CENTRO UNIVERSITÁRIO UNIDADE DE ENSINO SUPERIOR DOM BOSCO
CURSO ENGENHARIA DE SOFTWARE

CAIQUE DOURADO

SISTEMA DE CONTROLE DE ACESSO EM CONDOMÍNIOS UTILIZANDO TECNOLOGIA IOT: uma abordagem de engenharia de software para eficiência e segurança.

São Luís
2023

CAIQUE DOURADO

SISTEMA DE CONTROLE DE ACESSO EM CONDOMÍNIOS UTILIZANDO TECNOLOGIA IOT: uma abordagem de engenharia de software para eficiência e segurança.

Monografia apresentada ao Curso de Engenharia de Software do Centro Universitário Unidade de Ensino Superior Dom Bosco como requisito parcial para obtenção do grau de Bacharel em Engenharia de Software.
Orientador: Prof. Me. Allisson Jorge Silva Almeida.

São Luís

2023

Dados Internacionais de Catalogação na Publicação (CIP)

Centro Universitário - UNDB / Biblioteca

Dourado, Caique

Sistema de controle de acesso em condomínios utilizando Tecnologia Iot: uma abordagem de engenharia de software para eficiência e segurança. / Caique Dourado. __ São Luís, 2023.

84 f.

Orientador: Prof. Me. Alisson Jorge Silva Almeida.

Monografia (Graduação em Engenharia de Software) – Curso de Engenharia de Software - Centro Universitário Unidade de Ensino Superior Dom Bosco – UNDB, 2023.

1.Tecnologia Iot. 2. Internet das coisas. 3. Engenharia de software. I. Título.

CDU 004.738.5

CAIQUE DOURADO

SISTEMA DE CONTROLE DE ACESSO EM CONDOMÍNIOS UTILIZANDO TECNOLOGIA IOT: uma abordagem de engenharia de software para eficiência e segurança.

Monografia apresentada ao Curso de Engenharia de Software do Centro Universitário Unidade de Ensino Superior Dom Bosco como requisito parcial para obtenção do grau de Bacharel em Engenharia de Software.

Aprovada em: 19/06/2023.

BANCA EXAMINADORA:

_____

Prof. Me. Allisson Jorge Silva Almeida (Orientador)

Mestre em Inteligência Artificial (UFMA)

Centro Universitário Unidade de Ensino Superior Dom Bosco (UNDB)

_____

Prof. Dr. Giovanni Lucca França da Silva

Doutor em Engenharia Elétrica (UFMA)

Centro Universitário Unidade de Ensino Superior Dom Bosco (UNDB)

_____

Prof. Me. Arlison Wady Sousa Martins

Mestre em Ciência da Computação (UFMA)

Centro Universitário Unidade de Ensino Superior Dom Bosco (UNDB)

Elevo os meus olhos para os montes; de onde me vem o socorro? O meu socorro vem do Senhor, que fez os céus e a terra. Salmos 121:1-2.

AGRADECIMENTOS

Agradeço inicialmente ao Deus Vivo de Israel por ter me sustentado até aqui, ter-me concedido sabedoria e aberto inúmeras portas para que eu pudesse realizar este sonho. Agradeço a todos os meus familiares, em especial a minha mãe e minha avó que me apresentaram o mundo da tecnologia em meados de 2010 quando me puseram para estudar em um curso técnico local de informática. A cada docente que se dispôs a compartilhar seu conhecimento em sala de aula. E especialmente a minha grande amiga/mãe Josy que sempre me incentivou ao longo de minha vida a nunca desistir dos meus sonhos e acreditou que eu conseguiria realizar este sonho. E finalmente agradeço a minha grande e incrível amiga Regina, que ouviu a minha história e tornou-se a resposta de minhas tantas orações.

“Se pude enxergar a tão grande distância, foi subindo nos ombros de gigantes.”
(NEWTON, 1672, p. 1).

RESUMO

A Internet das Coisas (IoT) tem se mostrado uma área promissora de pesquisa e desenvolvimento, conectando dispositivos físicos à Internet e permitindo a troca de informações de maneira inovadora. O conceito de IoT tem ganhado popularidade nos últimos anos, oferecendo uma abordagem autônoma de conectividade entre dispositivos, coletando e transmitindo dados de forma automatizada. O mercado brasileiro de IoT tem experimentado um crescimento significativo, impulsionado pelo avanço da conectividade e adoção de tecnologias inteligentes em diversos setores. A implantação do 5G no país tem sido um fator-chave para esse crescimento, proporcionando maior conectividade e capacidade de rede. Setores como agricultura, saúde, energia, indústria e cidades inteligentes têm se destacado no mercado de IoT no Brasil. Este trabalho teve como objetivo desenvolver uma arquitetura que utiliza a tecnologia IoT para o controle eficiente e seguro da entrada e saída de pessoas em condomínios. A integração de dispositivos inteligentes, como câmeras e sensores, visa facilitar o acesso de moradores e visitantes, mantendo a segurança como prioridade. A solução proposta neste trabalho oferece uma experiência fluida e segura, proporcionando tranquilidade aos moradores e controle rigoroso sobre o acesso às dependências de condomínios. Os desafios abordados incluem a gestão de dados gerados pelos dispositivos IoT, a disponibilidade de cobertura de rede em áreas remotas, a interoperabilidade entre dispositivos de diferentes fabricantes e a criação de uma solução de utilizando um microcontrolador que seja flexível, segura e escalável. O Norte desta pesquisa direciona-se em construir uma solução que flexibilize o acesso de moradores e visitantes em condomínios residenciais, mantendo a segurança do condomínio como prioridade.

Palavras-chave: Internet das Coisas. Engenharia de Software. Esp32Cam. Controle de Acesso. Arquitetura Modular.

ABSTRACT

The Internet of Things (IoT) has proven to be a promising area of research and development, connecting physical devices to the Internet and allowing the exchange of information in an innovative way. The concept of IoT has gained popularity in recent years, offering an autonomous approach to connectivity between devices, collecting and transmitting data in an automated way. The Brazilian IoT market has experienced significant growth, driven by the advancement of connectivity and the adoption of smart technologies in various sectors. The deployment of 5G in the country has been a key driver of this growth, providing greater connectivity and network capacity. Sectors such as agriculture, health, energy, industry and smart cities have stood out in the IoT market in Brazil. This work aimed to develop an architecture that uses IoT technology to efficiently and safely control the entry and exit of people in condominiums. The integration of smart devices, such as cameras and sensors, aims to facilitate access for residents and visitors, keeping safety as a priority. The solution proposed in this work offers a fluid and safe experience, providing peace of mind to residents and strict control over access to condominium facilities. The challenges addressed include the management of data generated by IoT devices, the availability of network coverage in remote areas, the interoperability between devices from different manufacturers and the creation of a solution using a microcontroller that is flexible, secure and scalable. The north of this research is directed towards building a solution that makes access to residential condominiums more flexible, keeping the security of the condominium as a priority.

Keywords: Internet of Things. Software Engineering. Esp32Cam. Access control. Modular Architecture.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ADC	Analog to Digital Converter
ANATEL	Agência Nacional de Telecomunicações
AP	Access Point
APB	Advanced Peripheral Bus
API	Application Programming Interface
COAP	Constrained Application Protocol
CPU	Central Processing Unit
DAC	Digital to Analog Converter
DIY	Do-It-Yourself
DMIPS	Million Instructions per Second
DNS	Domain Network System
ERP	Enterprise Resource Planning
ESP	Eletronic Stability Program
GND	Ground
GPIOs	General Purpose Input/Output
GPRS	General Packet Radio Service
GPS	Global Positioning System
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
I2Cs	Inter-Integrated Circuit
IAB	Internet Architecture Board
IDC	International Data Corporation
IETF	Internet Engineering Task Force
IOT	Internet of Things
JSON	JavaScript Object Notation
JWT	Json Web Token
MCU	Microcontroller Unit
MQTT	Message Queuing Telemetry Transport
MVC	Model View Controller
MVP	Mínimo Produto Viável
NFC	Near Field Communication
NLP	Natural Language Processing

PWA	Progressive Web App
PWM	Pulse Width Modulation
QR	Quick Response Code
RDBMS	Data Base Management System
REST	Representational State Transfer
RFC	Request for Coments
RFID	Radio Frequency Identification
RTOS	Real Time Operating System
SDK	Software Development Kit
SPIs	Comunicação SPI (Serial Peripheral Interface
SQL	Structured Query Language
STA	Station
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
UARTs	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
URL	Uniform Resource Locators
VCC	Voltage
ZB	Zettabytes

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, o conceito de Internet das Coisas tem emergido e ganhado cada vez mais popularidade. A IoT pode ser compreendida como uma abordagem que visa conectar dispositivos de maneira autônoma, permitindo a transmissão e recepção de dados sem a necessidade de intervenção humana (COELHO, 2017). Esses dispositivos podem ser equipados com sensores capazes de interagir com ambientes específicos, coletando informações relevantes e gerando dados de forma automatizada.

> "[...] desde há bastantes anos que se imaginam e implementam soluções baseadas no conceito de ligação de dispositivos à Internet, muito para além dos tradicionais computadores, que foram os primeiros meios de acesso. O que há alguns anos era um conceito é hoje uma realidade em concretização, com muitos meios, pessoas e ideias envolvidos: a IoT (Internet of Things – Internet das Coisas) chegou e vai tomar conta do nosso dia a dia." (COELHO, 2017, p.11).

O mercado brasileiro de IoT tem apresentado um crescimento significativo nos últimos anos, impulsionado pelo avanço da conectividade e pela adoção de tecnologias inteligentes em diversos setores. De acordo com um relatório publicado pela consultoria McKinsey & Company, o Brasil tem o potencial de se tornar um dos principais mercados de IoT na América Latina, com um valor estimado de até US$ 50 bilhões até 2025. (MCKINSEY, 2018).

Um fator-chave impulsionando esse crescimento é a implantação da tecnologia 5G no país. O leilão das faixas de frequência para a implementação do 5G foi realizado em 2021, abrindo caminho para uma maior conectividade e capacidade de rede, o que impulsiona ainda mais a adoção da IoT em diversos setores. (ANATEL, 2021).

Setores como agricultura, saúde, energia, indústria e cidades inteligentes têm se destacado no mercado brasileiro de IoT. Um estudo realizado pela empresa de pesquisa de mercado IDC Brasil - *International Data Corporation*, indica que a indústria manufatureira é um dos setores com maior potencial de crescimento em IoT, com iniciativas que visam melhorar a eficiência operacional, otimizar a cadeia de suprimentos e adotar tecnologias como a análise de dados em tempo real. (IDC BRASIL, 2019).

Trabalhos anteriores que exploraram o tema de automação residencial enfrentaram desafios relacionados ao alto custo dos hardwares utilizados, bem como

à limitação no processamento e suporte a grandes volumes de tráfego de informações.

O objetivo deste trabalho foi o desenvolvimento de uma arquitetura que utiliza a tecnologia IoT para possibilitar o controle eficiente e seguro da entrada e saída de pessoas em condomínios. A integração de dispositivos inteligentes, como câmeras e sensores, garante um acesso simplificado tanto para moradores quanto para visitantes, ao mesmo tempo em que mantém a segurança do condomínio como prioridade. A solução proposta proporciona uma experiência fluida e segura, oferecendo tranquilidade aos moradores e um controle rigoroso sobre o acesso às dependências do condomínio.

1.1 PROBLEMA DE PESQUISA

Atualmente, o mercado oferece uma ampla variedade de dispositivos IoT para diversas funcionalidades, provenientes de startups, entusiastas que transformam seus hobbies em produtos e até mesmo empresas que já atuam no setor de tecnologia. No entanto, uma grande parte dessas soluções enfrenta desafios na gestão de todos os dados gerados. Se a comercialização dessas soluções aumentar em larga escala e não houver uma estrutura adequada para gerenciá-las, as chances de falhas nessas soluções aumentam significativamente.

Os problemas de disponibilidade de dispositivos IoT podem ser desafiadores em várias situações. Uma questão comum é a falta de cobertura de rede em determinadas áreas geográficas, especialmente em regiões rurais ou remotas. Isso pode limitar a capacidade de conexão e comunicação dos dispositivos IoT, dificultando sua disponibilidade e funcionalidade.

Além disso, a interoperabilidade entre dispositivos de diferentes fabricantes e padrões é outra preocupação que afeta a disponibilidade. A falta de padronização e protocolos de comunicação comuns pode levar a incompatibilidades e dificuldades na integração dos dispositivos em uma rede IoT, comprometendo sua disponibilidade e usabilidade.

Além dos desafios mencionados anteriormente, outro objetivo central deste trabalho é o desenvolvimento de uma solução IoT de baixo custo que ofereça flexibilidade e segurança, facilitando a integração de novos dispositivos com o mínimo esforço necessário. Essa abordagem visa consolidar a meta de criar uma solução

flexível e escalável, permitindo a expansão e aprimoramento contínuos do sistema, sem comprometer a acessibilidade financeira.

1.3 OBJETIVOS

O objetivo geral deste projeto é criar uma solução IoT de baixo custo que possibilite a interação e monitoramento simultâneo de entrada e saída de moradores de condomínios residenciais e a autorização de entrada de visitantes de forma remota e segura. A solução também deve permitir a interação com o usuário por meio de um aplicativo para smartphone e oferecer a capacidade de parametrização através de um painel de controle web.

Com base no objetivo geral, foram elaborados os objetivos específicos a seguir:

1. Definir um modelo de aplicação própria que faça comunicação cliente-servidor.
2. Implementar uma API REST, para que o aplicativo cliente consiga se comunicar com o servidor.
3. Codificar o módulo ESP32Cam com o objetivo de enviar e receber requisições do servidor.
4. Implementar o Mínimo Produto Viável (MVP[1]) do aplicativo *Progressive Web App* (PWA) para dispositivos móveis com o objetivo de interagir com a solução, como criar *tokens* de acesso para visitantes, realizar consulta de tokens.

[1] MVP: refere-se à criação da versão mais simples e eficiente de um produto, utilizando recursos mínimos para oferecer o valor principal da ideia. Dessa forma, torna-se possível validar o produto antes do seu lançamento.

1.5 ESTRUTURA DO TRABALHO

A estrutura do trabalho de conclusão de curso é composta por 4 capítulos que se subdividem em diferentes seções que abordam os principais aspectos relacionados à solução proposta. O Capítulo 1, inicia-se com a introdução, onde são apresentados o problema de pesquisa e a questão de pesquisa que norteiam a investigação. Além disso, são definidos os objetivos do trabalho, estabelecendo o que se pretende alcançar com a solução desenvolvida. A estrutura do trabalho também é mencionada, dando uma visão geral das seções subsequentes.

No Capítulo 2, é abordado os diferentes tópicos relevantes para o trabalho. Inicia-se com uma explanação sobre a Internet das Coisas, discutindo suas áreas de aplicação e destacando sua relevância em contextos como residências, varejo, transporte, sensores industriais e saúde. Em seguida, é apresentado o microcontrolador ESP32, discutindo sua arquitetura e características, incluindo detalhes sobre os pinos de energia, comunicação serial e outros aspectos relevantes.

Os protocolos de comunicação também são abordados, com foco especial no protocolo HTTP. São apresentados conceitos gerais sobre o protocolo, suas principais características, métodos de solicitação, cabeçalhos e códigos de status. Em seguida, são mencionadas as tecnologias e ferramentas de desenvolvimento utilizadas no projeto, como *Flutter*, *Dart*, MySQL, *Golang*, JWT e C++.

No Capítulo 3, a arquitetura proposta para a solução é detalhada na seção correspondente. É apresentada a estrutura da solução, incluindo uma visão geral da arquitetura e a descrição dos diferentes componentes, como o dispositivo de captura, o servidor de processamento, o modelo de token QR, o armazenamento de dados em nuvem e a aplicação cliente. Também é abordada a segurança no projeto, com ênfase nas medidas adotadas para garantir a segurança da solução.

Por fim no Capítulo 4, a validação da solução é realizada na seção correspondente, onde são descritas as etapas de criação e decodificação do token, incluindo detalhamentos das funções envolvidas nesses processos.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo estabelece as bases teóricas necessárias para compreender as características do dispositivo controlador e as tecnologias utilizadas no projeto de IoT proposto. Essa fundação teórica é essencial para alcançar os objetivos gerais e específicos do trabalho, permitindo o desenvolvimento de uma solução eficiente e confiável para atender às necessidades do projeto.

2.1 MICROCONTROLADOR ESP32

Segundo Jahnavi et al. (2022) o ESP32 é um microcontrolador integrado com recursos avançados, incluindo conectividade Wi-Fi e Bluetooth. Com seu poder de processamento de 32 bits, design dual core e consumo de energia eficiente, o ESP32 é uma opção acessível para projetos diversos. Sua arquitetura é baseada no chip ESP32S, oferecem flexibilidade e adaptabilidade aos usuários.

> "[...] o módulo ESP32-CAM é uma plataforma acessível e poderosa, que você pode desenvolver projetos inteligentes em Arduino. O módulo suporta transferência de dados via WiFi e Bluetooth. O microcontrolador ESP32-CAM, ao contrário de seu antecessor ESP8266, possui mais interfaces. O ESP32 foi lançado pela Espressif Systems em 2016." (Salikhov et al. ,2021, p. 4, tradução nossa).

O microcontrolador ESP32 é um dispositivo muito versátil e poderoso, utilizado em diversos projetos de eletrônica, automação e IoT. Segundo Mingdian et al. (2020), o ESP32 é um microcontrolador de baixo custo e baixo consumo de energia, com uma ampla gama de recursos e capacidades. Ele é baseado em um processador *dual-core Tensilica* LX6, capaz de rodar a uma frequência de até 240MHz. Além disso, possui suporte a conexões Wi-Fi e *Bluetooth*, bem como diversos periféricos de entrada e saída, como GPIOs, UARTs, SPIs e I2Cs.

De acordo com Souza et al. (2020), o ESP32 é muito utilizado em projetos de IoT, permitindo a criação de dispositivos inteligentes conectados à internet. Ele pode ser utilizado para coletar dados de sensores, controlar atuadores, enviar informações para servidores em nuvem e muito mais. Além disso, a comunicação Wi-Fi e Bluetooth integrada permite a conexão com outros dispositivos e serviços na rede.

Já para Babiuch et al. (2019), uma das grandes vantagens do ESP32 é sua facilidade de programação. Ele pode ser programado em diversas linguagens, como

C++, *MicroPython* e Arduino, além de contar com uma ampla variedade de bibliotecas e exemplos disponíveis na internet. Isso torna o desenvolvimento de projetos com o microcontrolador mais acessível a um público mais amplo de desenvolvedores e entusiastas.

No entanto, é importante ressaltar que o ESP32 também possui algumas limitações. Segundo a fabricante Espressif, o algoritmo de gerenciamento de energia incluído no ESP-IDF pode ajustar a frequência do *Advanced Peripheral Bus* (APB[2]), a frequência da *Central Processing Unit* (CPU) e colocar o chip no modo de suspensão leve para executar um aplicativo com o menor consumo de energia possível, dados os requisitos dos componentes do aplicativo. A ativação dos recursos de gerenciamento de energia tem o custo de aumentar a latência de interrupção. A latência extra depende de vários fatores, como a frequência da CPU, modo *single/dual* core, se a troca de frequência precisa ou não ser feita. A latência extra mínima é de 0,2 us (quando a frequência da CPU é de 240 MHz e a escala de frequência não está habilitada). A latência extra máxima é de 40 us (quando a escala de frequência está habilitada e uma mudança de 40 MHz para 80 MHz é executada na entrada de interrupção).

Outro fator é que a comunicação Wi-Fi pode ser afetada por interferências de outros dispositivos e pela distância em relação ao ponto de acesso.

Apesar dessas limitações, o ESP32 ainda é um dos microcontroladores mais populares e versáteis do mercado. Como apontado por Zhang et al. (2020), sua combinação de baixo custo, baixo consumo de energia, suporte a Wi-Fi e Bluetooth e facilidade de programação o torna uma escolha atraente para projetos de IoT e eletrônica em geral.

O microcontrolador ESP32 é uma ferramenta poderosa e versátil para o desenvolvimento de projetos de IoT e eletrônica. Sua ampla gama de recursos e facilidade de programação o tornam uma escolha popular entre desenvolvedores e entusiastas. No entanto, é importante considerar suas limitações em relação ao gerenciamento de energia e comunicação Wi-Fi.

[2] APB: é um barramento periférico básico, de velocidade moderada e simplificado, criado para dispositivos eletrônicos de menor desempenho.

2.2.1 Arquitetura do Microcontrolador ESP32

ESP32-CAM é uma placa de desenvolvimento baseada em ESP32 de baixo custo com câmera integrada, de tamanho pequeno. É uma solução ideal para aplicações de IoT, construções de protótipos e projetos *Do-It-Yourself* (DIY[3]).

A placa integra WiFi, *Bluetooth* tradicional e BLE de baixa potência, com 2 CPUs LX6 de 32 bits de alto desempenho. Adota arquitetura de *pipeline* de 7 estágios, sensor no chip, sensor *Hall*, sensor de temperatura e assim por diante, e seu ajuste de frequência principal varia de 80MHz a 240MHz.

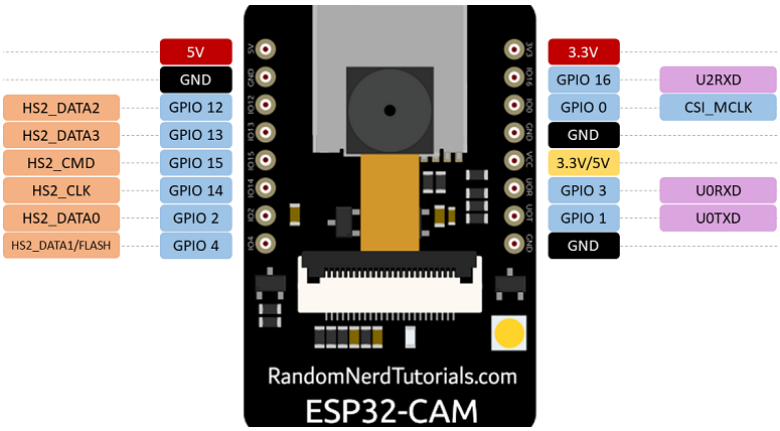
Totalmente compatível com os padrões WiFi 802.11b/g/n/e/i e *Bluetooth* 4.2, pode ser usado como um modo mestre para construir um controlador de rede independente ou como um escravo para outros *Microcontroller Unit* (MCUs[4]) *host* para adicionar recursos de rede aos dispositivos existentes

O ESP32-CAM pode ser amplamente utilizado em várias aplicações IoT. É adequado para dispositivos inteligentes domésticos, controle sem fio industrial, monitoramento sem fio, identificação sem fio de *Quick Response Code* (QR Code[5]), sinais do sistema de posicionamento sem fio e outras aplicações IoT. É uma solução ideal para aplicações IoT.

A seguir a Figura 1 mostra o diagrama de pinagem para o Esp32-cam *AI-Thinker*.

[3] DIY: são projetos ou atividades realizados por indivíduos que preferem fazer algo por conta própria, em vez de comprar um produto ou serviço pronto.
[4] MCU: é um dispositivo eletrônico que integra em um único chip um microprocessador, memória, periféricos de entrada/saída e outros componentes essenciais para o funcionamento de um sistema embarcado.
[5] QR Code: é um tipo de código de barras bidimensional que pode ser escaneado por dispositivos móveis, como smartphones, para acessar informações ou executar ações específicas.

Figura 1 - Pinagem ESP32-CAM



Fonte: Eletro Rules (2022).

2.2.1.1 Pinos de Energia

O ESP32-CAM possui três pinos *Ground* (GND[6]) – preto, e dois pinos de alimentação (vermelho): 3,3V e 5V, respectivamente.

O ESP32-CAM pode ser alimentado pelos pinos 3,3V ou 5V. No entanto, muitas pessoas notaram problemas ao usar 3,3 V para alimentar o ESP32-CAM, portanto, sempre se recomenda o uso do pino de 5 V.

2.2.1.2 Pino de Saída de Energia

Há também o pino *Voltage* (VCC[7]), rotulado na serigrafia (colorido com um retângulo amarelo). Esse pino não deve ser usado para alimentar o ESP32-CAM. Esse é um pino de saída de energia. Pode produzir 5V ou 3,3V.

Seja fornecido com 5V ou 3,3V, o ESP32-CAM produz 3,3V em nessa situação. Dois *pads* estão localizados ao lado do pino VCC. Um é rotulado como 3,3 V e o outro é rotulado como 5 V.

[6] GND: refere-se ao terminal de referência comum ou ponto de aterramento de um circuito.
[7] VCC: tensão. CC é sufixo usado por convenção.

2.2.1.3 Pin de Série

GPIO 1 e GPIO 3 são os pinos seriais (TX e RX, respectivamente). Como o ESP32-CAM não possui um programador embutido, é necessário usar esses pinos para se comunicar com a placa e fazer o upload do código.

Pode-se também usar GPIO 1 e GPIO 3 para conectar outros periféricos como saídas ou sensores após carregar o código. No entanto, não será possível abrir o Serial Monitor e verificar se tudo está indo bem com sua configuração.

2.2.1.4 GPIO 0

GPIO 0 determina se o ESP32 está em modo intermitente ou não. Este GPIO é conectado internamente a um resistor *pull-up* de 10k Ohm.

Quando o GPIO 0 está conectado ao GND, o ESP32 entra em modo intermitente e você pode enviar o código para a placa.

Para fazer o ESP32 rodar “normalmente”, basta desconectar o GPIO 0 do GND.

O ESP32-CAM pode ser amplamente utilizado em várias aplicações IoT. É adequado para dispositivos inteligentes domésticos, controle sem fio industrial, monitoramento sem fio, identificação sem fio QR, sinais do sistema de posicionamento sem fio e outras aplicações IoT. É uma solução ideal para aplicações IoT.

2.2.1.5 Características

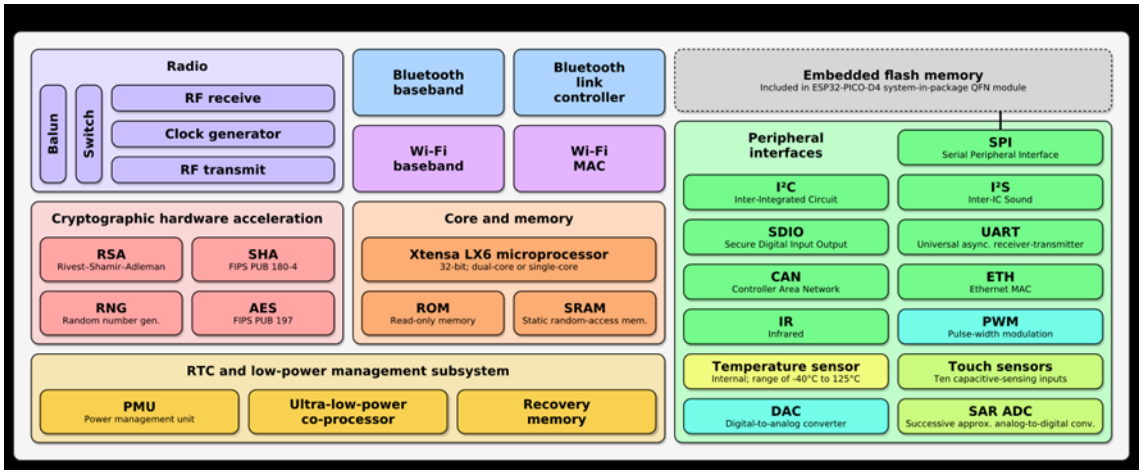
De acordo com o site Eletrogate (2022), o esp32cam, também conhecido como *AI-THINKER*, possui as seguintes características:

1. Velocidade de *clock* de até 160MHz, potência de computação resumida de até 600 *Million Instructions per Second* (DMIPS).
2. SRAM integrada de 520 KB, 4MPSRAM externa.
3. Suporta UART/SPI/I2C/PWM/ADC/DAC.
4. Suporta câmeras OV2640 e OV7670, lâmpada *Flash* embutida.
5. Suporta upload de imagem WiFI.
6. Suporta cartão TF.

7. Suporta vários modos de suspensão.
8. *Lwip* e *FreeRTOS* incorporados.
9. Suporta modo de operação STA/AP/STA+AP.
10. Suporta tecnologia *Smart Config*/*AirKiss.*
11. Suporte para atualizações de firmware local e remota de porta serial.

De acordo com o manual técnico da Espressif, a empresa responsável pelo ESP32, cada núcleo do processador é capaz de operar em até 240 MHz e executar instruções de 32 bits. Além do processador *dual-core*, o ESP32 também possui uma variedade de periféricos, incluindo Wi-Fi, *Bluetooth*, *Ethernet*, UART, I2C, SPI, PWM e ADC, que o tornam altamente versátil e adequado para uma ampla gama de aplicativos. De acordo com a Espressif, a plataforma também suporta uma ampla variedade de protocolos de comunicação, incluindo TCP, UDP, HTTP, MQTT e CoAP, que permitem a integração do ESP32 com sistemas de nuvem e outros dispositivos de IoT.

Figura 2 - Diagrama de blocos de funções do ESP32



Fonte: Winkipedia (2018).

A arquitetura do ESP32 também suporta a execução de programas em Real Time Operating System (RTOS), o que significa que o microcontrolador pode ser usado para implementar sistemas em tempo real com múltiplas tarefas. Em um artigo publicado pelo blog Eletrogate (2022), intitulado: RTOS com ESP32: Como Programar Multitarefas, aborda que o suporte a RTOS é um recurso importante do ESP32 que

permite a criação de sistemas complexos com várias tarefas executando simultaneamente.

Além disso, consoante a documentação oficial, a arquitetura do ESP32 suporta a programação em linguagem C++, o que o torna altamente acessível e adequado para uma ampla gama de desenvolvedores, pois os threads [8]C++ são criadas com base nos *pthreads*[9], que, por sua vez, agrupam as tarefas do *FreeRTOS*.

A arquitetura do ESP32 é altamente versátil, potente e adequada para uma ampla gama de aplicativos de IoT e sistemas embarcados. Com um processador dual-core, uma ampla variedade de periféricos e suporte para RTOS e várias linguagens de programação, o ESP32 é um dos microcontroladores mais poderosos e flexíveis disponíveis atualmente no mercado.

2.3 MICROCONTROLADOR ESP32

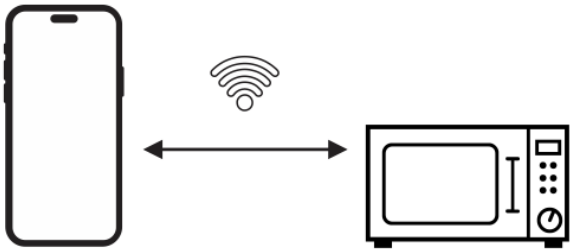
A *Request for Coments* (RFC) 7452, intitulada "Architectural Considerations in Smart Object Networking" e submetida pela *Internet Architecture Board* (IAB), foi publicada pelo *Internet Engineering Task Force* (IETF) em março de 2015. Essa RFC apresenta e define quatro modelos de comunicação para a Internet das Coisas.

2.3.1 Device-to-Device

No modelo de comunicação dispositivo para dispositivo - *device-to-device*, um dispositivo se comunica diretamente com outro dispositivo de destino. Um exemplo dessa abordagem é quando um usuário conecta seu smartphone à cafeteira em sua casa conectada durante o café da manhã. Através de uma conexão Bluetooth, o usuário pode selecionar o tipo de café e a quantidade de açúcar desejada no smartphone. Em seguida, essas informações são transmitidas diretamente para a cafeteira, que prepara o café de acordo com as preferências do usuário.

[8] Threads: são unidades de execução dentro de um programa, também conhecidas como threads de execução ou threads de tarefa.
[9] Pthreads: é uma API para programação de threads em sistemas operacionais baseados em *Portable Operating System Interface* (POSIX), como Unix e Linux.

Figura 3 - Exemplo modelo de comunicação Device-to-Cloud

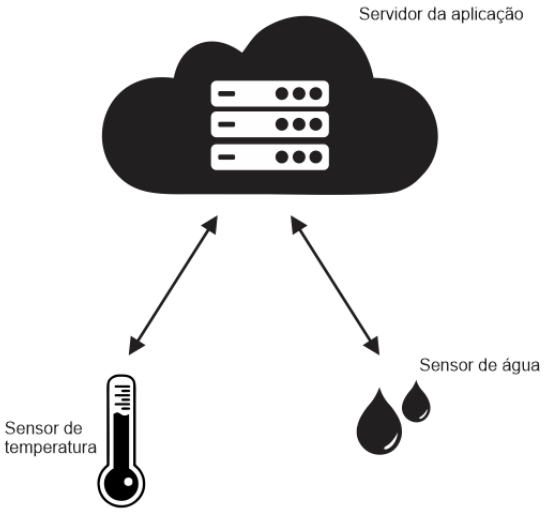


Fonte: Elaborado pelo autor (2023)

2.3.2 Device-to-Cloud

No padrão Device-to-Cloud - Dispositivo para Nuvem, o dispositivo se comunica diretamente com a Internet, sem a necessidade de um equipamento intermediário. Na Figura 4, é exemplificado o uso desse padrão em um carro equipado com uma placa de telemetria que possui um módulo *Global Positioning System* (GPS) e um *General Packet Radio Service (GPRS)* incorporado. As informações de posicionamento do veículo são transmitidas pela rede de dados da telefonia móvel diretamente para um serviço hospedado na nuvem. Essa comunicação ocorre sem depender de uma infraestrutura de rede própria, como Ethernet ou *WiFi*.

Figura 4 - Exemplo modelo de comunicação Device-to-Cloud



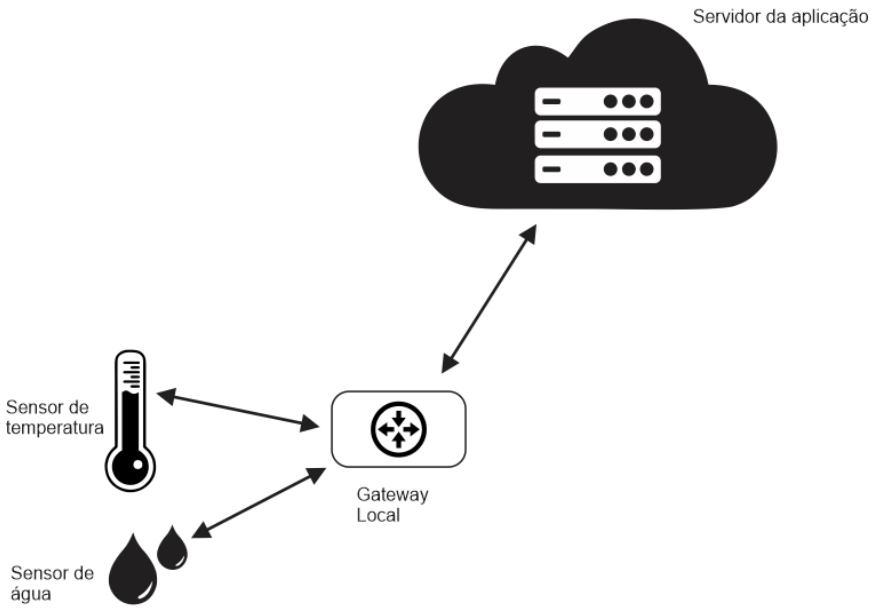
Fonte: Elaborado pelo autor (2023)

2.3.3 Device-to-Gateway

No modelo de comunicação *device-to-gateway* - dispositivo para *gateway*, o dispositivo se conecta a um gateway para ter acesso à Internet. Essa abordagem é ilustrada na Figura 5. Um exemplo comum desse modelo é o uso de pulseiras e plataformas que monitoram exercícios físicos.

Nesse caso, o dispositivo, como uma pulseira de monitoramento, se comunica com um *gateway*, que pode ser um smartphone ou um dispositivo dedicado, por meio de tecnologias como *Bluetooth* ou *Near Field Communication* (NFC). O gateway, por sua vez, possui conectividade à Internet, seja por meio de uma conexão Wi-Fi ou celular. As informações coletadas pela pulseira são enviadas ao *gateway* e, em seguida, transmitidas para a nuvem, onde podem ser processadas e analisadas. Esse modelo permite que os dados sejam compartilhados e acessados por meio de serviços online ou aplicativos móveis para acompanhamento dos exercícios físicos.

Figura 5 - Exemplo modelo de comunicação Device-to-Gateway



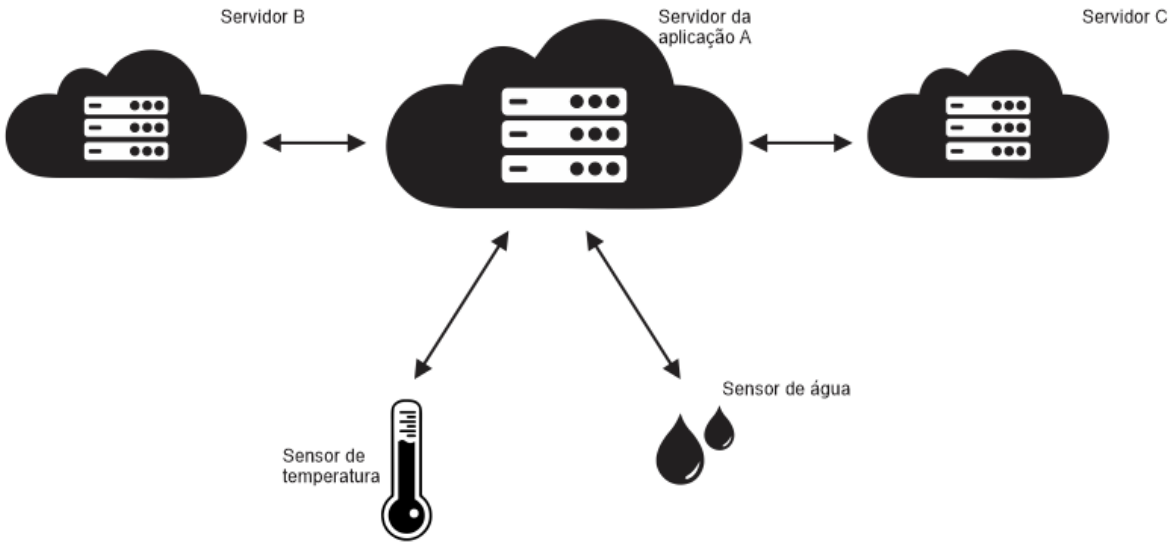
Fonte: Elaborado pelo autor (2023)

2.3.4 Back-End Data Sharing

A Internet das Coisas permite combinar e analisar uma grande quantidade de dados provenientes de diversas plataformas e dispositivos. No entanto, para extrair informações úteis e significativas desses dados, é necessário aplicar o conceito de compartilhamento de dados no *back-end*.

No *back-end* data *sharing*, os dados coletados pelos dispositivos conectados são compartilhados em uma infraestrutura de nuvem ou servidor, onde essas informações podem ser processadas, analisadas e combinadas. Por exemplo, é possível combinar dados meteorológicos, informações sobre o tráfego e as atividades diárias de cada indivíduo para gerar insights valiosos e informações de maior valor agregado.

Figura 6 - Exemplo modelo de comunicação Back-End Data Sharing



Fonte: Elaborado pelo autor (2023)

2.4 PROTOCOLOS DE COMUNICAÇÃO

Os protocolos de comunicação são conjuntos de regras e formatos estabelecidos para permitir a troca de informações entre sistemas computacionais. Esses protocolos definem as especificações técnicas e os procedimentos necessários para estruturar, transmitir e interpretar os dados, garantindo uma comunicação

eficiente e confiável entre os sistemas. Ao seguir essas regras e formatos, os sistemas podem se comunicar de maneira padronizada e interoperável, independentemente das diferenças em suas arquiteturas e tecnologias subjacentes. Isso permite que diferentes dispositivos e plataformas se conectem e compartilhem informações de forma harmoniosa e sem problemas de compatibilidade.

> "[...] um protocolo é um sistema de regras que define como o dado é trafegado dentro ou entre computadores. Comunicações entre dispositivos requer que estes concordem com o formato do dado que estiver sendo trafegado. O conjunto de regras que define esse formato é chamado de protocolo."(MOZILLA, 2022).

Existem vários protocolos de comunicação amplamente utilizados, cada um com suas características e finalidades específicas. A seguir, são apresentados alguns dos protocolos mais comuns:

TCP/IP - Transmission Control Protocol/Internet Protocol: De acordo com a Fortinet, uma empresa renomada no mercado de segurança da informação, esse protocolo é um padrão de comunicação que permite que programas, aplicativos e dispositivos de computação troquem mensagens em uma rede. Ele foi projetado para enviar pacotes pela Internet e garantir a entrega bem-sucedida de dados e mensagens pelas redes.

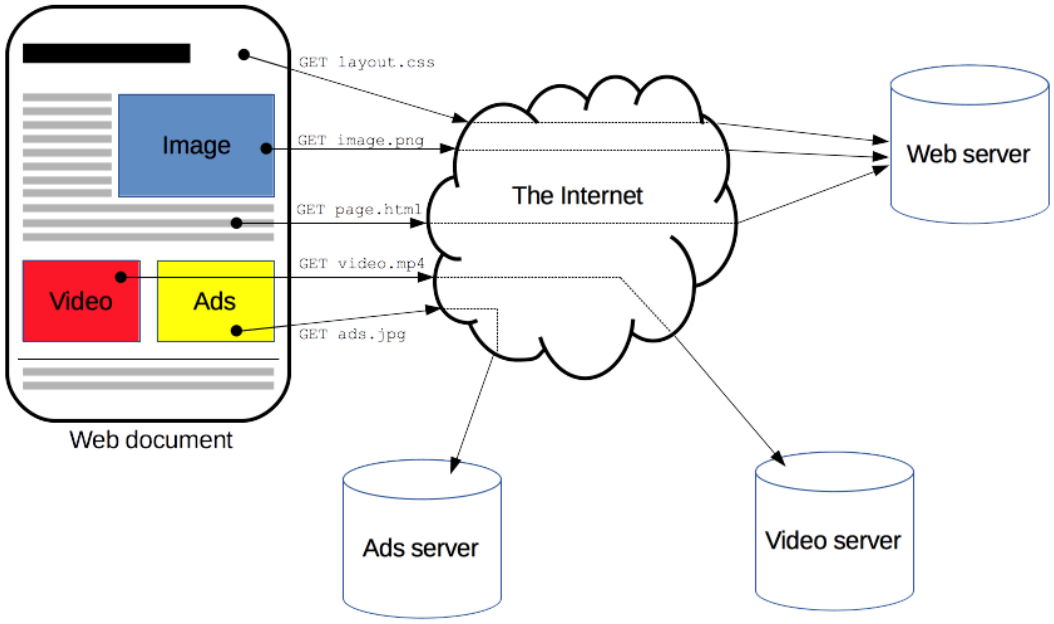
UDP - User Datagram Protocol: Conforme mencionado pela Cloudflare, o Protocolo User Datagram Protocol (UDP) é um protocolo de comunicação amplamente utilizado na Internet para transmissões de dados com tempo de vida limitado, como reproduções de vídeo ou consultas DNS. Ao contrário do TCP, o UDP não estabelece uma conexão formal antes da transferência de dados, o que acelera as comunicações. Essa abordagem permite uma transferência rápida de dados, mas também pode resultar em perda de pacotes durante o trânsito, além de abrir brechas para ataques DDoS.

HTTP - Hypertext Transfer Protocol: Conforme mencionado pela Cloudflare, o *Hypertext Transfer Protocol* (HTTP) é um protocolo fundamental para a World Wide Web, sendo utilizado para carregar páginas da web por meio de links de hipertexto. Ele é um protocolo de camada de aplicativo projetado para transferir informações entre dispositivos em rede e opera sobre outras camadas da pilha de protocolos de rede. Em um fluxo típico de comunicação via HTTP, uma máquina cliente envia uma solicitação a um servidor, que por sua vez responde com uma mensagem de resposta.

2.4.1 Visão Geral Sobre o Protocolo HTTP

Segundo a Mozilla, HTTP é um protocolo cliente-servidor fundamental na Web, permitindo a obtenção de recursos como documentos HTML. Ele é responsável pela troca de dados entre o cliente, geralmente um navegador, e o servidor, possibilitando a reconstrução de documentos completos a partir de sub-documentos como texto, layout, imagens, vídeos e scripts. Com o HTTP, é possível criar páginas web ricas e interativas, acessando uma variedade de conteúdos e serviços online.

Figura 7 - Fluxo geral do HTTP



Fonte: Mozilla (2022)

> "[...] clientes e servidores se comunicam trocando mensagens individuais (ao contrário de um fluxo de dados). As mensagens enviadas pelo cliente, geralmente um navegador da *Web*, são chamadas de solicitações (*requests*), ou também requisições, e as mensagens enviadas pelo servidor como resposta são chamadas de respostas - responses"(MOZILLA, 2022).

Segundo a descrição fornecida, o HTTP é um dos protocolos mais amplamente utilizados na comunicação na Internet. Ele possibilita que os clientes, como navegadores web, solicitem recursos a servidores web e recebam as respostas correspondentes. Essa troca de informações entre o cliente e o servidor permite o carregamento de páginas da web, o envio de formulários, o download de arquivos e outras operações relacionadas à interação entre o usuário e a aplicação web. O HTTP desempenha um papel fundamental na transferência de dados na World Wide Web.

2.4.1 Principais Características do HTTP

As principais características do HTTP incluem ser um protocolo *stateless*, o que permite uma fácil escalabilidade. O HTTP é baseado no modelo cliente-servidor, em que o cliente envia solicitações para o servidor e espera por respostas. As solicitações são feitas utilizando os métodos, permitindo operações diferentes com recursos web. O HTTP utiliza *Uniform Resource Locators* (URL) para identificar os recursos a serem acessados. Além disso, é um protocolo textual, o que significa que as mensagens são enviadas em formato legível por humanos, facilitando a depuração e inspeção das comunicações. O HTTP também suporta cabeçalhos, que contêm informações adicionais sobre a solicitação ou resposta, como cookies, tipo de conteúdo, cache, entre outros. Essas características tornaram o HTTP um dos protocolos fundamentais para a comunicação na web, permitindo a transferência eficiente de recursos e a construção de aplicações web interativas.

2.4.1.1 Stateful

Para Zis (2006), em sistemas dependentes de protocolos, cada nó da rede mantém um conjunto de endereços IP. Portanto, é necessário ter uma entidade adicional para atribuir um endereço IP a um novo membro da rede. Esses sistemas podem ser classificados de acordo com a maneira como a tabela de alocação de endereços é mantida. O autor define a classificação como sendo:

Manutenção centralizada da tabela de alocação: Nesse caso, apenas um nó na rede, conhecido como nó central, é responsável por distribuir endereços para os novos nós. O nó central deve estar sempre acessível na rede para atender às solicitações. Se o nó central deixar a rede, um novo nó será escolhido dinamicamente para desempenhar essa função. No entanto, quando a transferência da tabela de alocação de endereços para o novo nó central não for possível, os nós na rede devem ser notificados para informar seus endereços e serem registrados na tabela, ou devem fazer uma nova solicitação de endereço. Essa abordagem não é adequada para cenários altamente dinâmicos, nos quais a troca do nó central ocorre com muita frequência.

Manutenção distribuída de uma tabela de alocação comum: Nesse caso, a tabela de alocação de endereços é mantida de forma distribuída, mas compartilhada

entre os nós da rede. Cada nó possui uma cópia da tabela e pode atribuir endereços a novos membros. Essa abordagem permite uma maior flexibilidade e escalabilidade em comparação com a manutenção centralizada da tabela de alocação.

Manutenção distribuída de uma tabela de alocação individual: Nessa abordagem, cada nó possui sua própria tabela de alocação de endereços e é responsável por atribuir endereços a novos membros. Cada nó mantém seu próprio conjunto de endereços disponíveis e realiza a atribuição localmente. Essa abordagem oferece autonomia e independência aos nós, mas pode resultar em fragmentação da tabela de alocação em redes grandes.

2.4.1.2 Sateless

A abordagem independente de protocolos, não há uma tabela de alocação de endereços mantida. Os nós constroem seus próprios endereços com base em um número randômico ou no identificador do hardware. Nesse processo, é necessário um mecanismo chamado Detecção de Endereço Duplicado (DAD) para garantir a unicidade do endereço. O mecanismo de DAD é a parte mais importante desses protocolos. A configuração dos endereços pode ocorrer antes ou depois da execução do mecanismo de detecção de endereços duplicados.

Uma das principais dificuldades desses protocolos é a fusão de redes. Para lidar com esse cenário, o mecanismo de DAD pode ser executado apenas após a junção das redes ou de forma contínua. No primeiro caso, quando a fusão é detectada, todos os nós devem ser notificados para repetir o processo de detecção de endereços duplicados. Isso geralmente requer o envio de mensagens de difusão, o que pode consumir bastante largura de banda. Por outro lado, a execução contínua desse mecanismo pode ser alcançada por meio de sua execução periódica ou integração com o protocolo de roteamento. A abordagem de integração com o protocolo de roteamento é mais eficiente, pois causa menos sobrecarga na rede, porém é menos eficiente na detecção de endereços duplicados. (Zis, 2006).

2.4.1.3 Métodos de Solicitação

Gourley e Totty, afirmam em seu livro *HTTP: The Definitive Guide*, que cada método tem um propósito específico, como obter recursos, enviar dados, atualizar ou excluir informações, sendo eles: *GET*[10], *POST*[11], *PUT*[12], *DELETE*[13] .

> "[...] o HTTP suporta vários comandos de solicitação diferentes, chamados métodos HTTP.
> Toda mensagem de requisição HTTP tem um método. O método informa ao servidor qual ação executar (obter uma página da Web, executar um programa de gateway, excluir um arquivo etc.)". (Gourleyn Totty, 2022, p. 8, tradução nossa)

2.4.1.4 Cabeçalhos

Segundo o Mozilla, os cabeçalhos HTTP permitem que o cliente e o servidor transmitam informações adicionais juntamente com a solicitação ou a resposta HTTP. Um cabeçalho de solicitação é composto por um nome, que não diferencia maiúsculas de minúsculas, seguido por dois pontos ':' e o valor correspondente. Qualquer espaço em branco antes do valor é ignorado.

2.4.1.5 Códigos de Status

O HTTP usa códigos de status para indicar o resultado da solicitação. Para o Mozilla (2022)"Os códigos de status das respostas HTTP indicam se uma requisição HTTP foi corretamente concluída. As respostas são agrupadas em cinco classes". Alguns exemplos de códigos de status comuns incluem 200 *OK* - solicitação bem-sucedida, 404 *Not Found* - recurso não encontrado e 500 *Internal* Server *Error* - erro interno do servidor. Observe a figura a seguir contendo um breve resumo sobre os códigos de status de respostas a requisições. Observe a Figura 3, onde é categorizado os códigos http.

---

[10] GET: Destinado a Ler/recuperar uma representação de um recurso.

[11] POST: Usado para criar/gravar novos recursos.

[12] PUT: Assim como o POST, ele cria um recurso, mas também é capaz de atualizá-lo. Para tanto, esse método é somente usado para realizar atualizações de recursos.

[13] DELETE: Usado para excluir um recurso. Nesse método, várias solicitações idênticas excluirão o mesmo recurso

Figura 8 - Categorias dos códigos HTTP

Fonte: Warcontent (2022)

2.5 TECNOLOGIAS E FERRAMENTAS DE DESENVOLVIMENTO

No desenvolvimento do projeto em questão, uma cuidadosa seleção de tecnologias foi realizada, levando em consideração a capacidade de comunicação entre os dispositivos de forma eficiente e confiável. Para garantir uma infraestrutura segura, foram escolhidas tecnologias que atendessem às necessidades específicas do projeto focando principalmente na segurança dos dados processados.

2.5.1 Flutter

O *Flutter*, em sua primeira versão, era chamado de "*Sky*" e era projetado para ser executado no sistema operacional Android. Foi revelado por Eric Seidel durante a cúpula de desenvolvedores do *Dart* em 2015, com o objetivo declarado de renderizar de forma consistente a 120 quadros por segundo.

No evento Google *Developer Days* em Xangai, em setembro de 2018, o Google anunciou o *Flutter* Release Preview 2. Essa versão representou o último grande lançamento antes do lançamento oficial do *Flutter* 1.0. Em 4 de dezembro daquele ano, o *Flutter* 1.0 foi lançado no evento *Flutter* Live, denotando a primeira versão estável do framework. Em 11 de dezembro de 2019, o *Flutter* 1.12 foi lançado no evento *Flutter Interactive*.

Em 6 de maio de 2020, o kit *Software Development Kit* (SDK), do Dart versão 2.8 e *Flutter* 1.17.0 foram lançados, adicionando suporte para a API Metal que melhora o desempenho em dispositivos iOS em aproximadamente 50%, bem como novos widgets de materiais, rastreamento de rede e ferramentas de desenvolvimento.

No dia 3 de março de 2021, durante o evento online *Flutter Engage*, o Google lançou o Flutter 2, uma atualização significativa do framework. Essa versão trouxe suporte oficial para aplicativos baseados na web, introduzindo um novo renderizador chamado *Canvas Kit* e widgets específicos para a web. Além disso, o *Flutter* 2 inclui suporte antecipado para o desenvolvimento de aplicativos de desktop para Windows, macOS e Linux, juntamente com melhorias nas APIs de extensão de aplicativos.

Uma das principais mudanças do Flutter 2 foi a transição para o Dart 2.0, que apresentava o conceito de *null safety* - segurança nula. Essa transição gerou algumas alterações e problemas com diversos pacotes externos. No entanto, a equipe do Flutter forneceu instruções e ferramentas para ajudar os desenvolvedores a mitigar esses problemas e garantir a compatibilidade com a nova versão.

Em 8 de setembro de 2021, o Dart 2.14 e o *Flutter* 2.5 foram lançados pelo Google. A atualização trouxe melhorias para o modo de tela cheia do Android e a versão mais recente do *Material Design* do Google chamada *Material You*. O *Dart* recebeu duas novas atualizações, padronizando as condições do *lint* e marcando o suporte para *Apple Silicon* como estável.

Em 12 de maio de 2022, o Google anunciou o lançamento do *Flutter* 3 e do Dart 2.17. Esta atualização expandiu o número total de plataformas suportadas para seis, incluindo suporte estável para Linux e macOS nos processadores Intel e Apple Silicon.

Em 30 de agosto de 2022, o *Flutter* 3.3 foi anunciado. Esta versão apresentou interoperabilidade entre *Objective*-C e Swift, é uma prévia de um novo mecanismo de renderização chamado "*Impeller*", que visa reduzir a interrupção causada pela compilação de *shader*. Em 25 de janeiro de 2023, o *Flutter* 3.7 foi anunciado.

Na documentação oficial da ferramenta, dá-se a definição (em tradução direta): "*Flutter* é uma estrutura de código aberto do Google para criar aplicativos multiplataforma belos e compilados nativamente a partir de uma única base de código". Sendo assim, o *Flutter* é uma plataforma de desenvolvimento de interfaces de usuário que tem como objetivo permitir a reutilização de código em diferentes sistemas operacionais, como iOS e Android. Ele permite que os aplicativos interajam diretamente com os serviços nativos de cada plataforma subjacente. A proposta é possibilitar que os desenvolvedores criem aplicativos de alto desempenho, oferecendo

uma experiência nativa em diversas plataformas, abraçando as diferenças existentes enquanto compartilham o máximo de código possível.

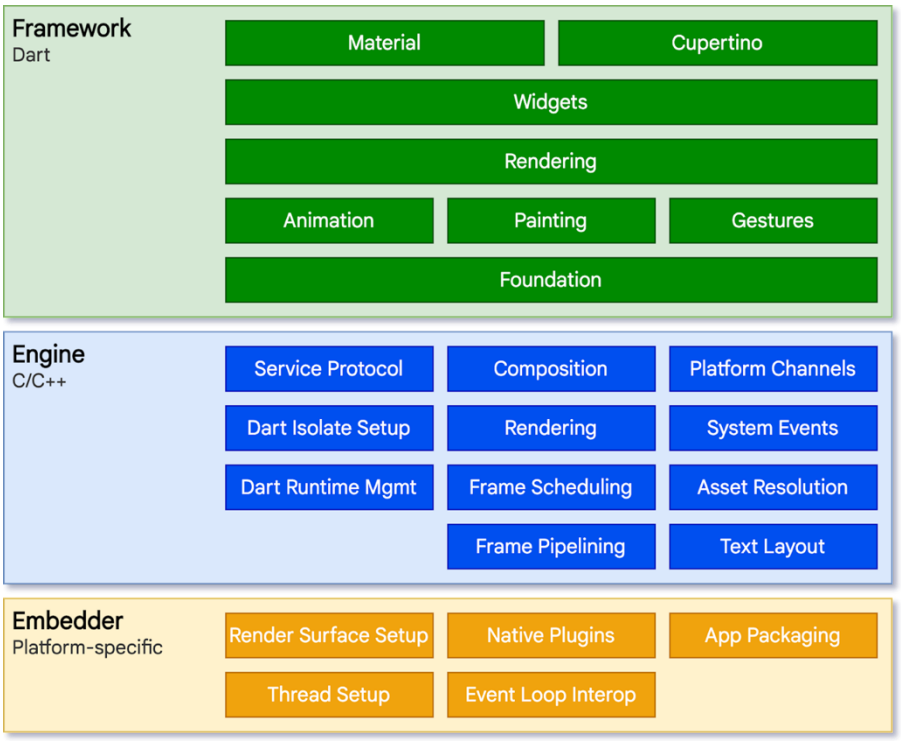
2.5.1.1 Arquitetura do Flutter

Baseado na documentação oficial, durante o processo de desenvolvimento, os aplicativos *Flutter* são executados em uma máquina virtual - VM, o que permite o recarregamento dinâmico de alterações sem a necessidade de uma recompilação completa. Já para o lançamento, os aplicativos *Flutter* são compilados diretamente para o código de máquina, seja para instruções Intel x64 ou ARM, ou até mesmo para *JavaScript*, caso sejam direcionados para a web.

O *Flutter* é uma estrutura de desenvolvimento de código aberto, licenciada sob uma licença BSD permissiva. Além disso, ele possui um ecossistema próspero de pacotes de terceiros que complementam e ampliam a funcionalidade da biblioteca principal. Esses pacotes adicionais são desenvolvidos pela comunidade e contribuem para a diversidade de recursos e ferramentas disponíveis para os desenvolvedores *Flutter*.

O *Flutter* foi concebido como um sistema modular e extensível em camadas. Ele é composto por um conjunto de bibliotecas independentes, sendo que cada uma delas depende da camada subjacente. Não há camadas com acesso privilegiado à camada inferior, e todas as partes do *framework* são projetadas para serem opcionais e substituíveis. Isso significa que as diferentes camadas podem ser personalizadas ou substituídas de acordo com as necessidades do desenvolvedor, garantindo flexibilidade e adaptabilidade ao utilizar o *Flutter*.

Figura 9 - Diagrama de arquitetura de camadas Flutter



Fonte: Flutter Docs

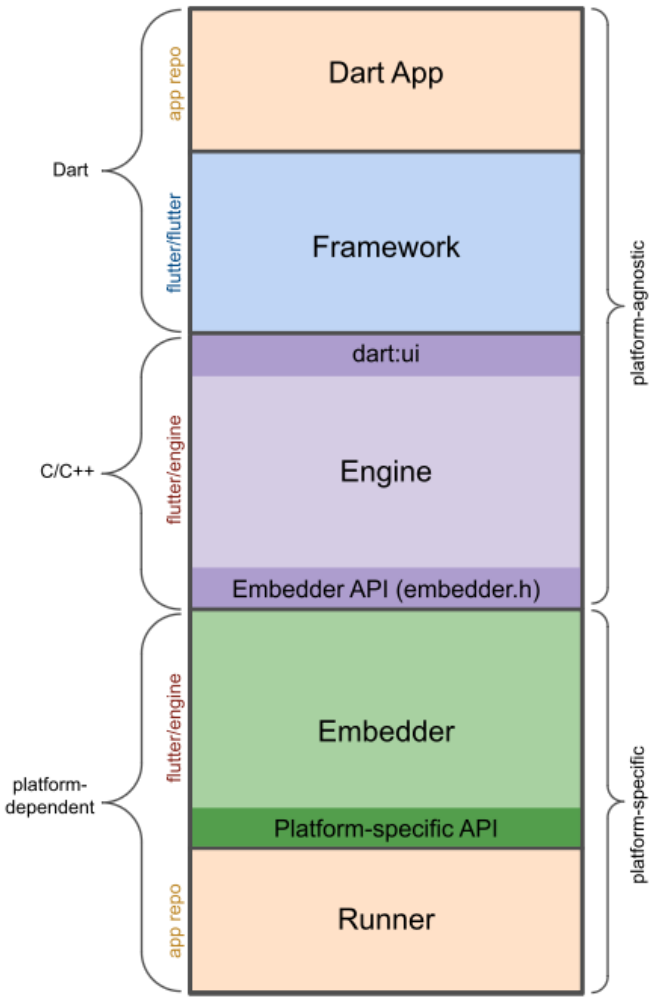
Como exposto na documentação oficial (em tradução direta):

> "[...]Para o sistema operacional subjacente, os aplicativos *Flutter* são empacotados da mesma forma que qualquer outro aplicativo nativo. Um incorporador específico da plataforma fornece um ponto de entrada; coordena com o sistema operacional subjacente para acesso a serviços como superfícies de renderização, acessibilidade e entrada; e gerencia o loop de eventos de mensagem. O incorporador é escrito em uma linguagem apropriada para a plataforma: atualmente Java e C++ para Android, Objective-C/Objective-C++ para iOS e macOS e C++ para Windows e Linux. Usando o incorporador, o código *Flutter* pode ser integrado a um aplicativo existente como um módulo ou o código pode ser todo o conteúdo do aplicativo. O *Flutter* inclui vários incorporadores para plataformas de destino comuns, mas também existem outros incorporadores. "(FLUTTER [s.d.]).

2.5.1.2 Anatomia do Flutter

A ilustração abaixo apresenta uma visão geral dos diferentes componentes que compõem um aplicativo *Flutter* típico criado por meio do comando "*flutter create*". Ela mostra a posição do *Flutter Engine* nessa estrutura, destaca os limites da API e identifica os repositórios onde cada parte individual reside. A legenda abaixo ajuda a esclarecer alguns termos comumente utilizados para descrever as diferentes partes de um aplicativo *Flutter*.

Figura 10 - Anatomia do Flutter



Fonte: Flutter Docs

2.5.1.2.1 Dart App

Compõe widgets na interface do usuário desejada e implementa a lógica de negócios. Essas tarefas são de responsabilidade do desenvolvedor do aplicativo.

2.5.1.2.2 Framework (Código-Fonte)

Fornece API de nível superior para criar aplicativos de alta qualidade (por exemplo, widgets, teste de cliques, detecção de gestos, acessibilidade, entrada de texto) e compõe a árvore de widgets do aplicativo em uma cena.

2.5.1.2.3 Engine (Código-Fonte)

O *Flutter Engine* é responsável por rasterizar cenas compostas, fornecer a implementação de baixo nível das principais APIs do *Flutter* (como gráficos, layout de texto e tempo de execução do *Dart*) e expor sua funcionalidade à estrutura usando a API dart:ui. Além disso, o *Flutter Engine* se integra a uma plataforma específica usando a API *Embedder* do *Engine*.

2.5.1.2.4 Embedder (Código-Fonte)

Coordena com o sistema operacional subjacente para acesso a serviços como superfícies de renderização, acessibilidade e entrada, gerencia o loop de eventos e expõe a API específica da plataforma para integrar o *Embedder* aos aplicativos.

2.5.1.2.5 Runner

Compõe as partes expostas pela API específica da plataforma do Incorporador em um pacote de aplicativo executável na plataforma de destino. Parte do modelo de aplicativo gerado pelo *flutter create*, de propriedade do desenvolvedor do aplicativo.

2.5.2 Dart

De acordo com a documentação oficial, *Dart* é uma linguagem de programação desenvolvida pelo Google, projetada para construir aplicativos de alta performance para a web, desktop e dispositivos móveis. Ela combina características de linguagens estáticas e dinâmicas, permitindo um desenvolvimento eficiente e escalável.

A documentação oficial destaca que o *Dart* é uma linguagem orientada a objetos, com suporte a herança, polimorfismo e interfaces. Isso permite a criação de código modular e reutilizável, promovendo a manutenção e a extensibilidade do software. Além disso, o *Dart* oferece suporte a bibliotecas e pacotes, permitindo que os desenvolvedores compartilhem e reutilizem código de terceiros

Além disso, o Dart possui uma rica biblioteca padrão, abrangendo desde manipulação de *strings* e coleções até recursos avançados, como acesso a APIs de rede e manipulação de arquivos. A documentação oficial fornece uma referência detalhada dessas bibliotecas, bem como uma ampla gama de tutoriais e exemplos para auxiliar os desenvolvedores no aprendizado e na utilização eficaz do *Dart*.

Essas qualidades fizeram com que o Dart se tornasse a linguagem base para a codificação de aplicativos utilizando o *framework Flutter*. O *Flutter* aproveita as capacidades do Dart para fornecer uma experiência de desenvolvimento eficiente, permitindo a criação de aplicativos multiplataforma com facilidade e alto desempenho. A combinação do *Dart* com o *Flutter* tem sido valorizada pela comunidade de desenvolvedores devido à sua eficácia e capacidade de entregar resultados de qualidade.

> "Por utilizar soluções como AOT (*Ahead Of Time*), JIT (*Just In Time*) e permitir que não seja necessário a separação de layout declarativo como para JSX ou XML e a pequena curva de aprendizado para programadores pois utiliza características tanto de linguagens estáticas quanto de linguagens dinâmicas, a utilização da linguagem *Dart* no framework *Flutter* se tornou inquestionável." (BUENO, 2021, 14)

Em resumo, de acordo com a documentação oficial, o *Dart* é uma linguagem de programação moderna e orientada a objetos, projetada para criar aplicativos de alto desempenho para a *web*, *desktop* e dispositivos móveis. Sua sintaxe concisa, suporte a programação assíncrona, foco em performance e capacidade de desenvolvimento multiplataforma tornam o Dart uma escolha atraente para desenvolvedores que buscam produtividade e eficiência no desenvolvimento de aplicativos.

2.5.3 MySQL

De acordo com a documentação oficial, o MySQL é um sistema de gerenciamento de *Relational Database Management System* (RDBMS) de código aberto amplamente utilizado em aplicações web e empresariais. Ele oferece uma plataforma robusta e confiável para armazenar, gerenciar e recuperar dados de forma eficiente.

O MySQL é conhecido por sua estabilidade, desempenho e escalabilidade. Ele suporta grandes volumes de dados e pode lidar com cargas de trabalho intensivas, tornando-o uma escolha popular para empresas de todos os tamanhos.

A documentação oficial destaca a flexibilidade do MySQL em lidar com diferentes tipos de dados. Ele suporta uma ampla gama de tipos de dados, como números, *strings*, datas, imagens e muito mais. Além disso, o MySQL oferece suporte a recursos avançados, como chaves estrangeiras, transações ACID - Atomicidade, Consistência, Isolamento e Durabilidade e índices, que garantem a integridade dos dados e facilitam a recuperação eficiente das informações.

O MySQL também é conhecido por sua compatibilidade e interoperabilidade. Ele suporta a linguagem de consulta *Structured Query Language (SQL)*, que é amplamente utilizada para manipulação e consulta de bancos de dados relacionais. Além disso, o MySQL é compatível com várias plataformas e sistemas operacionais, permitindo que os desenvolvedores implantem seus aplicativos em diferentes ambientes.

Em suma, de acordo com a documentação oficial, o MySQL é um sistema de gerenciamento de banco de dados relacional de código aberto, altamente confiável e escalável. Sua estabilidade, desempenho, compatibilidade com SQL e recursos avançados tornam-no uma escolha popular para aplicações web e empresariais. Ao seguir as orientações da documentação oficial, os usuários podem aproveitar ao máximo os recursos e benefícios oferecidos pelo MySQL.

2.5.4 Golang

De acordo com a documentação oficial, *Golang*, também conhecida como Go, é uma linguagem de programação de código aberto desenvolvida pelo Google. Ela foi projetada para ser simples, eficiente e escalável, permitindo o desenvolvimento de software confiável e de alto desempenho.

Seguindo as orientações da documentação oficial, uma das principais características do *Golang* é sua sintaxe concisa e clara. A linguagem foi projetada para ser fácil de ler e escrever, com um conjunto reduzido de palavras-chave e uma estrutura de controle simplificada. Essa simplicidade facilita a compreensão do código e reduz a ocorrência de erros.

Outro aspecto destacado na documentação é a eficiência do *Golang*. A linguagem foi projetada para executar de forma rápida e eficiente, mesmo em sistemas com recursos limitados. O *Golang* possui um coletor de lixo - *garbage collector* que gerencia automaticamente a alocação e desalocação de memória, aliviando os desenvolvedores dessa tarefa. Além disso, a linguagem suporta concorrência nativamente, permitindo a execução de múltiplas *goroutines* - rotinas leves que compartilham a mesma memória, facilitando a criação de programas paralelos e concorrentes.

O *Golang* também possui uma biblioteca padrão abrangente, com suporte para uma ampla gama de funcionalidades, desde manipulação de *strings* e operações matemáticas até criptografia e acesso a bancos de dados. Além disso, a comunidade de desenvolvedores do *Golang* é ativa e engajada, fornecendo uma ampla variedade de pacotes e bibliotecas de terceiros para expandir as capacidades da linguagem.

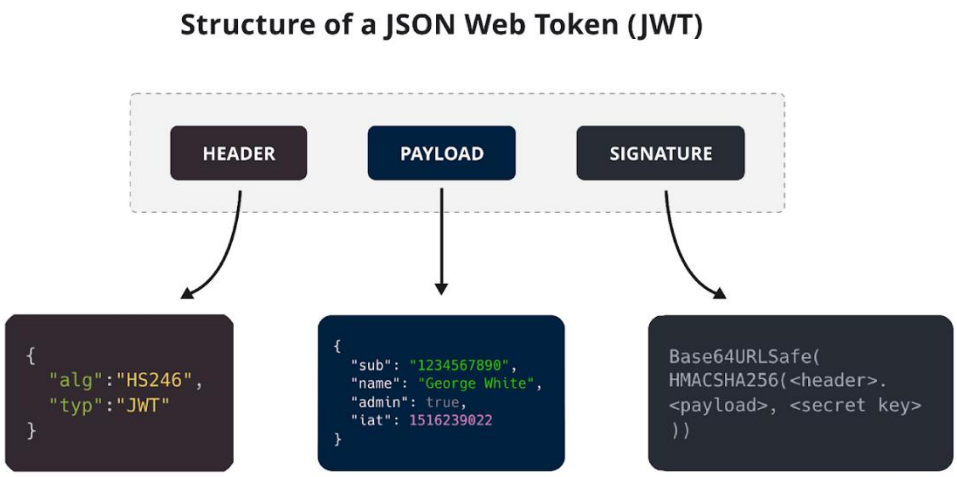
Em síntese, de acordo com a documentação oficial, o *Golang* é uma linguagem de programação simples, eficiente e escalável. Sua sintaxe clara e concisa, eficiência de tempo de execução, suporte nativo à concorrência e ênfase na simplicidade tornam-na uma escolha atraente para o desenvolvimento de uma ampla variedade de aplicativos. Ao seguir as orientações da documentação, os desenvolvedores podem aproveitar ao máximo os recursos e benefícios oferecidos pelo *Golang*.

2.5.5 JWT

De acordo com a documentação oficial, o JWT - *JSON Web Token* é um padrão aberto (RFC 7519) para a criação de *tokens* de autenticação e compartilhamento de informações entre diferentes partes de um sistema. Ele é um formato compacto e autossuficiente, permitindo que as informações sejam transmitidas de forma segura entre os participantes.

Seguindo suas diretrizes oficiais, um token JWT consiste em três partes separadas por pontos: *Header*, *Payload* e *Signature*. Observe a Figura 5:

Figura 11 - Estrutura do JWT



Fonte: SuperTokens (2022)

O header contém informações sobre o tipo de *token* e o algoritmo de criptografia usado na assinatura. O *payload* contém as informações adicionais desejadas, como o ID do usuário ou outras reivindicações relevantes, por fim, a *signature* é gerada combinando o header, o *payload* e a *secret-key,* garantindo a integridade do token.

A documentação oficial destaca que um dos principais benefícios do JWT é a sua capacidade de autenticar e autorizar solicitações de forma eficiente e segura. Ao receber um *token* JWT, o destinatário pode verificar sua autenticidade, pois a assinatura é verificada usando a chave secreta compartilhada. Isso permite que o sistema confie nas informações contidas no token e tome decisões baseadas nesses dados.

Além disso, a documentação oficial ressalta que o JWT é independente de estado - *stateless*, o que significa que todas as informações necessárias para validar e processar o token estão contidas nele mesmo. Isso evita a necessidade de armazenar informações de sessão no servidor, simplificando a escalabilidade e a manutenção do sistema.

Outro aspecto importante destacado na documentação é a capacidade de incluir reivindicações personalizadas na carga útil do *token*. Essas reivindicações podem fornecer informações adicionais sobre o usuário ou controlar o acesso a recursos específicos. O JWT é flexível nesse aspecto, permitindo que as aplicações definam suas próprias regras e estruturas de reivindicações de acordo com suas necessidades.

Resumidamente, o JWT é um padrão aberto e seguro para a criação de *tokens* de autenticação. Sua estrutura autossuficiente e independente de estado torna-o uma escolha popular para a autenticação de APIs e serviços web. Seguir as diretrizes da documentação é fundamental para garantir a implementação correta e segura do JWT em um sistema.

2.5.6 C++

Seguindo as diretrizes fornecidas pela documentação oficial do C++, a linguagem de programação C++ é uma linguagem de alto nível e multiparadigma amplamente utilizada no desenvolvimento de software. Ela foi desenvolvida como uma extensão da linguagem C, com o objetivo de adicionar recursos de programação orientada a objetos e outras funcionalidades avançadas. Pode-se afirmar que o C++ é altamente eficiente e oferece um desempenho excepcional. Essa característica torna a linguagem muito popular em aplicações que exigem uma resposta rápida, como jogos, sistemas embarcados e aplicativos com intensivo processamento de dados. Além disso, o C++ proporciona um alto nível de controle sobre a memória, permitindo que os desenvolvedores gerenciem o uso de recursos de forma precisa e eficiente.

De acordo com a documentação oficial, o C++ suporta os princípios e conceitos da programação orientada a objetos, como encapsulamento, herança e polimorfismo. Isso permite que os programadores criem um código modular, reutilizável e de fácil manutenção. Além disso, a linguagem oferece recursos avançados, como templates, que possibilitam a criação de código genérico, e exceções, que permitem o tratamento adequado de erros e exceções durante a execução do programa.

Uma vantagem destacada pela documentação oficial é a compatibilidade do C++ com o código C existente. Isso significa que programas escritos em C podem ser facilmente integrados em projetos C++ e vice-versa. Essa característica proporciona uma transição suave para os desenvolvedores que estão migrando de C para C++ ou que precisam combinar código legado com novas funcionalidades.

No entanto, é importante considerar as informações presentes na documentação oficial do C++, pois ela destaca alguns desafios da linguagem. Sua

sintaxe pode ser complexa, exigindo um cuidado extra na escrita do código. Além disso, o gerenciamento manual de memória pode ser mais complexo em comparação com outras linguagens de programação de alto nível.

3 ARQUITETURA PROPOSTA

A arquitetura modular *Model View Controller* (MVC) proposta para este projeto no contexto IoT é projetada para permitir a captura eficiente e o processamento de tokens no formato de QR *code* em tempo real, bem como a integração com outros dispositivos e serviços. Ela busca garantir uma experiência fluida e sem interrupções para os usuários finais, ao mesmo tempo em que oferece a flexibilidade necessária para lidar com diferentes cenários e requisitos de uso.

Um dos principais componentes da arquitetura é o dispositivo IoT responsável pela captura dos tokens. Esse dispositivo é equipado com uma câmera capaz de registrar imagens dos tokens em formato QR *code* e transmitir os dados capturados para processamento. Além disso, o dispositivo deve ser capaz de se conectar a uma rede de comunicação, como Wi-Fi ou redes móveis, para enviar os dados para análise e tomada de decisões.

Outro componente essencial é o servidor ou serviço de processamento. Ele recebe os dados capturados pelos dispositivos IoT, realiza a decodificação dos códigos QR e executa as ações apropriadas com base nas informações extraídas.

A arquitetura proposta também envolve a integração com serviços em nuvem, onde os dados capturados são armazenados, processados e disponibilizados para acesso remoto. Isso permite a análise posterior dos dados coletados, a geração de relatórios e a aplicação de técnicas de aprendizado de máquina para obter *insights* valiosos.

Além disso, é importante considerar aspectos de segurança na arquitetura proposta. Mecanismos de autenticação, criptografia e controle de acesso devem ser implementados para garantir a integridade e a privacidade dos dados transmitidos e armazenados. Também é fundamental garantir a proteção contra possíveis ataques cibernéticos ou manipulação indevida dos *tokens*.

Além da comunicação, tornou-se necessário o desenvolvimento de um MVP PWA, responsável pela interação dos usuários com a solução.

O aplicativo PWA é desenvolvido como uma interface de usuário acessível via navegador web em dispositivos móveis e *desktops*. Ele permite que os usuários gerem os *tokens* QR, inserindo as informações relevantes como: nome do visitante, data que a visita irá ocorrer e número da vaga de estacionamento - se for o caso. O aplicativo PWA oferece uma interface intuitiva e fácil de usar, tornando o processo de

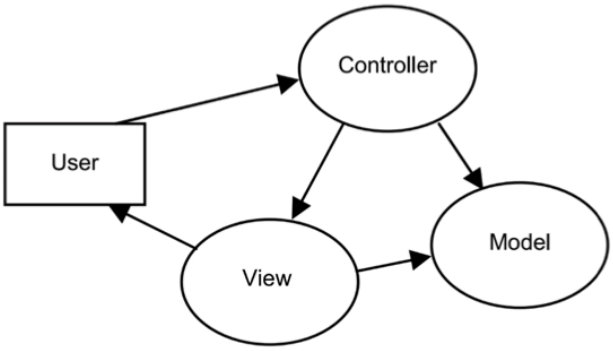
geração de tokens de acesso para visitantes simples e acessível para os usuários finais. Além disso, fornece também toda uma gestão de controle de visitas e permissões a demais residentes da mesma unidade.

Em suma, a arquitetura proposta para o projeto no contexto IoT é projetada para fornecer uma solução eficiente e escalável. Ela envolve dispositivos IoT para a captura de *tokens* QR, um servidor com integração em nuvem, um aplicativo PWA e medidas de segurança robustas. Essa arquitetura visa fornecer uma solução confiável e automatizada para a leitura e o processamento de tokens de visitas, abrindo possibilidades para diversas aplicações no campo do IoT.

3.1 MODEL-VIEW-CONTROLLER

No projeto em questão, é utilizada a arquitetura modular MVC para lidar com a interface do usuário e o fluxo de dados. Quando um usuário interage com a *Graphical User Interface* (GUI[14]), ele envia uma solicitação para um controlador. A Figura 12 mostra a representação do fluxo da arquitetura modular.

Figura 12 - Diagrama MVC



Fonte: Sunardi et al. (2019)

> "[...] a arquitetura MVC foi originalmente introduzida para interfaces de usuário em aplicativos que foram implementados com *Smalltalk*[15] (Linguagem de Programação). Nesta abordagem, o sistema é dividido em três componentes, nomeadamente o *Model* que revela a área Lógica, a *View* que apresenta a interface do utilizador e o *Controller* que gere as alterações da *View*." (Sunardi et al. 2019, p. 135, tradução nossa).

[14] GUI: é uma interface de usuário que utiliza elementos gráficos, como janelas, botões, menus e ícones, para permitir a interação entre o usuário e um software ou sistema.

[15] Smalltalk: ou smalltalk-80, é uma linguagem de programação orientada a objetos desenvolvida na década de 1970 no laboratório de pesquisa da Xerox PARC. Ela influenciou significativamente o desenvolvimento de outras linguagens de programação orientadas a objetos, como Java e Ruby.

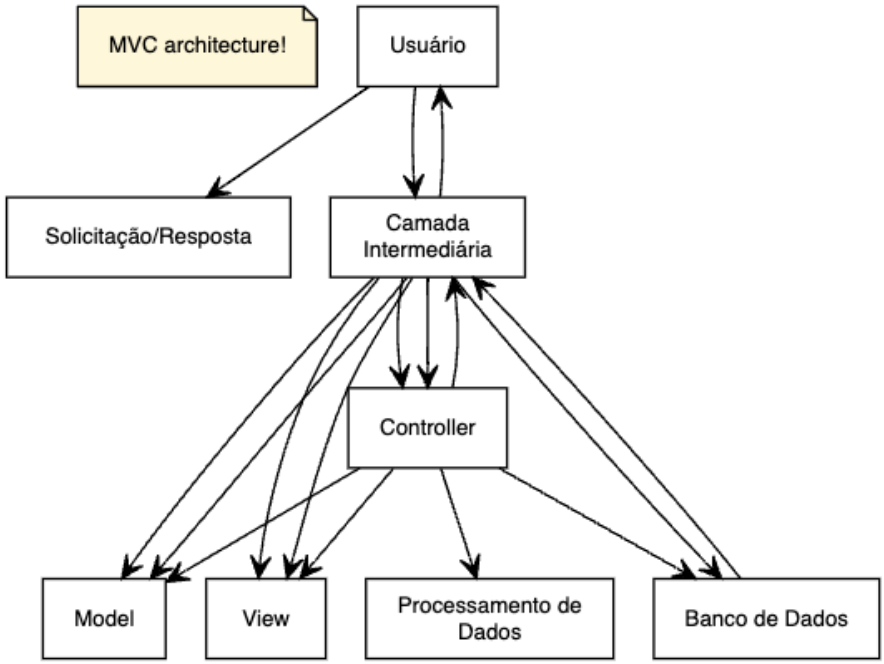
O controlador recebe a solicitação do usuário e atua como intermediário entre a visualização e o modelo. Ele acessa o componente de modelo apropriado para recuperar os dados necessários com base na solicitação do usuário. O modelo representa os dados e a lógica de negócios do aplicativo.

Após recuperar os dados, o modelo os retorna para o controlador. O controlador, então, determina a visualização adequada para apresentar os dados ao usuário. A visualização é responsável por exibir os dados e fornecer uma representação visual das informações.

Por meio desse processo, a arquitetura MVC garante uma separação de preocupações. A visualização é responsável pela camada de apresentação, o controlador lida com a lógica do aplicativo e a entrada do usuário, e o modelo gerencia os dados e as regras de negócios. Essa separação permite flexibilidade, modularidade e facilita a manutenção do aplicativo.

Observe a seguir na Figura 13 cujo é apresentado o diagrama que representa a interação entre as diferentes camadas da arquitetura MVC:

Figura 13 - Diagrama MVC da Solução



Fonte: Elaborado pelo autor (2023)

O diagrama representa a arquitetura MVC com três camadas distintas: a camada do Usuário, a camada Intermediária e a camada do Banco de Dados.

A camada do Usuário representa os usuários da aplicação. Eles interagem com a camada Intermediária enviando solicitações e recebendo respostas.

A camada Intermediária é composta pelos componentes *Model*, *View* e *Controller*. Esses componentes se comunicam e processam dados entre si para fornecer a funcionalidade da aplicação.

O *Model* representa a lógica de negócio e manipulação de dados da aplicação. Ele interage com a camada do Banco de Dados para armazenar e recuperar dados permanentemente.

A *View* é responsável pela apresentação dos dados aos usuários. Ela exibe as informações e interage com o *Controller* para atualizar a interface do usuário conforme necessário.

O *Controller* atua como intermediário entre o Model e a *View*. Ele recebe as solicitações dos usuários, processa os dados necessários e atualiza o Model e a *View* conforme necessário. O *Controller* realiza o processamento de dados e coordena a lógica da aplicação.

A camada do Banco de Dados armazena os dados permanentemente. Ela é acessada pelo Model para armazenar e recuperar informações conforme necessário.

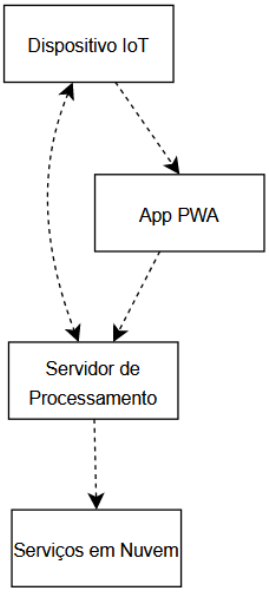
3.2 ESTRUTURA DA SOLUÇÃO

No projeto a estrutura da solução segue o modelo de comunicação device-to-cloud, onde os dispositivos IoT enviam as informações capturadas dos tokens diretamente para um serviço em nuvem que realiza a decodificação, processamento e armazenamento dos dados.

3.2.1 Visão Geral da Arquitetura

Na arquitetura proposta - Figura 14, temos três componentes principais: Dispositivo IoT, Servidor de Processamento e Aplicativo PWA. Observe o diagrama a seguir:

Figura 14 - Diagrama de visão geral da arquitetura



Fonte: Elaborado pelo autor (2023)

O Dispositivo IoT representa o dispositivo responsável por capturar os tokens em formato QR *code* usando sua câmera embutida. Ele se conecta ao Servidor de Processamento para transmitir os dados capturados.

O Servidor de Processamento cria *tokens*, realizar consultas e recebe os dados do token do Dispositivo IoT e realiza a decodificação e validação do token. Em seguida, se integra com um Banco de Dados na nuvem para armazenar informações extraídas do token em caso de ser válido. Feito isso, o Servidor de Processamento fornece um retorno ao IoT após a conclusão do processamento do token por meio de uma comunicação bidirecional, liberando ou não a entrada do visitante/morador.

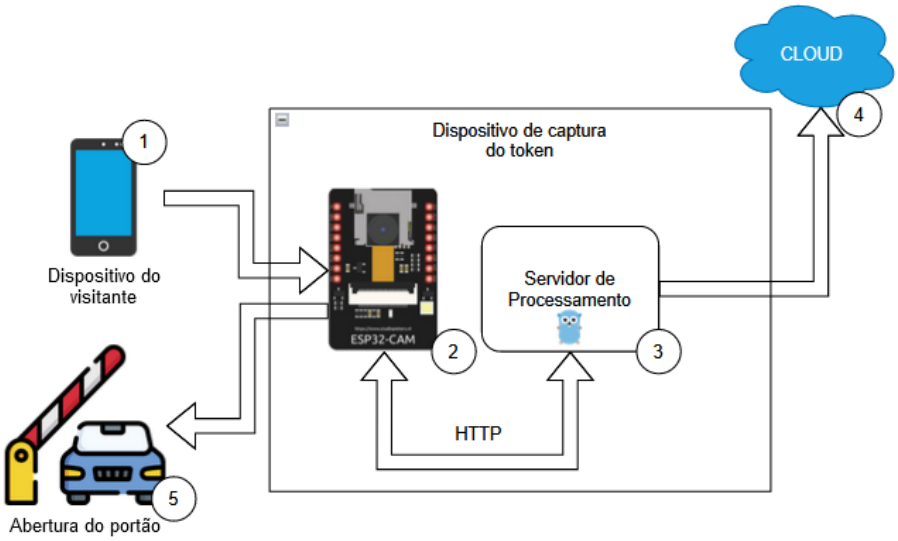
O Aplicativo PWA é responsável por gerar os tokens de acesso no formato de QR *code*. Os usuários podem acessar o aplicativo por meio de um navegador web e criar seus tokens com base em suas necessidades. O Aplicativo PWA se conecta ao Servidor de Processamento para transmitir os dados dos tokens gerados.

Essa arquitetura permite que os tokens sejam gerados no Aplicativo PWA e posteriormente lidos pelo Dispositivo IoT. Os dados capturados são enviados ao Servidor de Processamento, que realiza o processamento necessário, como decodificação, validação e armazenamento.

3.2.2 Dispositivo de Captura da Solução

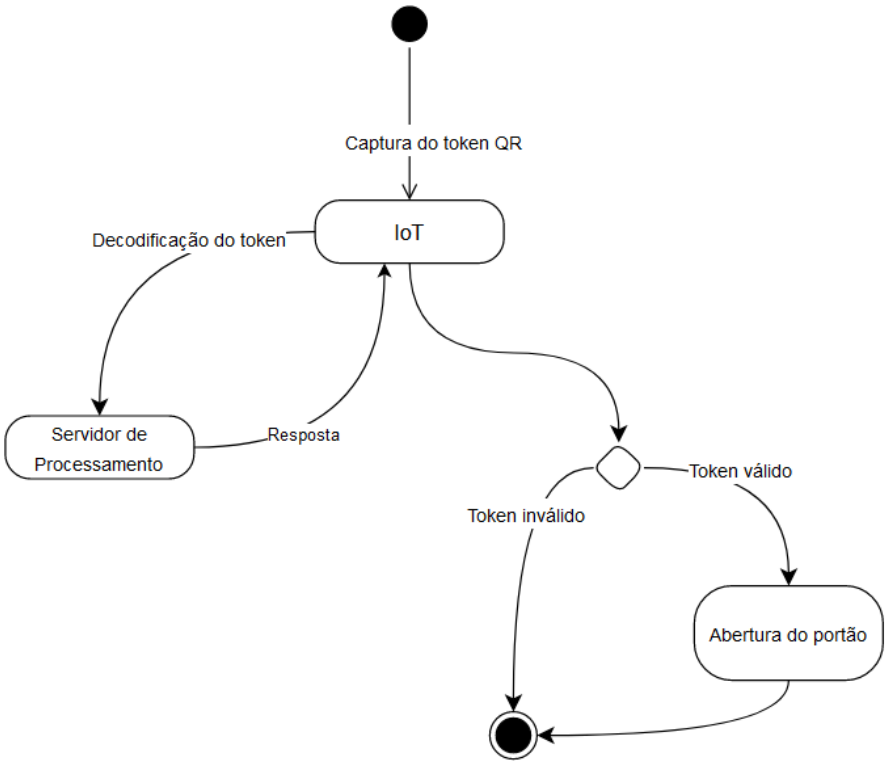
O ESP32CAM é um componente importante na arquitetura do projeto, é responsável por coletar os dados necessários para a leitura dos *tokens*, atuando como o ponto inicial do fluxo de informação, capturando o *token* QR do dispositivo do visitante e transmitindo-o para o próximo estágio de processamento. Observe as Figura 15 e Figura 16, que demonstram respectivamente o dispositivo de captura do *token* e o processo existente por trás do dispositivo.

Figura 15 - Dispositivo de captura da solução



Fonte: Elaborado pelo autor (2023)

Figura 16 - Fluxo do processo de leitura do token QR



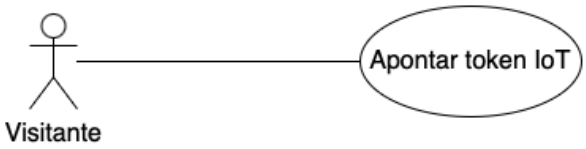
Fonte: Elaborado pelo autor (2023)

Na Figura 16 é demonstrado como ocorre o fluxo do processo mostrado na Figura 15, onde o processo demonstrado se inicia quando visitante aponta a tela de seu dispositivo móvel, mostrando o *token* QR para o dispositivo de captura da solução - esp32cam. Em seguida, o microcontrolador irá enviar uma solicitação de decodificação do *token* para o servidor de processamento que será responsável por realizar o processo de validação do token e enviar uma resposta ao microcontrolador. Em caso do *token* ser válido, será iniciada o processo de abertura do portão, caso contrário o visitante será informado sobre a invalidade do token.

3.2.2.1 Dispositivo do Visitante

O dispositivo do visitante é extremamente simples, não requer nenhuma complexidade nem instalação adicional, já que sua única função é exibir o *token* QR ao microcontrolador recebido de uma aplicação de terceiros.

Figura 17 - Caso de uso do visitante



Fonte: Elaborado pelo autor (2023)

3.2.2.2 Dispositivo de Captura ESP32Cam

Sendo escolhido por sua acessibilidade, flexibilidade e baixo consumo de recursos computacionais e energia comparado a outros microcontroladores existentes no mercado que necessitam de configurações complexas para o seu funcionamento.

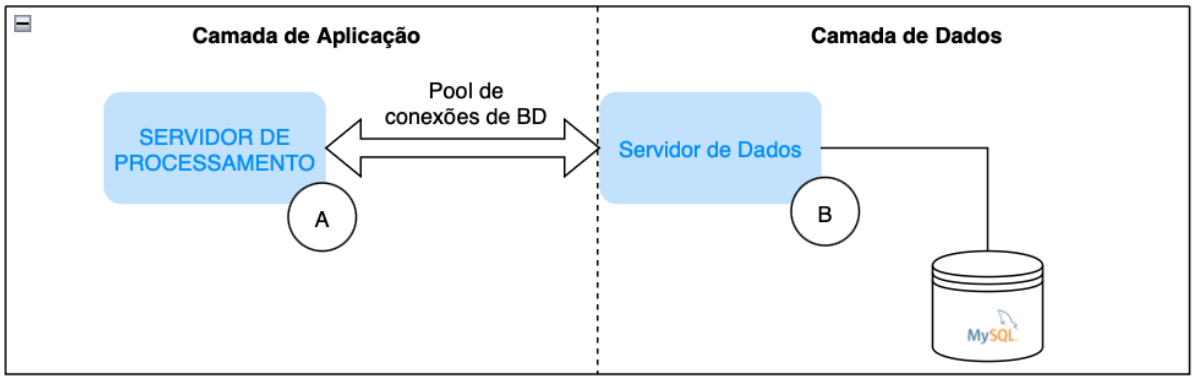
Sua função é ler o token em formato de QR *code* e enviar o código lido para o servidor de processamento (2 Figura 15). Além do envio, ele receberá uma resposta do servidor, e agirá de acordo com esse retorno, se comunicando ou não com dispositivo do portão responsável por sua abertura.

3.2.3 Servidor de Processamento

O servidor de processamento desempenha um papel fundamental. Ele é responsável por receber os dados provenientes do dispositivo IoT, que realiza a leitura dos *tokens* QR, e processá-los de forma adequada. O servidor é responsável por extrair as informações contidas nos *tokens* QR, realizar a validação e executar as ações necessárias com base nos dados obtidos. Veja na Figura 16, o fluxo do Servidor de Processamento (A), localizado na Camada de Aplicação, onde ele realiza um *Pool* de Conexões[16] bidirecionais com o Servidor de Dados (B), localizado na Camada de Dados responsável por armazenar as informações na nuvem.

[16] Pool de conexões: Na engenharia de software, um pool de conexões é um mecanismo que consiste em armazenar em cache conexões de banco de dados, permitindo sua reutilização em solicitações futuras ao banco de dados, quando necessário.

Figura 18 - Arquitetura do servidor de Processamento



Fonte: Elaborado pelo autor (2023)

3.2.4 Modelo de Token QR

No Quadro 1 são descritos os campos contidos no *token* decodificado, onde as colunas representam, respectivamente, a identificação da propriedade e sua descrição.

Quadro 1 - Conteúdo do token da solução proposta

Propriedade	**Descrição**
name	Destinado a identificar o nome do visitante a qual o *token* é destinado.
relationship	Destinado a identificar o grau de relacionamento com a visita, pode ser: Familiar, Amigo, Outros.
date	Data que a visita irá ocorrer.
timeValidate	Indica o tempo de validade do token gerado.
username	Destinado a armazenar o usuário que gerou o token.

Fonte: Elaborado pelo autor (2023)

3.2.5 Cloud e Armazenamento de Dados

A utilização da nuvem desempenha um papel importante no armazenamento e processamento dos dados relacionados aos códigos QR. O servidor de processamento, que pode ser hospedado na nuvem, é responsável por

receber as informações capturadas pelos dispositivos IoT, como o Esp32Cam, realizar o processamento necessário e fornecer uma resposta adequada.

Além disso, a utilização da nuvem permite o armazenamento eficiente e seguro dos dados relacionados aos *tokens*. Os dados capturados pelos dispositivos IoT podem ser enviados para a nuvem e armazenados em um banco de dados ou sistema de armazenamento apropriado. Isso garante a disponibilidade e a integridade dos dados, além de facilitar o acesso e a recuperação das informações quando necessário.

Ao centralizar o processamento e o armazenamento na nuvem, o projeto se beneficia de recursos avançados de segurança oferecidos pelos provedores de nuvem. Isso inclui medidas como criptografia de dados, autenticação de usuários, monitoramento de segurança e *backups* regulares, que ajudam a proteger as informações sensíveis relacionadas aos tokens.

3.2.6 Aplicação Cliente

A solução conta com um aplicativo web cujo qual pode ser acessado de qualquer plataforma, fornecendo flexibilidade ao cliente.

3.2.6.1 Aplicativo PWA

O desenvolvimento de aplicativos móveis tornou-se uma parte essencial do mundo digital. Com o avanço da tecnologia, os usuários esperam ter acesso rápido e fácil aos serviços e informações por meio de seus dispositivos móveis. Nesse contexto, o *Flutter*, um *framework* de desenvolvimento de interfaces de usuário da Google, surge como uma solução poderosa para a criação de aplicativos modernos e eficientes.

Neste projeto, foi desenvolvido um aplicativo PWA utilizando o *framework Flutter*. O PWA combina a acessibilidade da web e a experiência do usuário semelhante à de um aplicativo nativo, proporcionando uma experiência aprimorada para os usuários em diferentes plataformas.

O *Flutter* é uma escolha ideal para o desenvolvimento de PWAs, pois permite a criação de interfaces de usuário atraentes, interativas e responsivas. Com

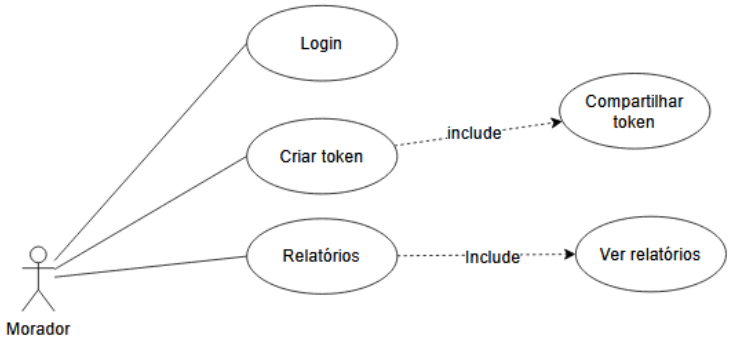
sua abordagem de desenvolvimento de código único, é possível criar aplicativos para diversas plataformas, como *Android*, iOS e *web*, com uma única base de código.

Este aplicativo PWA desenvolvido com *Flutter* visa oferecer uma experiência fluida e consistente aos usuários, independentemente do dispositivo que estão usando. Ao implementar as melhores práticas do *Flutter*, como widgets personalizáveis, animações suaves e desempenho otimizado, busca-se fornecer uma interface agradável e intuitiva.

Além disso, a natureza progressiva do aplicativo permite que os usuários o acessem diretamente por meio de um navegador da *web*, sem a necessidade de baixar ou instalar nada. Isso proporciona uma maior acessibilidade e facilidade de uso, além de permitir atualizações instantâneas sem a necessidade de reinstalação.

O aplicativo PWA desenvolvido com *Flutter* é uma solução versátil para atender às demandas do mundo mobile, oferecendo uma experiência aprimorada aos usuários e permitindo uma maior flexibilidade no desenvolvimento e distribuição de aplicativos. Com seu conjunto de recursos e funcionalidades avançadas, ele se torna uma poderosa ferramenta para a criação de soluções móveis modernas e eficientes.

Figura 19 - Caso de uso do morador



Fonte: Elaborado pelo autor (2023)

3.3 SEGURANÇA

De acordo com (FONTES 2010), a Segurança da Informação é um conceito fundamental que visa proteger um conjunto de informações e preservar o valor que essas informações possuem para indivíduos e organizações. O autor ainda destaca os critérios essenciais da Segurança da Informação:

- *Confidencialidade:* garante que as informações sejam acessíveis apenas por pessoas autorizadas.
- *Integridade:* assegura que as informações permaneçam íntegras e livres de alterações não autorizadas.
- *Disponibilidade:* garante que as informações estejam acessíveis quando necessárias.
- *Autenticidade:* que assegura a identificação correta das partes envolvidas na comunicação e a validade das informações transmitidas.

> “Pesquisas mostram que 55% dos profissionais de TI listam a segurança da IoT como sua principal prioridade, de acordo com uma pesquisa realizada pela 451 Research. De servidores corporativos a armazenamentos em nuvem, os cibercriminosos podem encontrar uma forma de explorar informações em vários pontos de um ecossistema de IoT. Isso não significa que você deve abrir mão de seu tablet de trabalho e trocá-lo por papel e caneta.” (KASPERSKY, [s.d.]).

Para o site HSCBRASIL (2018), um dos principais desafios na garantia da segurança dos sistemas de IoT é a limitação de recursos em alguns dispositivos, o que impede a execução de funções de segurança convencionais. Muitos desses dispositivos foram projetados sem priorizar a segurança, mas sim para oferecer funcionalidade a baixo custo. Isso significa que medidas adicionais precisam ser implementadas para proteger adequadamente esses dispositivos e mitigar os riscos de segurança associados.

3.3.1 Segurança no Projeto

A segurança é um aspecto fundamental neste trabalho, e várias medidas são adotadas para garantir a proteção dos dados e a integridade do sistema. Dentre as tecnologias e recursos utilizados, destacam-se o uso de JWT, o algoritmo *bcrypt* da linguagem de programação Go e os recursos de segurança fornecidos pela *cloud*.

O JWT é uma forma segura de autenticação e autorização na comunicação entre diferentes partes do sistema. Ele consiste em um *token* assinado digitalmente que contém informações sobre o usuário e suas permissões. No projeto, o JWT é utilizado para autenticar e autorizar os usuários e tokens permitindo que apenas usuários autenticados e autorizados acessem as funcionalidades do sistema. Isso ajuda a evitar acesso não autorizado e proteger os dados sensíveis do projeto.

O algoritmo *bcrypt* do *Golang* é utilizado para o armazenamento seguro das senhas dos usuários e realização da codificação das informações contidas no token. O *bcrypt* é um algoritmo de *hash*[17] que incorpora medidas de segurança, como o uso de *salt* [18] e iterações para aumentar a resistência a ataques de força bruta. Essa abordagem garante que os dados dos usuários e tokens sejam armazenados de forma criptografada e protegida contra tentativas de quebra.

Além disso, a utilização dos recursos de segurança fornecidos pela cloud contribui para a proteção do projeto. Os provedores de serviços em nuvem oferecem medidas de segurança avançadas, como criptografia de dados em repouso e em trânsito, autenticação de acesso, controle de acesso baseado em papéis e auditoria de atividades. Essas medidas ajudam a proteger os dados armazenados na nuvem contra acesso não autorizado e garantem a conformidade com regulamentações de segurança.

> "[...] A segurança da informação tem deixado de ser tratada meramente como um assunto técnico da área de informática, e vem sendo considerada uma real necessidade nas empresas e nas instituições, visto que a informação é o bem ativo mais valioso de uma empresa. A segurança passa a ser um requisito estratégico, que interfere na capacidade das organizações de realizarem negócios e no valor de seus produtos no mercado." (DRAGO, 2004, p.1).

A combinação dessas tecnologias e recursos de segurança fortalece a segurança do projeto. O uso do JWT e do algoritmo *bcrypt* do *Golang* protege as informações de autenticação e garante que apenas usuários autorizados possam acessar o sistema. Além disso, o *JWT* realiza um segundo papel importante, agindo como encriptador dos dados contidos no token. Os recursos de segurança da cloud, por sua vez, fornecem uma camada adicional de proteção aos dados armazenados, garantindo a confidencialidade, integridade e disponibilidade das informações.

[17] Hash: refere-se a uma função ou algoritmo que é usado para converter um conjunto de dados em uma sequência de caracteres de comprimento fixo, geralmente em formato hexadecimal.

[18] Salt: um valor aleatório adicionado antes do dado ser hashado.

4 VALIDAÇÃO DA SOLUÇÃO

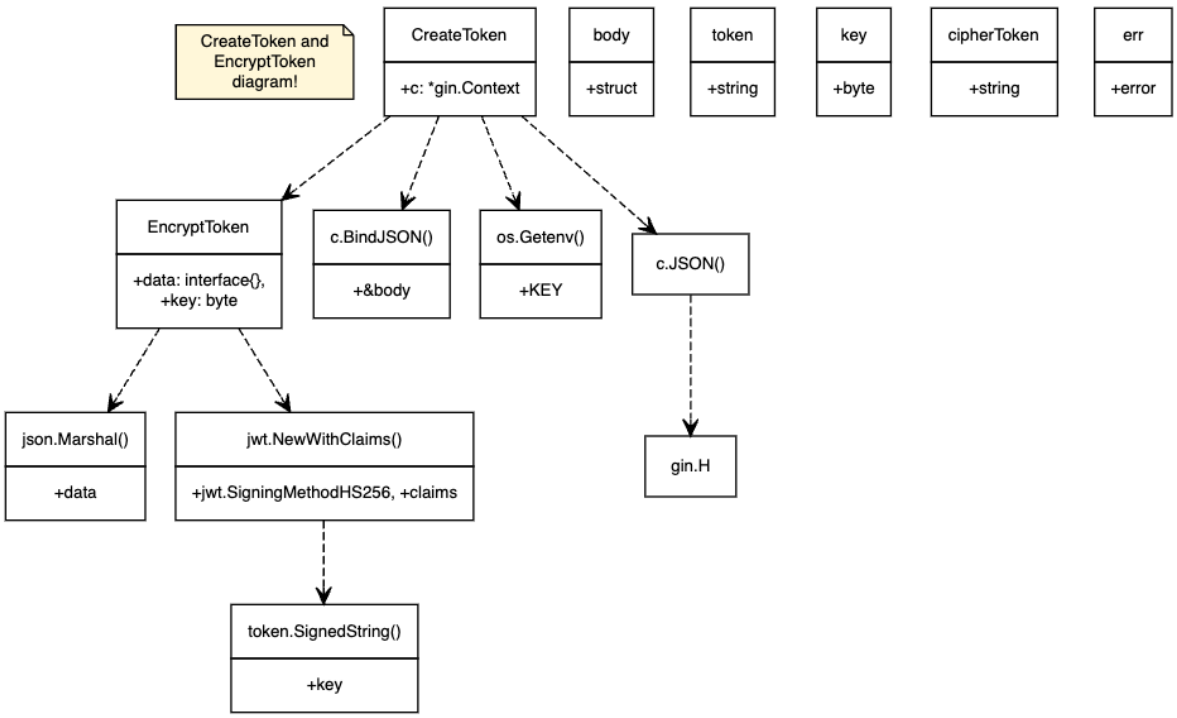
A fim de validar a arquitetura, foi construída uma solução para o controle de entrada e saída de moradores e visitantes de condomínios e residenciais com o intuído de facilitar o acesso destes com o uso da tecnologia sem dispensar a segurança. Através do uso de um microcontrolador, capaz de e capturar informações e se comunicar com um servidor de processamento na cloud, foi possível desenvolver essa solução eficiente e de baixo custo. Durante a pesquisa sobre trabalhos relacionados ao microcontrolador ESP32CAM, foram encontrados alguns trabalhos voltados a detecção de objetivos, leitura de QR Code e vídeo streaming usando servidor web, porém percebeu-se uma carência de solução que pudesse facilitar o acesso de pessoas a condomínios/residenciais, ocasionando em grande número um descontrole de entrada e saída principalmente de terceiros – visitantes.

O objetivo da solução foi facilitar o acesso de pessoas, fornecer segurança e controle de entrada principalmente para os moradores, dispensando em alguns casos uma portaria. Conforme citado no Capítulo 3, os agentes presentes no Dispositivo de Captura e no Servidor de Processamento foram desenvolvidos respectivamente nas linguagens C++ e Golang. O aplicativo web, foi desenvolvido através do framework Flutter usando a linguagem de programação Dart, como descrito no Capítulo 3, embora a arquitetura permita a aplicabilidade de outras tecnologias tanto de front-end quanto de back-end. Os detalhes da solução desenvolvida estão descritos ao longo deste capítulo.

4.1 CRIAÇÃO DO TOKEN

O objetivo do *endpoint* é criar os tokens de acesso baseados em criptografia JWT usando uma *key* pré-definida, em seguida armazenar no banco a string criada para servir de ponto de partida para consultas SQL e por fim retornar a string criada para o usuário. A seguir na Figura 17, encontra-se um diagrama geral do processo de criação de token abordando dois *endpoints*, *CreateToken* e *EncryptToken*.

Figura 17 - Diagrama geral da criação do token

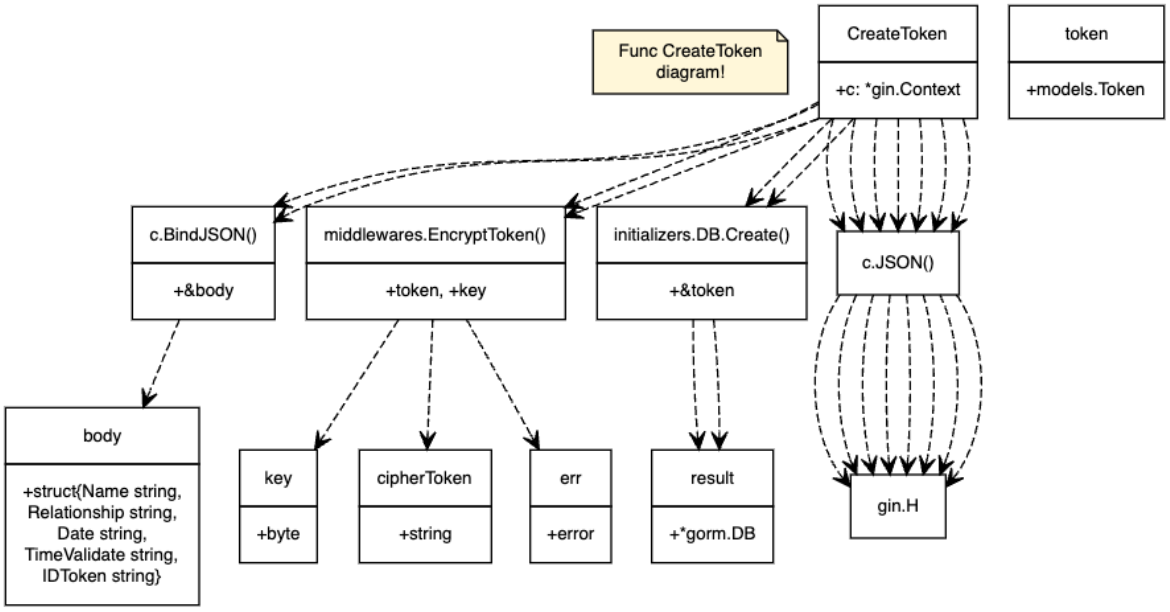


Fonte: Elaborado pelo autor (2023)

4.1.1 Detalhamento da Função CreateToken

A função é responsável por recebe dados JSON na requisição, vincular esses dados a uma estrutura, criar um *token* e armazená-lo no banco de dados, solicitar a criptografia e retornar o token criptografado na resposta.

Figura 18 - Diagrama da função CreateToken



Fonte: Elaborado pelo autor (2023)

1. A função *CreateToken* recebe um parâmetro c do tipo **gin.Context*, que representa o contexto da solicitação HTTP.
2. É declarada uma variável body do tipo *struct* com os campos *Name*, *Relationship*, *Date*, *TimeValidate* e *IDToken*.
3. A função *c.BindJSON()* é chamada para fazer o *bind* [19] dos dados JSON da solicitação para a variável body. Se ocorrer algum erro durante o *bind*, a função retorna um JSON de erro.
4. É criada uma instância do modelo Token com base nos campos da variável body.
5. A chave de criptografia é obtida do ambiente usando *os.Getenv("KEY")* e armazenada na variável *key*.
6. A função *middlewares.EncryptToken()* é chamada passando o objeto token e a chave de criptografia *key* como argumentos. O resultado da função é armazenado nas variáveis *cipherToken* e err.
7. Se ocorrer algum erro durante a criptografia do token, a função retorna um JSON de erro.

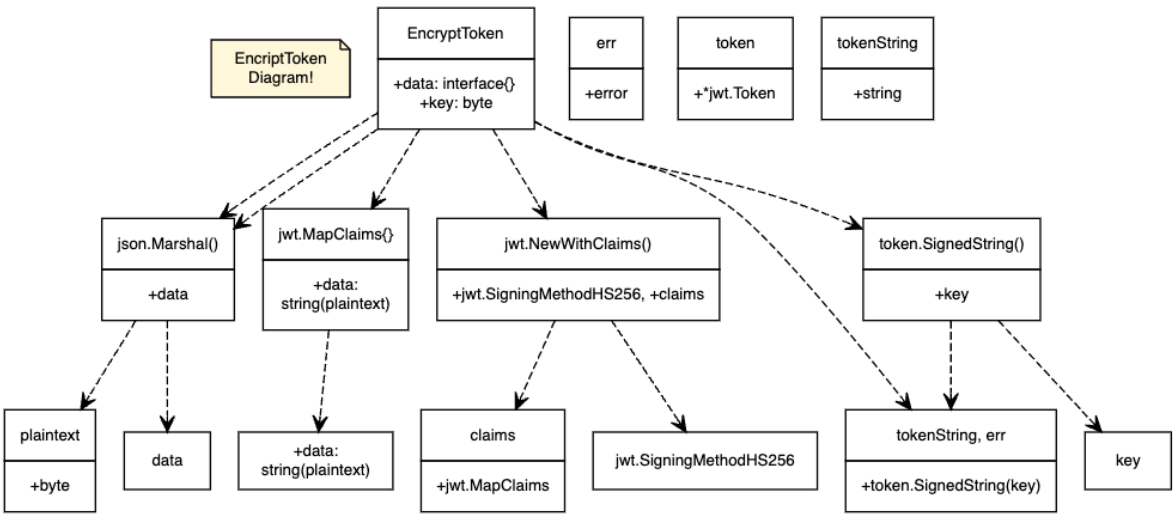
[19] Bind: é um processo de vinculação ou associação dos dados recebidos em uma solicitação HTTP aos parâmetros de uma estrutura ou variável em Go

8. O campo *JWTToken* do objeto token é atualizado com o valor do token criptografado *cipherToken*.
9. A função *initializers.DB.Create()* é chamada para inserir o objeto token no banco de dados. O resultado da operação é armazenado na variável result.
10. Se ocorrer algum erro durante a inserção do token no banco de dados, a função retorna um JSON de erro.
11. A função *c.JSON()* é chamada para enviar a resposta HTTP vazia com status 201, indicando que o token foi criado com sucesso.

4.1.2 Detalhamento da Função EncryptToken

A função *EncryptToken* recebe dados a serem criptografados e uma chave de criptografia. Ela serializa os dados em formato JSON, cria um *token* JWT com as reivindicações adequadas e o assina usando a chave fornecida. O *token* criptografado é retornado como uma *string*, juntamente com um possível erro, se ocorrer algum problema durante o processo.

Figura 19 - Diagrama da função EncryptToken



Fonte: Elaborado pelo autor (2023)

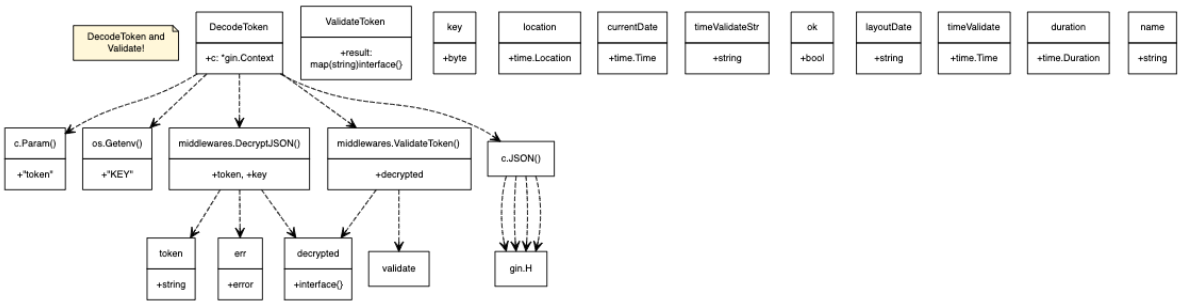
1. A função *EncryptToken* recebe dois parâmetros: data, que é o objeto de dados a ser criptografado, e *key*, que é a chave de criptografia utilizada para assinar o *token* JWT.

2. Primeiro, o objeto data é serializado em *bytes* usando a função *json.Marshal().* Isso transforma o objeto em uma sequência de bytes que pode ser processada pela função de criptografia.
3. Em seguida, é criado um mapa de reivindicações - *claims* do *token* JWT. Neste caso, o mapa de reivindicações contém apenas uma única reivindicação chamada *data*, que armazena a *string* serializada *string(plaintext).*
4. Um novo token JWT é criado utilizando o método de assinatura HS256 - *jwt.SigningMethodHS256* e *claims* fornecidas.
5. O token é assinado com a chave de criptografia fornecida usando o método *token.SignedString(key)*. Isso cria uma *string* assinada do token JWT.
6. A *string* assinada do token JWT, *tokenString*, é retornada como resultado da função, juntamente com um possível erro, err.

4.2 DECODIFICAÇÃO DO TOKEN

O objetivo do *endpoint* a seguir é decodificar os tokens de acesso JWT usando a *key* que já definida na encriptação e em seguida validar a string. A seguir na Figura 20, encontra-se um diagrama geral do processo de criação de token abordando dois *endpoints*, *DecodeToken* e *ValidateToken*.

Figura 20 - Diagrama geral da decodificação do token

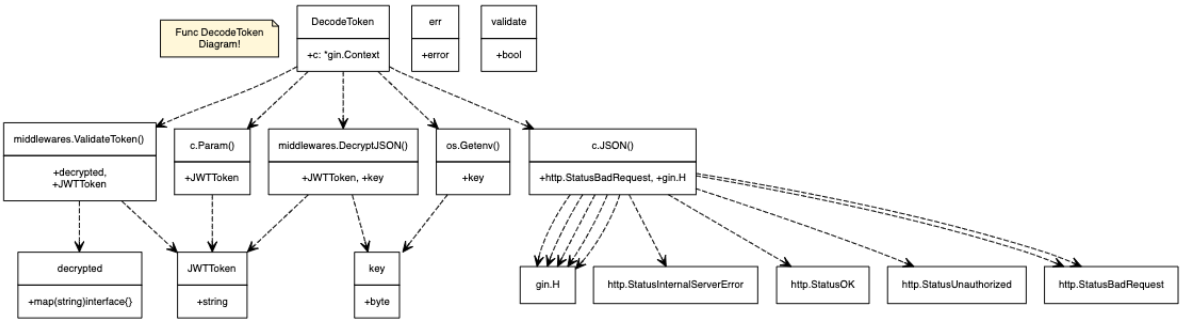


Fonte: Elaborado pelo autor (2023)

4.2.1 Detalhamento da Função DecodeToken

Esta função recebe um token criptografado na URL, descriptografa o token usando uma chave, solicita a validação do token descriptografado e retorna uma resposta JSON com o status apropriado com base no resultado da validação.

Figura 21 - Diagrama da função DecodeToken



Fonte: Elaborado pelo autor (2023)

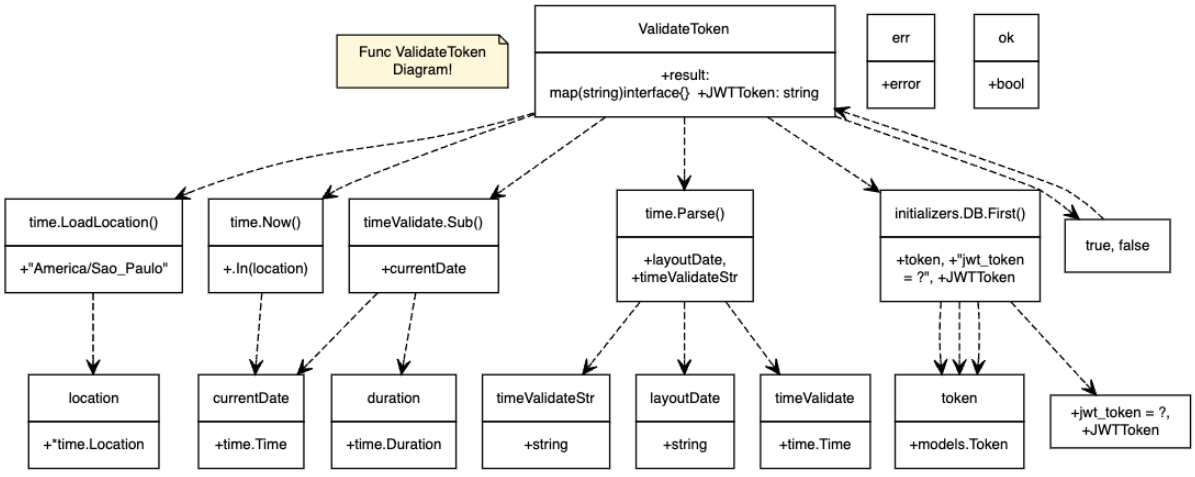
1. A função *DecodeToken* recebe um parâmetro c do tipo **gin.Context*, que representa o contexto da solicitação HTTP.
2. A variável *JWTToken* é declarada para armazenar o token JWT recebido como parâmetro da URL através de *c.Param*("*token*").
3. A chave de criptografia é obtida do ambiente usando *os.Getenv*("KEY") e armazenada na variável *key*.
4. É feita uma verificação do tamanho do token JWT. Se o tamanho for maior que 4, o fluxo continua; caso contrário, a função retorna um JSON de erro indicando um erro de tamanho.
5. A função *middlewares.DecryptJSON*() é chamada passando o token JWT e a chave de criptografia como argumentos. O resultado da função é armazenado nas variáveis *decrypted* e err.
6. Se ocorrer algum erro durante a decodificação do token, a função retorna um JSON de erro indicando uma falha na decodificação.
7. A função *middlewares.ValidateToken()* é chamada passando o token decodificado e o token JWT original como argumentos. O resultado da função é armazenado na variável *validate*.

8. Se ocorrer algum erro durante a validação do token, a função retorna um JSON de erro indicando uma falha na validação.
9. Se a validação do token for bem-sucedida (ou seja, *validate* for verdadeiro), a função retorna uma resposta HTTP vazia com status 200, indicando que o token é válido.
10. Caso contrário, a função retorna uma resposta HTTP vazia com status 401, indicando que o token não é válido.
11. Se o tamanho do token JWT for menor ou igual a 4, a função retorna um JSON de erro indicando um erro de tamanho.

4.2.2 Detalhamento da Função ValidateToken

A função *ValidateToken* verifica a validade de um token com base na data de validação. Ela calcula a duração entre a data atual e a data de validação do token e consulta o banco de dados para confirmar a existência do token. Se o token estiver válido e corresponder a um registro no banco de dados, a função retorna *true*, caso contrário, retorna false.

Figura 22 - Diagrama da função ValidateToken



Fonte: Elaborado pelo autor (2023)

1. A função recebe dois parâmetros: *result*, que é um mapa contendo as informações do token decodificado, e *JWTToken*, que é o token JWT original recebido.

2. Em seguida, é carregado o fuso horário "America/Sao_Paulo" usando a função *time.LoadLocation()*, e o resultado é armazenado na variável *location*. Se ocorrer algum erro durante o carregamento do fuso horário, a função retorna false, indicando falha na validação do token.
3. A função obtém a data e hora atual na localização de fuso horário carregada usando *time.Now()*, e o resultado é armazenado na variável *currentDate*.
4. É verificado se a chave "*timevalidate*" existe no mapa *result* e se seu valor é uma *string* válida. Essa verificação é feita por meio da conversão de tipo *result["timevalidate"].(string)*. Se a chave não existir ou o valor não for uma *string* válida, a função retorna false.
5. É definido o layout da data esperada para o token, utilizando o formato "2006-01-02 15:04:05.99999 -0700 -07". Esse formato especifica a estrutura da data de validação do token.
6. A função realiza o *parsing* da *string* de data de validação - *timeValidateStr* utilizando o layout especificado. Se ocorrer algum erro durante o *parsing*, a função retorna false.
7. A duração do token é calculada através da data de validade- timeValidate e a data atual *currentDate* utilizando o método *timeValidate.Sub(currentDate)*. O resultado é armazenado na variável *duration*. Em caso de não válido, já retorna um status 401.
8. A função faz uma consulta ao banco de dados utilizando *initializers.DB.First()*, buscando um token com o mesmo valor de *JWTToken*. O resultado é armazenado na variável token do tipo *models.Token*.

É verificado se o valor de *JWTToken* é igual ao valor de *token.JWTToken*. Se forem iguais, significa que o token é válido e a função retorna *true*. Caso contrário, retorna false.

4.2.3 Detalhamento da Função GetAllUserTokens

O software cliente desenvolvido consiste em um aplicativo PWA que fornece ao usuário uma interface intuitiva e amigável, a fim de tornar a sua experiencia prazerosa. O aplicativo se comunica com um servidor de processamento por meio de

uma API REST desenvolvida de acordo com as especificações presentes na Subseção 4.1. A seguir é apresentado as telas da aplicação e seus componentes.

1. A função GetAllUserTokens recebe um objeto c do tipo *gin.Context, que é utilizado para lidar com a solicitação HTTP. Em seguida, a função extrai o parâmetro "username" da solicitação usando o método c.Param("username") e o armazena na variável username.
2. Em seguida, é declarada a variável tokens do tipo []models.Token, que será usada para armazenar os tokens encontrados no banco de dados.
3. A função realiza uma consulta no banco de dados usando o objeto initializers.DB, que representa a conexão com o banco de dados. A consulta utiliza o método Where para buscar todos os tokens relacionados ao usuário específico, usando a cláusula "username = ?" e passando o valor da variável username. O resultado da consulta é armazenado na variável tokens.
4. Após a consulta, a função verifica se ocorreu algum erro durante a busca no banco de dados. Caso haja algum erro, a função retorna imediatamente.
5. Se a consulta for bem-sucedida, a função utiliza o método c.JSON para retornar uma resposta HTTP com o status 200 (OK) e um objeto JSON contendo a lista de tokens encontrados, no formato {"tokens": tokens}.
6. Dessa forma, o diagrama ilustra o fluxo da função GetAllUserTokens, mostrando os métodos utilizados, as variáveis envolvidas e os relacionamentos entre eles.

4.2.4 Software Construído

A função *ValidateToken* verifica a validade de um token com base na data de validação. Ela calcula a duração entre a data atual e a data de validação do token e consulta o banco de dados para confirmar a existência do token. Se o token estiver válido e corresponder a um registro no banco de dados, a função retorna *true*, caso contrário, retorna false.

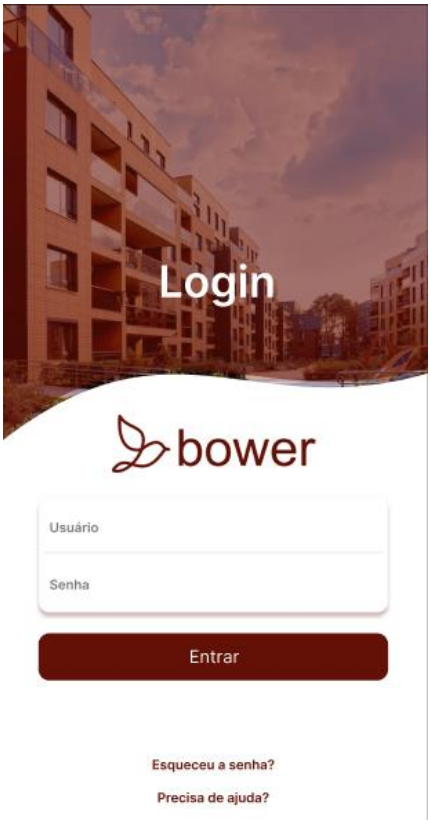
4.2.4.1 Aplicativo PWA

O aplicativo web foi desenvolvido seguindo as tecnologias apresentadas nas Subseções 2.5.1 e 2.5.2. Além disso foi desenvolvido um logo para o app em formato de pássaro, representando a liberdade que os moradores terão. Por fim, o aplicativo foi nomeado de bower e está disponível em bowerapp.com.br.

4.2.4.1.1 Tela de Login

A Figura 23 permite observar como é a tela de login da aplicação. Esta tela permite exigir que os usuários forneçam credenciais válidas, como nome de usuário e senha, é possível autenticar sua identidade. Isso ajuda a proteger as informações e os recursos do aplicativo contra acessos não autorizados.

Figura 23 - Tela de login



Fonte: Elaborado pelo autor (2023)

Esta tela também permite coletar dados e informações valiosas sobre os usuários. Com base nas informações fornecidas durante o processo de login, para obter insights sobre o público-alvo, como sua demografia e comportamentos. Esses dados podem ser usados para melhorar o aplicativo e oferecer recursos adicionais.

A tela de login permite personalizar a experiência do usuário. Ao solicitar o login, pode ser criados perfis individuais para cada usuário, o que possibilita fornecer conteúdo personalizado e funcionalidades específicas. Isso pode melhorar a usabilidade do aplicativo e fornecer uma experiência mais relevante para cada usuário.

4.2.4.1.2 Tela Inicial

Esta tela está organizada em uma arvore de *widgets* e possui uma lista de botões em formato de *card*, sendo uma abordagem popular para exibir informações e funcionalidades de forma visualmente atraente e organizada. Cada card *button*, ao ser tocado, o usuário será redirecionado para a tela em específico, sendo: Criação de *token* e acompanhamento de *token*. Observe a Figura 24 que exibe a interface do usuário:

Figura 24 - Tela de início

Fonte: Elaborado pelo autor (2023)

4.2.4.1.3 Tela de Criação de Token

Intitulada de Sessão de convites, esta tela permite ao usuário criar tokens de acesso aos visitantes fornecendo basicamente três informações: Nome, Grau de parentesco e Data que a visita irá ocorrer. Após o usuário tocar em *convidar*, a tela atual será redesenhada e irá apresentar o layout conforme a Figura 26 demonstra:

Figura 25 - Tela de criação de token de acesso

Fonte: Elaborado pelo autor (2023)

Figura 26 - Tela de criação de token reconstruída

Fonte: Elaborado pelo autor (2023)

Conforme observado na Figura 26, irá aparecer o QR token e o botão de compartilhamento de token. Ao usuário tocar em Compartilha, ele poderá estar compartilhando o token com aplicativos de terceiros, dando maior flexibilidade para o morador e o visitante.

4.2.4.1.4 Tela de Convites Criados

Conforme exposto ao longo deste trabalho, o aplicativo MVP também é capaz de gerar um relatório. Observe a Figura 27, onde também é exibido o horário de entrada e saída da visita. Com base nesses dados, diversas tomadas de decisões poderão ser tomadas, além de trazer um controle mais eficiente para o morador e aumentar a segurança do condomínio.

Figura 27 - Tela de acompanhamento de tokens

Fonte: Elaborado pelo autor (2023)

4.2.4.1.5 Appbar

Note que ao longo de todas as telas temos um widget *appbar* personalizado que contém em seu centro a imagem do morador, em sua parte superior os botões de voltar para a página anterior e *logout*. Toda a *appbar* possui um *wallpaper* para melhor experiência do usuário. Por fim tem-se um card centralizado na *appbar* que indica a tela atual que o usuário se encontra.

Figura 28 - AppBar

Fonte: Elaborado pelo autor (2023)

5 CONCLUSÃO

O presente trabalho de conclusão de curso teve como objetivo principal desenvolver uma solução de Internet das Coisas de baixo custo para o controle eficiente e seguro da entrada e saída de pessoas em condomínios residenciais. Ao longo do projeto, foram abordados conceitos de engenharia de software, arquitetura modular e integração de dispositivos inteligentes visando aprimorar a experiência dos moradores e visitantes, ao mesmo tempo em que se priorizou a segurança condominial.

Durante a pesquisa, foi identificado o potencial da Internet das Coisas como uma abordagem inovadora, conectando dispositivos físicos à Internet e permitindo a troca de informações de forma autônoma. O mercado brasileiro de IoT tem apresentado um crescimento significativo nos últimos anos, impulsionado pelo avanço da conectividade e adoção de tecnologias inteligentes em diversos setores. Essa tendência foi corroborada por estudos e relatórios de consultorias renomadas, que apontam o potencial de crescimento do mercado de IoT no Brasil.

No entanto, foram identificados desafios enfrentados pelas soluções de IoT, tais como a gestão dos dados gerados e a disponibilidade dos dispositivos. Além disso, a interoperabilidade entre dispositivos de diferentes fabricantes e padrões se mostrou uma preocupação importante que afeta a usabilidade e escalabilidade das soluções. Diante desses desafios, foi proposto o desenvolvimento de uma solução de baixo custo que abrangesse essas questões e oferecesse flexibilidade, segurança e integração.

A solução proposta consiste em uma arquitetura modular, onde dispositivos inteligentes, como câmeras e sensores, são integrados a um sistema central. Esse sistema é responsável pelo gerenciamento e controle de acesso em condomínios residenciais, permitindo interações simultâneas de entrada e saída de moradores, além da autorização remota e segura de visitantes. Para facilitar a interação com o sistema, foi desenvolvido um aplicativo PWA multiplataforma, que oferece funcionalidades como a criação de tokens de acesso para visitantes e consulta de tokens.

Ao longo do desenvolvimento do projeto, foram definidos objetivos específicos, como a criação de um modelo de aplicação próprio para a comunicação cliente-servidor, a implementação de uma API REST para a integração entre o

aplicativo cliente e o servidor, e a programação do módulo ESP32Cam para a troca de requisições com o servidor. Esses aspectos técnicos foram fundamentais para garantir a eficiência e segurança da solução.

A solução proposta apresenta diversas vantagens, como a redução de custos de implementação em relação a soluções tradicionais de automação residencial, a flexibilidade para integração de novos dispositivos sem comprometer a acessibilidade financeira e a simplificação do controle de acesso para moradores e visitantes. Além disso, a solução contribui para a segurança condominial, fornecendo um ambiente controlado e monitorado, permitindo que os moradores tenham tranquilidade e confiança no sistema.

5.1 TRABALHOS FUTUROS

O trabalho de conclusão de curso apresentou uma solução de IoT para o controle de acesso em condomínios residenciais, porém, existem várias oportunidades de pesquisa e desenvolvimento que podem ser exploradas no futuro para aprimorar e expandir ainda mais essa solução. Algumas abordagens sugeridas para trabalhos futuros são:

a) Implementação de recursos avançados de segurança: A segurança é uma preocupação central em sistemas de controle de acesso. Portanto, trabalhos futuros podem se concentrar em aprimorar os recursos de segurança da solução proposta, como a utilização de criptografia robusta, autenticação multifatorial e detecção de intrusões. Além disso, a aplicação de técnicas de inteligência artificial e aprendizado de máquina pode ajudar a identificar e prevenir atividades suspeitas.
b) Desenvolvimento de um sistema de monitoramento inteligente: Trabalhos futuros podem se concentrar no desenvolvimento de um sistema de monitoramento inteligente, capaz de analisar dados coletados pelos dispositivos IoT, como câmeras e sensores, para identificar padrões e comportamentos anormais. Isso permitiria uma resposta mais rápida a incidentes de segurança e a implementação de medidas preventivas.
c) Integração com sistemas de gerenciamento de condomínios: A solução proposta neste trabalho se concentra no controle de acesso em condomínios residenciais. Trabalhos futuros podem explorar a integração

dessa solução com sistemas de gerenciamento condominial, permitindo a automatização de processos como reserva de áreas comuns, gestão de pagamentos e comunicação entre moradores.

d) Estudo de viabilidade econômica e escalabilidade: É importante realizar estudos de viabilidade econômica para avaliar o custo-benefício da implementação da solução IoT em diferentes contextos, considerando aspectos como investimento inicial, manutenção e economia de recursos. Além disso, a escalabilidade da solução também deve ser considerada, garantindo que ela possa ser facilmente adaptada e expandida para atender a condomínios de diferentes tamanhos e necessidades.

REFERÊNCIAS

COELHO, Pedro. **Internet das Coisas Introdução Prática**. 1. ed. Lisboa: FCA, 2017. 11 p.

CLOUDFLARE. **O que é UDP?**. Disponível em: https://www.cloudflare.com/pt-br/learning/ddos/glossary/user-datagram-protocol-udp/. Acesso em: 23 mai. 2023.

DRAGO, Isabela. **Segurança da Informação: Estudo Exploratório em Organizações de Grande Porte do Município de Curitiba.** Curitiba. 2004. 53 f. Orientador: Patricia Zeni Marchiori. Trabalho de Conclusão de Curso de Gestão da Informação) – Ciências Sociais Aplicadas, Universidade Federal do Paraná, Paraná, 2004. Disponível em: https://acervodigital.ufpr.br/bitstream/handle/1884/48707/ISABELA-DRAGO.pdf?sequence=1&isAllowed=y. Acesso em: 25 mai. de 2022.

ELECTRORULES. **ESP32 CAM Pinout Guide: GPIOs Usage Explained**. Disponível em: https://www.electrorules.com/esp32-cam-ai-thinker-pinout-guide-gpios-usage-explained/ml. Acesso em: 22 mai. 2023.

ELETROGATE. **RTOS com ESP32: Como Programar Multitarefas**. Disponível em: https://blog.eletrogate.com/rtos-com-esp32-como-programar-multitarefas/. Acesso em: 22 mai. 2023.

ESPRESSIF. **ESP IDF Programming Guide**. Disponível em: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/power_management.html. Acesso em: 20 mai. 2023.

ESPRESSIF. **Hardware Reference**. Disponível em: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/index.html. Acesso em: 22 mai. 2023.

ESPRESSIF. **C++ Support**. Disponível em: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/cplusplus.html. Acesso em: 23 mai. 2023.

FLUTTER. **Flutter architectural overview.** Disponível em: https://docs.flutter.dev/resources/architectural-overview#sidenav-1. Acesso em: 01 jun. 2023.

FLUTTER. **Flutter architectural overview.** Disponível em: https://docs.flutter.dev/resources/architectural-overview#sidenav-1. Acesso em: 02 jun. 2023.

FONTES, Edison. **Segurança da Informação**: o usuário faz a diferença. 1. Ed. São Paulo: Saraiva, 2010.

FORTINET. **What is Transmission Control Protocol TCP/IP**. Disponível em: https://www.fortinet.com/resources/cyberglossary/tcp-ip#:~:text=Transmission%20Control%20Protocol%20(TCP)%20is,data%20and%20messages%20over%20networks. Acesso em: 23 mai. 2023.

GO. **Documentation.** Disponível em: https://go.dev/doc/. Acesso em: 24 mai. 2023.

GOBI, Leonardo. **Projeto E Implementação De Solução Iot Modular Com Contingência** Local. 2019. Nº p./f. Orientador: Alexandre Erasmo Krohn Nascimento. Trabalho de Conclusão de Curso (Bacharel em Ciências da Computação) – Ciências Exatas e Engenharias, Caxias do Sul, Caxias do Sul, 2019. Disponível em: https://repositorio.ucs.br/xmlui/bitstream/handle/11338/6307/TCC%20Leonardo%20Gobi.pdf?sequence=1&isAllowed=y. Acesso em: 21 mai. de 2022.

GOURLEY, D; TOTTY, B **HTTP**: The Definitive Guide. 1. ed. O'Reilly Media, 2002. p. 8.

Gubbi, Jayavardhana, et al. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. **Future Generation Computer Systems**, [s.l.], 2013, v. 29, n. 7, p. 1645-1660, set/2013. Disponível em: https://doi.org/10.1016/j.future.2013.01.010. Acesso em: 20 mai. 2023.

HE, Li, et al. Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing. **IEEE Network**, [s.l.], 2018, v. 32, n. 1, p. 96-101, jan-fev/2018. Disponível em: https://ieeexplore.ieee.org/document/8270639. Acesso em: 20 mai. 2023.

HSCBRASIL. **Segurança para IoT: quais os desafios e seus impactos em segurança.** Disponível em: https://www.hscbrasil.com.br/seguranca-em-iot/. Acesso em: 25 mai. 2023.

IDC. **Future of Industry Ecosystems: Shared Data and Insights**. Disponível em: https://blogs.idc.com/2021/01/06/future-of-industry-ecosystems-shared-data-and-insights//. Acesso em: 20 mai. 2023.

INTELBRAS. **Internet das Coisas: o que é, exemplos e impactos**. Disponível em: https://blog.intelbras.com.br/internet-das-coisas/. Acesso em: 20 mai. 2023.

Jahnavi, TS, et al. Health Monitoring Smart Glove Using Esp32 Microcontrolle. **Journal of Physics: Conference Series**, Reino Unido, 2022, v. 2325, n. 1, p. 1-9, set/2022. Disponível em: https://doi.org/10.1088/1742-6596/2325/1/012007. Acesso em: 21 mai. 2023.

JWT. **What is JSON Web Token?.** Disponível em: https://jwt.io/introduction. Acesso em: 23 mai. 2023.

KASPERSKY. **Internet of Things: what is IoT? IoT Security.** Disponível em: https://www.kaspersky.com.br/resource-center/definitions/what-is-iot. Acesso em: 25 mai. 2023.

Liu, Mingdian, et al. An Iot-Enabled Paper Sensor Platform for Real-Time Analysis of Isothermal Nucleic Acid Amplification Tests. **Biosensors and Bioelectronics**, [s.l.], 2022, v. 169, n. 1, p. [n.p.], dec/2020. Disponível em: https://doi.org/10.1016/j.bios.2020.112651. Acesso em: 22 mai. 2023.

MCKINSEY. **Unlocking the potential of the Internet of Things**. Disponível em: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-internet-of-things-the-value-of-digitizing-the-physical-world. Acesso em: 20 mai. 2023.

MICROSOFT. **C++ language documentation.** Disponível em: https://learn.microsoft.com/pt-br/cpp/cpp/?view=msvc-170. Acesso em: 28 mai. 2023.

MORDOR INTELLIGENCE. **IOT no Mercado de Transporte - Crescimento, Tendências, Impacto do Covid-19 e Previsões (2023 - 2028)**. Disponível em: https://www.mordorintelligence.com/pt/industry-reports/iot-in-transportation-market. Acesso em: 21 mai. 2023.

MOZILLA. **Protocolo**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Glossary/protocol. Acesso em: 25 mai. 2023.

MOZILLA. **An overview of HTTP**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview/. Acesso em: 25 mai. 2023.

MOZILLA. **HTTP headers**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Headers. Acesso em: 25 mai. 2023.

MOZILLA. **HTTP headers.** Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Headers. Acesso em: 28 mai. 2023.

MOZILLA. **HTTP response status codes.** Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status. Acesso em: 28 mai. 2023.

MYSQL. **MySQL Documentation.** Disponível em: https://dev.mysql.com/doc/. Acesso em: 23 mai. 2023.

ORACLE. **O que é IoT?**. Disponível em: https://www.oracle.com/br/internet-of-things/what-is-iot/. Acesso em: 24 mai. 2023.

SALIKHOV, R B, et al. Internet of Things (IoT) Security Alarms on ESP32-CAM. **Journal of Physics: Conference Series**, Reino Unido, 2021, v. 2096, n. 1, p. 1-8, out/2021. Disponível em: https://iopscience.iop.org/article/10.1088/1742-6596/2096/1/012109/meta. Acesso em: 24 mai. 2023.

SUNARDI, A, et al. MVC Architecture: A Comparative Study Between Laravel Framework and Slim Framework in Freelancer Project Monitoring System Web Based. **Procedia Computer Science**, Indonésia, 2019, v. 147, n. 1, p. 134-141, set/2019. Disponível em: https://doi.org/10.1016/j.procs.2019.08.150. Acesso em: 01 jun. 2023.

TOTVS. **Internet das Coisas: o que é, exemplos e impactos**. Disponível em: https://www.totvs.com/blog/inovacoes/aplicacoes-da-internet-das-coisas/. Acesso em: 18 mai. 2023.

TOTVS. **Internet das Coisas: o que é, Exemplos e Impactos**. Disponível em: https://www.totvs.com/blog/inovacoes/aplicacoes-da-internet-das-coisas/#:~:text=Alguns%20exemplos%20de%20aplicações%20comuns,casa%20de%20US%241%20trilhão. Acesso em: 18 mai. 2023.

REDHAT. **Segurança para dispositivos de IoT.** Disponível em: https://www.redhat.com/pt-br/topics/security/security-for-iot-devices#. Acesso em: 19 mai. 2023.

RFC-EDITOR. **Architectural Considerations in Smart Object Networking**. Disponível em: https://www.rfc-editor.org/rfc/rfc7452. Acesso em: 19 mai. 2023.

SUPERTOKENS. **What is a JWT? Understanding JSON Web Tokens.** Disponível em: https://supertokens.com/blog/what-is-jwt. Acesso em: 20 mai. 2023.

WARCONTENT. **{ HTTP Status Code } What is it and what are the HTTP Status Codes?**. Disponível em: https://warcontent.com/status-code-http/. Acesso em: 23 mai. 2023.

WIKIPEDIA. **ESP32**. Disponível em: https://en.wikipedia.org/wiki/ESP32/. Acesso em: 22 mai. 2023

Zanjal, S. V.; Talmale, G. R. Medicine Reminder and Monitoring System for Secure Health Using IO. **Procedia Computer Science**, [s.l.], 2016, v. 78, n. 1, p. 471-476, dec/2016. Disponível em: https://doi.org/10.1016/j.procs.2016.02.090. Acesso em: 21 mai. 2023.

Zis, Alex. **Um Estudo Sobre Protocolos de Autoconfiguração de Endereços para Redes Móveis Ad-Hoc.** Florianópolis/SC. 2006. 80 f. Orientador: Mário Antônio Ribeiro Dantas. Trabalho de Conclusão de Curso de Sistemas da Informação) – Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis/SC, 2006. Disponível em: https://repositorio.ufsc.br/bitstream/handle/123456789/184330/TCC%20Final%20-%20Alex.pdf?sequence=-1&isAllowed=y. Acesso em: 29 mai. de 2022.