 
190     def update_password_user(email, new_password):
191         "Atualiza a senha do usuário a partir do e-mail"
192         statement = select(User).filter_by(email=email)
193         user = db.session.execute(statement).scalars().first()
194         if user is not None:
195             user.password = user.set_password(new_password)
```

```
196             db.session.commit()
197
198     def get_compartment_by_compartment_in_use_user(user_id):
199         """Retorna o objeto compartimento a partir do id do usuario caso esteja
    associado a algum compartimento em uso"""
200         statement = select(CompartmentUsage).filter_by(user_id=user_id)
201         compartment_in_use = db.session.execute(statement).scalars().first()
202         statement = select(Compartment).filter_by(id=compartment_in_use.
    compartment_id)
203         compartment = db.session.execute(statement).scalars().first()
204         return compartment
205
206     def get_user_by_id(id_user):
207         """Retorna o objeto usuário a partir do seu id"""
208         statement = select(User).filter_by(id=id_user)
209         usuario = db.session.execute(statement).scalars().first()
210         return usuario
211
212     def get_user_by_email(email):
213         statement = select(User).filter_by(email=email)
214         usuario = db.session.execute(statement).scalars().first()
215         return usuario
216
217     def set_user(nome, email, senha, tag, user_type):
218         usuario = User(
219             name= nome,
220             email=email,
221             password=senha,
222             tag=tag,
223             user_type=user_type
224         )
225         db.session.add(usuario)
226         db.session.commit()
227
228     def set_password_admin_user(usuario):
229         try:
230             if(usuario.password == 'admin'):
231                 usuario.set_password('admin')
232                 db.session.commit()
233         except:
234             pass
235
236     def update_email_user(old_email, new_email, password):
237         statement = select(User).filter_by(email=old_email)
238         user = db.session.execute(statement).scalars().first()
239         user.email = new_email
240         if(user.check_password(password)):
```

```
241             db.session.commit()
242
243     def update_admin_user(email, name, tag, password):
244         statement = select(User).filter_by(email='admin')
245         user = db.session.execute(statement).scalars().first()
246         user.email = email
247         user.name = name
248         user.tag = tag
249         user.password = password
250         db.session.commit()
251
252     def get_compartment_by_compartiment_in_use_by_user(user_id):
253         statement = select(CompartmentUsage).filter_by(user_id=user_id)
254         compartment_in_use = db.session.execute(statement).scalars().first()
255         if compartment_in_use is not None:
256             statement = select(Compartment).filter_by(id=compartment_in_use.
    compartment_id)
257             compartment = db.session.execute(statement).scalars().first()
258             return compartment
259         return None
260
261     def get_compartment_by_number(num, locker_id):
262         statement = select(Compartment).filter_by(number=num, locker_id=locker_id)
263         compartment = db.session.execute(statement).scalars().first()
264         return compartment
265
266     def get_compartment_in_use_by_id_comp(id_compartment):
267         statement = select(CompartmentUsage).filter_by(compartment_id=
    id_compartment)
268         compartment_in_use = db.session.execute(statement).scalars().first()
269         return compartment_in_use
270
271     def set_compartment_in_use_now(user_id, compartment_id, open_time, close_time):
272         new_compartment_in_use = CompartmentUsage(
273             id_user=user_id,
274             open_time=open_time,
275             close_time=close_time,
276             id_compartment=compartment_id
277         )
278         db.session.add(new_compartment_in_use)
279         db.session.commit()
280
281     def get_all_lockers():
282         return LockerSchedules.query.all()
283
284     def get_all_compartiment_in_use():
285         return CompartmentUsage.query.all()
```

```
286
287     def get_all_lockers_schedules_by_id_user(user_id):
288         statement = select(LockerSchedules).filter_by(user_id=user_id)
289         locker_schedules = db.session.execute(statement).scalars().all()
290         return locker_schedules
291
292     def get_all_compartment_in_use_by_id_user(user_id):
293         statement = select(CompartmentUsage).filter_by(user_id=user_id)
294         compartment_in_use = db.session.execute(statement).scalars().all()
295         return compartment_in_use
```

APÊNDICE C – SCRIPT SQL DO SERVIDOR

Código C.1 – ddl-dml.sql

```sql
START TRANSACTION;
DROP TABLE IF EXISTS user;
CREATE TABLE IF NOT EXISTS user (
    id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(100) NOT NULL,
    email VARCHAR(40) NOT NULL UNIQUE,
    password VARCHAR(120) NOT NULL,
    tag VARCHAR(20) NOT NULL,
    user_type VARCHAR(10) NOT NULL
);
INSERT INTO user VALUES (1,'','admin','admin','admin');


CREATE TABLE locker (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);

CREATE TABLE compartment (
    id INT AUTO_INCREMENT PRIMARY KEY,
    locker_id INT NOT NULL,
    number INT NOT NULL,
    FOREIGN KEY (locker_id) REFERENCES locker(id)
    UNIQUE (locker_id, number)
);

CREATE TABLE compartment_usage (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    compartment_id INT NOT NULL,
    open_time DATETIME NOT NULL,
    close_time DATETIME NOT NULL,
    FOREIGN KEY (user_id) REFERENCES user(id_user),
    FOREIGN KEY (compartment_id) REFERENCES compartment(id),
    UNIQUE (user_id)  -- Garante que um usuário só possa usar um compartimento por
    vez.
);

```

```
38 CREATE TABLE locker_schedule (
39     id INT AUTO_INCREMENT PRIMARY KEY,
40     compartment_id INT NOT NULL,
41   user_id INT NOT NULL,
42     open_time DATETIME NOT NULL,
43     close_time DATETIME NOT NULL,
44     retrieve_time DATETIME NOT NULL,
45     end_retrieve_time DATETIME NOT NULL,
46     FOREIGN KEY (compartment_id) REFERENCES compartment(id),
47   FOREIGN KEY (user_id) REFERENCES user(id_user)
48 );
49
50 CREATE TABLE forgot_password (
51     id INT AUTO_INCREMENT PRIMARY KEY,
52     code VARCHAR(10) NOT NULL,
53     date_time DATETIME NOT NULL,
54   user_id INT NOT NULL,
55   FOREIGN KEY (user_id) REFERENCES user(id),
56     UNIQUE (user_id)
57 );
58
59 COMMIT;
60
61 drop table compartment_usage;
62 drop table locker_schedule;
63 drop table compartment;
64 drop table locker;
```