MONOGRAFIA DE PROJETO FINAL DE GRADUAÇÃO

RECONHECIMENTO FACIAL AUTOMATIZADO EM UMA ARQUITETURA COM PROCESSAMENTO DISTRIBUÍDO

Heloísa Rodrigues Vargas

Pedro Aguiar Bulhões

Brasília, Fevereiro de 2023

Curso Superior de Engenharia de Redes de Comunicação

DEPARTAMENTO DE ENGENHARIA ELÉTRICA
FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

MONOGRAFIA DE PROJETO FINAL DE GRADUAÇÃO

RECONHECIMENTO FACIAL AUTOMATIZADO EM UMA ARQUITETURA COM PROCESSAMENTO DISTRIBUÍDO

Heloísa Rodrigues Vargas

Pedro Aguiar Bulhões

Monografia de Projeto Final de Graduação submetida ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Bacharel em Engenharia de Redes de Comunicação

Banca Examinadora

Dr. Rafael Timóteo de Sousa Júnior, EnE/UnB _____
Orientador

Dr. Fábio Lúcio Lopes de Mendonça, EnE/UnB _____
Examinador Interno

Ms. Francisco Lopes de Caldas Filho, IFB/Brasília _____
Examinador Externo

FICHA CATALOGRÁFICA

VARGAS, H.R., BULHÕES, P.A.
RECONHECIMENTO FACIAL AUTOMATIZADO EM UMA ARQUITETURA COM PROCESSAMENTO DISTRIBUÍDO [Distrito Federal] 2023.
xvi, 81 p., 210 x 297 mm (ENE/FT/UnB, Bacharel, Engenharia de Redes de Comunicação, 2023).
Monografia de Projeto Final de Graduação - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Fog Computing	2. IoT
3. Machine Learning	4. Biometria
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

VARGAS, H.R., BULHÕES, P.A. (2023). *RECONHECIMENTO FACIAL AUTOMATIZADO EM UMA ARQUITETURA COM PROCESSAMENTO DISTRIBUÍDO*. Monografia de Projeto Final de Graduação, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 81 p.

CESSÃO DE DIREITOS

AUTORES: Heloísa Rodrigues Vargas e Pedro Aguiar Bulhões
TÍTULO: RECONHECIMENTO FACIAL AUTOMATIZADO EM UMA ARQUITETURA COM PROCESSAMENTO DISTRIBUÍDO.
GRAU: Bacharel em Engenharia de Redes de Comunicação ANO: 2023

Heloísa Rodrigues Vargas
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP: 70919-970 - Brasília-DF - Brasil

Pedro Aguiar Bulhões
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP: 70919-970 - Brasília-DF - Brasil

Dedicamos esse trabalho à Deus por nos auxiliar em nossa trajetória.

AGRADECIMENTOS

Eu, Heloísa Rodrigues Vargas, agradeço imensamente todo apoio que recebi dos meus pais, Admilson e Maria de Fátima, a ajuda deles foi essencial para que pudesse chegar até aqui. Também devo um agradecimento especial aos meus irmãos, Marisa e Rubens, por sempre comemorem todas as vitórias e estarem ao meu lado em todos os momentos, bons ou ruins, que tive na graduação.

Agradeço ao meu grande parceiro de vida e profissão, Hítalo Bruno, por toda ajuda e paciência ao longo de todos esses anos, ele tornou toda essa trajetória um pouco mais leve. Agradeço a todos aos amigos que fiz nessa jornada, que me ajudaram no crescimento pessoal e profissional, e todos aqueles que estiveram ao meu lado há pelo menos 10 anos, onde acompanharam minha entrada na UnB e agora a saída como engenheira. Agradeço a todos os familiares que sempre me apoiaram em toda essa trajetória e que levo comigo toda dedicação e força que herdei deles.

Por fim, agradeço a Deus pela força ao longo dessa graduação e por me abençoar com uma família que me apoia em todos os meus passos. Sem essa combinação, não conseguiria.

Eu, Pedro Aguiar Bulhões, agradeço aos meus pais, Arão Fernandes Bulhões e Joana Maria Pinto de Aguiar, por todo o carinho e apoio que eu recebi durante toda a vida, por sempre acreditarem em mim e que foram pais maravilhosos que fizeram tudo ao seu alcance para me ajudar em todos os caminhos que passei, sem eles nada disso seria possível.

Quero agradecer também a minha parceira incrível, Emanuelle Arcângela, por sempre acreditar em mim, me apoiar em todas as minhas decisões e me proporcionar os melhores momentos de alegria. Agradeço a todos os meus grandes amigos de longa data, que foram essenciais nesse caminho, que sempre me acompanharam, me incentivaram e comemoraram minhas vitórias junto comigo, e me fizeram eu me tornar a pessoa que sou hoje. Agradeço a minha dupla de tcc Heloísa Vargas, que foi uma grande amiga desde o começo, sempre me ajudou, e topou finalizar essa jornada comigo. Agradeço aos meus amigos de faculdade, que foram parceiros pra caramba ao longo de todo o curso, vocês tornaram esses últimos anos muito mais tranquilo.

Por fim, obrigado a todos que estiveram comigo e torceram por mim. Tiveram momentos muito bons e outros muito difíceis, mas com certeza o apoio de vocês fez a diferença durante toda essa jornada.

Este trabalho contou com suporte do Laboratório de Tecnologias da Tomada de Decisão da Universidade de Brasília (LATITUDE/UnB), que tem apoio do CNPq - Conselho Nacional de Pesquisa (Outorgas 312180/2019-5 PQ-2 e 465741/2014-2 INCT em Cibersegurança), do Conselho Administrativo de Defesa Econômica (Outorga CADE 08700.000047/2019-14), da Advocacia Geral da União (Outorga AGU 697.935/2019), do Departamento Nacional de Auditoria do SUS (Outorga DENASUS 23106.118410/2020-85), da Procuradoria Geral da Fazenda Nacional (Outorga PGFN 23106.148934/2019-67), da Agência Brasileira de Inteligência (Outorga ABIN 08/2019) e da Universidade de Brasília (Outorga FUB/COPEI 7129).

RESUMO

Como relatado pela reportagem da revista Época Negócios (1), a procura por serviços que automatizam a marcação de ponto pelo funcionário cresceu em 80%, e esse número vem sendo crescente diante da adequação cada vez maior das empresas em ambientes de trabalho híbridos.

Fog computing é um tipo de computação distribuída que estende os serviços de computação em nuvem até a borda da rede, mais perto da fonte de dados. Em contraste com a computação em nuvem tradicional, que depende de centros de dados centralizados, a computação em fog usa uma rede descentralizada de dispositivos e servidores para processar e analisar dados.

O termo "fog"é usado para descrever a camada de recursos de computação que fica entre a nuvem e o usuário final, assim como a névoa fica entre o céu e o solo. Essa camada pode incluir dispositivos como roteadores, *switches* e *gateways*, bem como servidores de ponta e outros dispositivos de computação.

Esse projeto visa implementar um sistema de reconhecimento facial automatizado em uma arquitetura otimizada e de baixo custo, para atender às expectativas de identificação. O trabalho proposto tem como objetivo de aliar conceitos de *fog computing* e *cloud storage* com uma arquitetura utilizando sistemas de *web service* para desenvolver uma aplicação de registro de frequências utilizando visão computacional com auxílio de microprocessadores.

Palavras-chave: fog, nuvem, biometria, visão computacional, inteligência artificial, web server, cadastro

ABSTRACT

As reported by Época Negócios [1] magazine, the demand for services that automate time attendance by the employee grew by 80%, and this number has been growing in view of the increasing adaptation of companies to hybrid work environments.

Fog computing is a type of distributed computing that extends cloud computing services to the edge of the network, closer to the data source. In contrast to traditional cloud computing, which relies on centralized data centers, fog computing uses a decentralized network of devices and servers to process and analyze data.

The term "fog"is used to describe the layer of computing resources that sits between the cloud and the end user, just as fog sits between the sky and the ground. This layer can include devices such as routers, switches, and gateways, as well as edge servers and other computing devices.

This project aims to implement a controlled facial recognition system in an optimized and low-cost architecture, to meet identification expectations. The proposed work aims to combine concepts of fog computing and cloud storage with an architecture using web service systems to develop a frequency recording application using computer vision with the aid of microprocessors.

Keywords: fog, cloud, biometrics, computer vision, artificial intelligence, web server, attendance

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SÍMBOLOS

Siglas

API	*Application Programming Interface*
IoT	*Internet of Things*
CAPEX	*Capital Expenditure*
CLT	*Consolidação das Leis do Trabalho*
CMS	*Content Management System*
CPU	*Central Processing Unit*
DNS	*Domain Name System*
DOS	*Disk Operating System*
DR-DOS	*Digital Research's Disk Operating System*
GCP	*Google Cloud Platform*
GNU	*GNU's Not Unix*
GWS	*Google Web Server*
HTTP	*Hypertext transfer protocol*
HTTPS	*Hypertext transfer protocol* sobre TLS
IaaS	*Infrastructure as a Service*
IIS	*Internet Information Services*
IP	*Internet Protocol*
MS-DOS	*MicroSoft Disk Operating System*
NAT	*Network Address Translation*
NIST	*National Institute of Standards and Technology*
OpenCV	*Optimized Computer Vision*
OPEX	*Operational Expenditure*
PaaS	*Platform as a Service*
PC-DOS	*Personal Computer Disk Operating System*
RAM	*Random Access Memory*
SaaS	*Software as a Service*
SO	Sistema Operacional
SSH	*Secure Shell*
TCP	*Transport Control Protocol*
UDP	*User Datagram Protocol*
VM	*Virtual machine*

1 INTRODUÇÃO

A biometria consiste no estudo estatístico de características físicas ou comportamentais que estabelece a identidade de uma pessoa [11]. Para que essas características possam ser usadas como forma de reconhecimento, as mesmas devem possuir os seguintes requisitos [12]:

- Devem existir em todas as pessoas;
- Devem ser distinta em cada pessoa;
- Não podem variar com tempo;
- Podem ser medida;

O reconhecimento facial (uma forma de biometria) é utilizado para diversas finalidades, sendo uma delas (coincidentemente mais utilizada), a identificação pessoal em atividades públicas e privadas. Vale destaque também para uma outra finalidade que é no uso de controle de acesso, que se refere às medidas de segurança física adotadas para gerenciar e controlar o fluxo de pessoas, veículos e bens físicos num determinado espaço físico, principalmente com o uso de visão computacional [13].

A visão computacional, de acordo com [14], é definida com a ciência responsável pela visão de uma máquina, pela forma como um computador enxerga o meio à sua volta, extraindo informações significativas a partir de imagens capturadas por câmeras de vídeo, sensores, *scanners*, entre outros dispositivos com este fim. Logo, a visão computacional veio com o intuíto de facilitar a resolução de problemas complexos, buscando replicar a cognição e habilidades do ser humano em tomadas de decisões de acordo com informações contidas na imagem.

Ainda sobre, a visão computacional permite o reconhecimento de diversas características que podem ser essenciais dentro do contexto do reconhecemineto e/ou identificação de pessoas e, atrelada ao avanço das tecnologias **IoT** (*Internet of things*), proporcionam o crescimento de muitas áreas de pesquisa e isso claro, está associado ao fato de ter capacidade de coletar e enviar dados com o mínimo de interferência humana, sendo que tais dados são coletados pelas câmeras presentes nos dispositivos, por exemplo, e podem ser usados em aplicações estratégicas, como monitoramento climático e monitoramento interno de edifícios, melhorando assim o gerenciamento do sistema, inclusive dentro do propósito do reconhecimento facial [15].

Um ponto interessante de se destacar a respeito da IoT é que a mesma tem por característica permitir que as pessoas e dispositivos (computacionais) conectem-se sem restrições de hora ou lugar, utilizando um caminho de rede ou serviço que encontrar disponível [16]. Esse benefício se torna bastante útil dentro do contexto da identificação de pessoas, tendo em vista que facilita a implementação e traz simplicidade no gerenciamento do sistema.

Abordado durante a descrição deste trabalho, a tecnologia em *Cloud Computing* surge com um propósito semelhante aos propósitos anteriores, isto é, de colaboração no processo do reconhecimento facial,

já que traz em sua abordagem disponibilidade, um gerenciamento de forma centralizada [17] e se ajusta conforme demandas aos recursos que forem necessários para um sistema de biometria, mas visando um contexto de processamento de uma ou mais câmeras e, com a menor velocidade no envio de dados, a latência e a necessidade de resposta ágil no reconhecimento automatizado passa a ser um fator preocupante.

Nesse contexto, são aplicados os conceitos em *Fog Computing*, que atua como um intermediador entre os dispositivos IoT e a camada em nuvem trabalhando vcomo uma extensão para *Cloud*, isto é, como componentes que se somam [18]. Com a premissa de otimizar o reconhecimento facial, realiza-se um pré-processamento das informações, otimizando a quantidade de transmissão dos dados para a *Cloud*, melhorando de forma significativa a operacionalidade do sistema.

1.1 OBJETIVOS

O presente trabalho apresenta os objetivos gerais e específicos descritos a seguir:

1.1.1 Objetivo Geral

O objetivo geral desse trabalho é desenvolver um sistema de reconhecimento facial com plataformas *web* de acesso à câmeras, que automatize processo de identificação e cadastro de pessoas específicas. Como parte fundamental desse trabalho, tem-se o estudo e implementação de uma estrutura em *Fog Computing* que otimize o processamento local da arquitetura com auxílio da nuvem para transferência de arquivos.

1.1.2 Objetivos específicos

Os objetivos específicos relacionados a esse trabalho são:

- Utilizar dispositivos IoT para realizar a identificação facial por meio de visão computacional.
- Abordar conceitos de uma estrutura em *Fog Computing* para otimizar o processamento e o armazenamento dos dados coletados.
- Desenvolver sistema com processamento local próximo aos usuários;
- Desenvolver uma aplicação *web* onde se possa realizar o cadastro das pessoas que serão reconhecidas no ambiente.
- Analisar os resultados de reconhecimento facial e identificar a assertividade e confiança em diferentes tipos de faces e ambientes.
- Realizar a comunicação do sistema central com o banco de dados em tempo real para armazenamento de fotos e histórico dos usuários;

1.2 JUSTIFICATIVA

De acordo com a o artigo 74, parágrafo 2º do Decreto-Lei nº 5.452, de 1º de maio de 1943, decreto-lei este que regulamenta a CLT (Consolidação das Leis do Trabalho) [19]:

> *Para os estabelecimentos com mais de 20 (vinte) trabalhadores será obrigatória a anotação da hora de entrada e de saída, em registro manual, mecânico ou eletrônico, conforme instruções expedidas pela Secretaria Especial de Previdência e Trabalho do Ministério da Economia, permitida a pré-assinalação do período de repouso.*

Tomando por base um contexto pandêmico de COVID-19, novas modalidades de trabalho remoto bem como a determinação apresentada pelo Decreto-Lei nº 5.452, entende-se que há uma necessidade de se realizar um registro e/ou controle de frequência de forma mais assertiva, confiável e transparente, possibilitando a garantia dos direitos de empregadores e empregados.

Não somente, é possível analisar a presença massiva do registro de ponto por meio eletrônico, como relatado pela reportagem da revista Época Negócios [1]. Nesta reportagem, é destacado que a procura por serviços que automatizam a marcação de ponto pelo funcionário cresceu em 80%, e sendo que esse número tende a seguir uma vertente de crescimento diante a adequação, cada vez maior, das empresas em ambientes de trabalho híbridos.

Por fim, justifica-se este trabalaho fazendo uma análise dos registro automatizados no cenário escolar, tendo uma significativa importância dado que auxilia a gestão e o controle do professor dentro da sala de aula. A implementação da chamada eletrônica, conforme matéria da Unimestre [20], vem trazendo benefícios em escolas onde foi implementada, destacando que a sua operação evitou o desperdício de merenda escolar, já que se tinha de forma automatizada e rápida a contagem de alunos presentes, além de contribuir para diminuir a evasão escolar e faltas de forma significativa.

Dessa forma, com base nos três pontos apresentados anteriormente, o proposto nesse trabalho é implementar um protótipo adaptável de um aplicação *web* que possibilite a automação de cadastro e registro de pessoas (sejam elas profissionais do mercado de trabalho ou alunos) em uma arquitetura distribuída de baixo custo e com o processamento mais próximo o usuário, que além de facilitar e trazer maior confiabilidade no registro para os empregadores, empregados, professores e alunos, faz uso de inteligência artificial.

A mecanização desses processos, com um gerenciador central orquestrado em nuvem e processamento local com o auxílio de nós da arquitetura de *Fog Computing*, pretende auxiliar o controle de seus administradores de forma simplificada e potencializar os benefícios do uso da tecnologia no âmbito escolar e empresarial.

1.3 METODOLOGIA

Esse projeto tem como objetivo implementar e analisar resultados de um sistema de reconhecimento facial, onde é registrado o horário de identificação, em uma arquitetura em baseada em conceitos de *fog computing*. O tipo de pesquisa foi descritivo e exploratório, onde tem-se o estudo de aplicações em uma arquitetura distribuída e verificação de resultados, observando possíveis benefícios e aplicações.

Nesse contexto, a metodologia envolve a construção desse sistema de identificação facial bem como a realização de testes em diferentes ambientes e condições de captura, a fim de explorar cenários e condições de captura que possam gerar falhas de reconhecimento e de efetividade de biometria, utilizando diferentes pessoas em sua execução.

Assim que se obteve a assinatura de **consentimento do uso de imagem e voz** de todos os envolvidos, o procedimento de coleta de dados foi realizado através de câmeras por meio da visão computacional, onde foi analisado de forma qualitativa a performance da aplicação diante da arquitetura otimizada e de baixo custo em relação aos resultados corretos de identificação facial de pessoas cadastradas na aplicação proposta.

No primeiro instante foi construído a aplicação em um computador residencial, realizando os testes de conexão com provedora de nuvem e conexão com outros computadores na mesma rede. Assim que foi obtido resultados satisfatórios, foi hospedado o sistema em um microcomputador, onde foi realizados os testes de efetividade do sistema de fato.

Os dados analisados foram expostos em imagens de forma anonimizada para validação inicial das possibilidades do software, sem constituir uma pesquisa sistemática para efeito de desenvolvimento e operação de um sistema real, onde pode-se perceber de forma visual a identidade reconhecida pelo sistema por meio do número ID, e assim, analisar se a identificação foi feita de forma correta ou não, implicando o sucesso ou não do cadastro de horário.

1.4 HIPÓTESE

Diante da evolução da Internet das Coisas (IoT) e do aumento de dispositivos como smartphones e notebooks, a transmissão de dados entre os dispositivos IoT e sistemas de processamento também acompanharam esse crescimento, podendo criar gargalos no sistema e prejuízos aos usuários, principalmente em sistemas de processamento centralizado, criando pontos únicos de armazenamento criando pontos de falha.

Na implementação de uma arquitetura em *fog computing* espera-se minimizar custos, trazendo a flexibilidade de processamento em hardware de fácil manuseio onde descentraliza o processamento e armazenamento de dados coletados e gerados, sendo uma possível opção para minimizar gargalos e pontos centrais de armazenamento encontrados em arquitetura tradicionais.

1.5 SÍNTESE DE RESULTADOS ALCANÇADOS

Tendo em vista o que foi proposto na seção 1.4, foi possível implementar uma solução utilizando conceitos de *fog computing*, onde o objetivo foi realizar o reconhecimento facial e cadastramento em uma arquitetura onde o processamento e armazenamento dos dados coletados se encontra descentralizada.

A performance da aplicação em termos de efetividade se mostrou positiva, mesmo com alteração de cenários onde implicariam empecilhos em termos de visão computacional, tornando sua aplicação em sistemas de baixo custo como o amplamente citado extremamente viável.

As identidades das pessoas que participaram dos testes foram anonimadas, uma vez que trata-se de uma validação inicial das possibilidades do software, sem constituir uma pesquisa sistemática para efeito de desenvolvimento e operação de um sistema real.

Entretanto, como pode ser observado na seção 6.2, foi amplamente possível validar a assertividade da identidade por meio do registro de número de ID cadastrados únicos sem prejuízos ao projeto e mantendo a ética educacional com relação ao cadastro de indivíduos.

2 TRABALHOS RELACIONADOS

O ramo de identificação através da biometria por meio de *machine learning* usando visão computacional já desencadeou diversas pesquisas focadas em diversas áreas de estudo. Projetos que vinculam *fog computing* com essas tecnologias vêm ganhando bastante notoriedade.

Os trabalhos a seguir apresentam temáticas semelhantes e relacionados com as que serão apresentadas, muitas se relacionam na estrutura proposta, sendo o desenvolvimento de arquitetura com processamento distribuído como os projetos detalhados em:

- 2.0.2 - Sistema para a identificação de aglomerações operando em Redes IoT e *fog computing*;
- 2.0.4 - Sistema de monitoramento seguro por reconhecimento facial baseado em Internet das Coisas;
- 2.0.6 - *IoT based facial recognition security system*;
- 2.0.7 - *A face recognition method in the Internet of Things for security applications in smart homes and cities*;
- 2.0.9 - *Qos scheduling algorithm for a fog iot gateway*;
- 2.0.10 - *Design of a fog controller to provide an iot middleware with hierarchical interaction capability*;
- 2.0.18 -*Gateway* redundante IoT com uso de *fog computing*;
- 2.0.19 - Gerenciamento proativo baseado na análise computacional dos nodos *fog* que integram *smart classroom*;
- 2.0.20 - Sistema inteligente para controle de acesso e monitoramento de múltiplos ambientes (*class control*).

Para a implementar a solução proposta também foi necessária a pesquisa em projetos que relacionam a biometria com a utilização de visão computacional.

Os projetos a seguir utiliza o reconhecimento facial como parâmetro de identificação:

- 2.0.11 - *A New IoT Combined Body Detection of People by Using Computer Vision for Security Application*;
- 2.0.12 - *Edge Artificial Intelligence: Real-Time Noninvasive Technique for Vital Signs of Myocardial Infarction Recognition Using Jetson Nano*;
- 2.0.13 - *A new IoT combined face detection of people by using computer vision for security application*;
- 2.0.14 - Desenvolvimento de uma aplicação web com detecção e reconhecimento facial;

- 2.0.15 -Sistema de reconhecimento e identificação facial em dispositivo embarcado;
- 2.0.16 -Sistema de monitoramento seguro por reconhecimento facial em internet das coisas;
- 2.0.17 - Reconhecimento facial com técnicas de *Machine Learning*.

O projeto a seguir utiliza o andar do(a) indivíduo(a) como parâmetro de identificação:

- 2.0.1 - Identificação de pessoas baseada em características antropométricas e de marcha extraídas de poses 2D.

Por fim, os projetos a seguir trabalham os dois aspectos, o reconhecimento facial com processamento distribuído baseado em *fog computing*.

- 2.0.3 - Segurança do Ambiente Usando Dispositivo IoT com Processamento Distribuído;
- 2.0.5 - *IoT Fog-based Image Matching Monitoring System for Physical Access Control through IP Camera Devices*;
- 2.0.8 - *A cross-age face recognition approach using fog computing architecture for user authentication on mobile devices.*

2.0.1 Identificação de pessoas baseada em características antropométricas e de marcha extraídas de poses 2D

Este trabalho [21] propõe um método de identificação de pessoas baseado na fusão de características antropométricas e de marcha, classificadas como características biométricas *soft* e comportamentais, respectivamente, como forma de aumentar a eficácia em ambientes pouco controlados que não demandam muita interação da pessoa que vai ser identificada.

É aplicado nos *frames* do vídeo de entrada um método de estimativa de pose 2D, como *OpenPose* [22], *PifPaf* [23] ou *TF-Pose-Estimation* [24]. Para as características antropométricas foram destacadas três estratégias: média das medidas obtidas dos esqueletos, concatenação das medidas obtidas dos esqueletos e histograma das medidas obtidas dos esqueletos. Já para as características de marcha foi proposto um método baseado na média dos mapas de calor das articulações dos esqueletos. Além deste método de extração de características de marcha.

O método possui quatro etapas principais: pré-processamento, estimativa de pose, extração de características e reconhecimento do indivíduo, e foi avaliado com base na CASIA Gait Database-A [25] e CASIA Gait Database-B [26], duas bases de dados de acesso público comumente utilizadas em pesquisas sobre identificação biométrica baseada em reconhecimento de padrões de marcha humana, e também em uma base de dados privada, contendo vídeos fornecidos pela Petrobrás.

Diante das validações, o autor destaca que a biometria utilizando características *soft* e de marcha podem se tornar, de fato, bastante efetivas quando se deseja realizar a identificação de uma pessoa em ambiente pouco controlado, sendo que o de marcha tendeu a apresentar resultados melhores que o das medidas

antropométricas. E entre os métodos avaliados, o *PifPaf* apresentou melhor desempenho no contexto apresentado, além do fato de que se for pequenos grupos a identificação se torna mais precisa.

2.0.1.1 Semelhanças Destacadas

O projeto em questão assemelha-se do proposto em seus objetivos, visto que ambos utilizam visão computacional e algoritmos de inteligência artificial para realização de biometria. A maneira que é realizada a identificação difere-se do proposto, visto que além de utilizar algoritmos diferentes, sendo um sistema multi biométrico, o foco é a realização da identificação em um ambiente pouco controlado.

2.0.2 Sistema para a identificação de aglomerações operando em Redes IoT e *fog computing*

Foi desenvolvido [27] um sistema de rede IoT, utilizando *fog computing* para identificação de aglomerações a partir de imagens de câmeras IP e processadas para reconhecimento de padrões e cálculos de distância. O objetivo desse trabalho é realizar o monitoramento de maneira eficiente utilizando uma solução de baixo custo com arquitetura descentralizada, fazendo todo o processamento localmente trazendo economia quanto ao envio e armazenamento dos dados.

Esse projeto aborda muito economicidade em seu escopo, sendo assim um sistema composto por dispositivos inteligentes, ou SBC (*Single Board Computers*), que são responsáveis por processar e identificar as aglomerações nas imagens enviadas pelas câmeras IP, assim como também um computador, que é responsável por receber todo o fluxo de imagens das possíveis aglomerações detectadas pelos dispositivos, validar as imagens e notificar as quebras de distanciamento social.

Com base nos dados coletados, ele recebe os dados e os envia ao computador com um Orquestrador nele instalado, para que se execute a operação correta, de acordo com as regras estabelecidas pelo sistema. Foi utilizado o algoritmo de detecção de aglomerações *Social Distancing AI* [28] com a rede neural YOLOv4, para filtrar essas pessoas fora da área de interesse, foi utilizado o filtro presente em [29].

Foi utilizado o *dataset* [30] para os testes e resultados, que é voltado para o estudo de detecção de aglomerações. E a partir desse foi realizado dois testes: teste de carga e detecção de aglomeração. O primeiro consiste em fazer com que o orquestrador receba dados enviados por vários dispositivos, para entender justamente qual a carga máxima de dispositivos conectados que este sistema suporta e qual a escalabilidade desse sistema. No segundo, o algoritmo procura atingir um número determinado de *frames* que contenham algum tipo de quebra de distanciamento social detectadas em um período de tempo.

Como resultado, os autores expõe que a análise de carga se mostrou eficiente para até mil dispositivos com apenas um agente, e com a arquitetura proposta, a aplicação se tornou resiliente e facilmente escalável.

2.0.2.1 Semelhanças Destacadas

Em semelhança com o projeto proposto, encontra-se a implementação de uma arquitetura baseada em *fog computing*, visto que usam dispositivos semelhantes e um processamento descentralizado para

realização do sistema de aglomeração. Difere-se na aplicação e no propósito do uso dessa arquitetura, visto o objetivo de detectar aglomeração, sem objetivo em realizar biometria.

2.0.3 Segurança do Ambiente Usando Dispositivo IoT com Processamento Distribuído

Esse projeto [15] se trata da implementação de uma solução de identificação facial utilizando dispositivos IoT, em uma arquitetura que se utiliza aspectos de *Cloud* e *fog computing*, com objetivo de otimizar o processamento com um projeto de baixo custo que notifica o usuário final do resultado do reconhecimento.

O processo de identificação foi realizado em três etapas: registro, treinamento e monitoramento. O registro consiste na inclusão de imagens de pessoas feitas por um usuário, esse componente acontece em nuvem, no UIMS [31], que envia ao orquestrador as imagens a serem usadas no treinamento do algoritmo. A etapa de treinamento ocorre quando as imagens são submetidas ao algoritmo de aprendizado de máquina, sendo *Face Recognition* [32], OpenFace [33] e Dlib [34], que auxiliou a modelar o ambiente IoT de detecção de faces usando *fog computing*. E por fim, o monitoramento, em que é realizado a detecção de rosto para detectar a presença de um rosto em uma imagem e, ao detectar uma pessoa, reconhecimento de rosto.

Um dispositivo inteligente, que no caso é a *Raspberry* Pi III, consiste em módulos de detecção, identificação, sincronização e notificação. Sendo assim, utiliza o método chamado Histogramas de Gradientes Orientados (HOG) [35] para detectar a presença de um rosto, onde foram submetidos a algoritmos como marco da face [36], método apresentado em [37], OpenFace [33], SVM (Support Vector Machine) [38], que podem resultar na identificação do usuário ou informar que ele não existe, sendo essa análise enviada por *bot* de mensagem para um usuário final.

Como conclusão, os autores destacam que os resultados obtidos foram altos e eficazes, e o desenvolvimento do modelo foi possível alcançar esses números devido ao treinamento ter sido realizado de forma distribuída.

2.0.3.1 Semelhanças Destacadas

Com escopo bastante parecido com o proposto, utilizando de uma estrutura em *fog computing* e realizando a identificação facial das pessoas do ambiente, esse projeto visa realizar o registro de forma que não tenha interação do usuário ao sistema e não tem controle do horário, diferente do proposto, em que o registro necessita da interação do usuário, e este, tem a visualização dos horários registrados.

2.0.4 Sistema de monitoramento seguro por reconhecimento facial baseado em Internet das Coisas

Este projeto [39] teve como objetivo desenvolver um sistema de monitoramento por reconhecimento facial utilizando dispositivos IoT. Nesse contexto, ele aborda conceitos de segurança tanto dos dados quanto ao acesso da plataforma em si, *edge computing*, já que seu processamento é realizado na própria borda e *cloud computing* para realizar a autenticação dos dados e armazená-los.

O sistema de câmeras fará o constante monitoramento e as imagens poderão ser acessadas em tempo real via *streaming* de vídeo, seu propósito permeia o monitoramento residencial de forma segura, em que o sistema efetue o reconhecimento facial de usuários e dispare alertas. Para o desenvolvimento em si do projeto, foram utilizados: *Raspberry* Pi, câmera, as plataformas de acesso e um algoritmo de reconhecimento facial. Do ponto de vista do usuário, o acesso ao sistema em si poderá ser feito de duas formas: por um *smartphone Android* ou por computador com acesso a internet.

Foram utilizadas várias tecnologias que auxiliaram na composição da arquitetura, um deles foi o *Firebase* que permitiu a utilização de um banco de dados em tempo real, para o armazenamento e controle de dados (vídeos, imagens e dados do usuário), onde é hospedado na nuvem. Assumindo que o sistema tenha internet, a configuração da câmera é realizada e carrega-se os dados dos usuários que se cadastraram através das plataformas web implementadas relacionando-os ao endereço MAC (*Media Access Control*) da *Raspberry* Pi e a partir disso é realizado três tipos de processamento em paralelo: reconhecimento facial de usuários cadastrados no sistema, processamento de comandos recebidos por usuário do sistema e *Streaming* de vídeo e exibição em tempo real de imagens que estão sendo capturadas pela *Raspberry* Pi.

Para o reconhecimento em si, biblioteca de código aberto OpenCV [40] foi utilizada para fazer todas as funções que se relacionam com captura de imagens e vídeo provenientes da RaspiCam. Utiliza-se uma técnica de detecção de rostos [41] em imagens através de um classificador em cascata (*Cascade Classifier*), se reconhece alguém, o reconhecedor LBPH (*Local Binary Patterns Histograms*) é utilizado para fazer a comparação das características do rosto presente no *frame* capturado.

Na análise, ele aborda alguns pontos de segurança que foram implementados que valida seu objetivo, como a requisição de senha de acesso ao sistema e criptografia AES com 256 bits para a transmissão de dados, além do reconhecimento em si que detectou identidades com facilidade, atingindo métricas satisfatórias com transmissão com poucos travamentos e perda de *frames*.

2.0.4.1 Semelhanças Destacadas

O projeto em questão utiliza-se também da visão computacional, utilizando-se de OpenCV, além de implementar uma arquitetura descentralizada. A implementação difere-se no propósito, visto que nesse trabalho o foco é realizar a identificação sem a interação do usuário no intuito de realizar a segurança do ambiente, mas os parâmetros como uso da biometria, envio dos dados para nuvem e utilização de uma interface web para registro e manipulação do usuário se assemelha.

2.0.5 IoT Fog-based Image Matching Monitoring System for Physical Access Control through IP Camera Devices

Esse projeto [42] teve como finalidade implantar um sistema de monitoramento facial, com dispositivos IoT, trabalhando com conceitos em nuvem, para registro e controle de usuários, e com conceitos de *fog computing*, onde o processamento e a análise em si é realizada por dispositivos locais, para uma arquitetura que exige poucos custos.

A arquitetura em si é formada por uma interface de usuário, o orquestrador, módulos de treinamento,

as câmeras de segurança e dispositivos inteligentes. Dentro dessa arquitetura, foi desenvolvido um novo componente, chamado *Camera Coordinator*, ele é um portal que coordena todo o sistema, localizado na nuvem, com propósito de mandar regras para os dispositivos inteligentes por exemplo. Ele além de ser a interface do usuário, onde será registrada as pessoas, ele também será um centro de controle.

Os dispositivos inteligentes são instalados na mesma rede das câmeras e recebem o *streaming* de vídeo que é utilizado na detecção e identificação das pessoas, vale ressaltar que todos têm que estar registrados na *Camera Coordinator*. Ele segue processando os *frames* até que alguém é identificado, onde entra o módulo de identificação que tem implementado com uma biblioteca de reconhecimento facial [43], dependendo do resultado ele manda alertas pro controle central, de acordo com as políticas que foram pre-estabelecidas.

Os testes foram realizados na casa de um pesquisador, devido as condições da pandemia no momento, e foi realizado de três formas: Identificar uma pessoa já registrada dentro do período de horas estabelecido nas políticas, detectar uma pessoa que não foi registrada no sistema e identificar uma pessoa já registrada que não está dentro do horário das políticas estabelecidas. Os autores destacam que o sistema se comportou como esperado, enviando alertas quando necessário e fazendo o reconhecimento de forma ideal.

2.0.5.1 Semelhanças Destacadas

O projeto em questão assemelha-se no uso de conceitos de *fog computing* e *cloud computing* para realização da biometria, o que se assemelha do trabalho proposto. Apesar de existir a relação de registro do horário, ambos projetos possuem propósitos diferentes, já que esse alerta um usuário final quando o horário não se encaixa com o pré-estabelecido em políticas no centro de comando.

2.0.6 IoT based facial recognition security system

Esse projeto [44] implementa um sistema de segurança, com foco residencial, mas aplicável para ambientes mais controlados, com reconhecimento facial, utilizando dispositivos IoT com processamento local. O propósito é ser fácil de construir, barato e efetivo.

O sistema funciona da seguinte forma, no instante que o visitante aperta a campainha, uma câmera tira foto da pessoa, focando em seu rosto, então a foto é processada e enviada para um banco de dados que tenta igualar com algum dado armazenado. Se ele achar algum perfil correspondente, ele envia uma notificação para o morador via e-mail com as informações encontradas e a foto, e então decide se libera ou não a pessoa, caso não seja identificada, ele guarda a foto em um compartimento de desconhecidos. E, caso o dono libere o acesso, a porta abre automaticamente.

Em termos da arquitetura em si, o autor usa uma*Raspberry* como centro do sistema, que controla tanto as ações quanto a performance dos outros dispositivos. A câmera que é um componente importante também, é conectada na *Raspberry*. O algoritmo é construído em *python*, e de novo, implementado na mesma *Raspberry* Pi, que envia as notificações para o morador, quando necessário, com o nome da pessoa identificada no bando de dados, horário da visita entre outros fatores.

O sistema teve êxito em todas as etapas de avaliação de desempenho, sendo essas: processo de treinamento, reconhecimento e liberação da entrada. O autor ressalta a importância dos dispositivos IoT para

integração do sistema e a aplicabilidade não só em ambientes residenciais, mas também em escritórios, aeroportos, industrias etc.

2.0.6.1 Semelhanças Destacadas

O projeto em questão utiliza-se de dispositivos IoT e um centro de dados para realizar o reconhecimento facial de pessoas a partir do gatilho da campainha a fim de automatizar a entrada de pessoas reconhecidas. A arquitetura se assemelha do que foi proposto, com a utilização de processamento local, reconhecimento facial, um banco de dados centralizado, uso de dispositivos IoT e linguagem Python para execução do seu objetivo.

2.0.7 A face recognition method in the Internet of Things for security applications in smart homes and cities

O projeto [45] implementa um método de reconhecimento facial utilizando dispositivos IoT, com processamento local, que realizam a detecção, reconhecimento e alerta em tempo real, com propósito elaborar um sistema de segurança visando residência, escritórios etc.

A *Rapberry Pi* é o centro de todo o sistema, em que a câmera interligada nela tira foto da pessoa, assim que um sensor PIR (*Passive Infrared*) detecta algum movimento, aplicando a visão computacional, detectando e reconhecendo o rosto de uma pessoa. E então, o sistema encaminha a imagem pela Internet para uma aplicação do *Telegram*, que recebe a imagem com a notificação de reconhecimento.

O reconhecimento facial é realizado primeiro com a detecção e localização do rosto, de acordo com recursos do Haar-like [46]. E depois o reconhecimento com o algoritmo LBP [47] que é feito em três estágios: representação facial, extração de informações e classificação. A detecção de rosto e algoritmo de reconhecimento foram mais razoáveis para serem utilizados em tempo real porque eles precisam de menos recursos de CPU e possuem custos baixos.

Nos testes e análises, os autores relatam que todos os processos citados foram realizados da forma esperada sem impeditivos, porém na fase de reconhecimento eles fizeram testes com outros algoritmos, como *Principal Component Analysis* (PCA), *Linear Discriminant Analysis* (LDA), e *Local Binary Patterns Histograms* (LBPH) e perceberam o melhor resultado por meio do algoritmo LBPH diante das variações de iluminação e poses propostas.

2.0.7.1 Semelhanças Destacadas

O projeto em questão utiliza-se de conceitos de *edge computing*, onde todo processamento é realizado na borda por meio de uma *Raspberry*, inclusive o envio de notificações ao usuário. Este se assemelha do proposto por utilizar recursos de inteligência artificial para realização de reconhecimento facial por meio da visão computacional em um dispositivo IoT.

2.0.8 A cross-age face recognition approach using *fog computing* architecture for user authentication on mobile devices

Esse projeto [48] traz uma proposta de reconhecimento facial trabalhando com a possibilidade de mudança de aparência devido a idade, trazendo um *cross-over* referente ao envelhecimento das pessoas dentro de uma arquitetura que faz o processamento inicial distribuído baseado em *fog computing*.

Os autores utilizam o modelo *MobileNets* [49], que é uma família de modelos leves de visão computacional para TensorFlow_lite anunciada pela Google, que traz 16 modelos de pré-treinamento para melhorar o desempenho do reconhecimento de imagem em dispositivos móveis, em conjunto com redes neurais convolucionais (CNN) em um servidor de nuvem distribuído para finalmente determinar o usuário identidade. O uso dos conceitos de *fog computing* vem para responder rapidamente à autenticação do usuário realizando o processamento de dados na borda da rede, perto da fonte dos dados.

A autenticação consiste em realizar a identidade em tempo real com as características faciais dos usuários localmente e verificar detalhadamente as características faciais nos servidores em nuvens remotas. Dentro da autenticação, existe duas fases: a verificação em que é realizado a comparação entre a informação que foi recolhida e a presente no banco de dados, e por fim, a validação, onde é feito os testes classificar a verdadeira identidade de imagens não identificadas em nós de borda com MobileNets.

Na parte de validação, é utilizado uma biblioteca com rostos de usuários rotuladas no CACD [50] e experimentos utilizando MobileNet-V1 e MobileNet-V2 para validação de performance e resultados de acurácia. Os resultados experimentais mostraram que a precisão de reconhecimento do MobileNetV2 usando 2.000 imagens para testar amostras no *Cross-Age Celebrity Dataset* (CACD) foi de 86,12%, além fazer uma dupla validação evitando problemas de fraude de identidade.

2.0.8.1 Semelhanças Destacadas

Esse projeto visa o reconhecimento facial utilizando-se de visão computacional em uma arquitetura com processamento distribuído, escopo semelhante ao proposto. As diferenças baseiam-se nos objetivos, visto que neste projeto a ideia é realizar o reconhecimento facial validando o fator de envelhecimento.

2.0.9 Qos scheduling algorithm for a fog iot gateway

Dentro de uma arquitetura IoT, destaca-se três componentes: os dispositivos, os *gateways* e os *middlewares*. O primeiro se refere os dispositivos finais como sensores, o segundo é responsável por centralizar a comunicação entres esses dispositivos, enquanto o terceiro armazena os dados coletados. Esse projeto [51] tem como objetivo propor um algoritmo que trabalha como um classificador para otimizar as mensagens recebidas pelo *gateway*, estabelecendo critérios definido pelo usuário e priorizar alguns dados que serão enviados para o *middleware*.

Nesse estudo a priorização dos dados pode ser feita quando o dispositivo informa de forma claro que a mensagem é crítica, e então a mensagem é priorizada na fila do *gateway*, e logo enviada para o *middleware*. Uma forma também é a otimização, feita pelos autores, do algoritmo HTB (*Hierarchical Token Bucket*) que

executa uma variedade de técnicas sofisticadas de controle de tráfego que permite um controle complexo e granular sobre esse fluxo. Os autores desenvolveram uma versão modificada desse algoritmo, que cria uma fila para cada dispositivo, onde é possível atribuir pesos e assim estabelecer qual dado tem mais prioridade.

O processamento em si do *gateway* pode ser divido em algumas fases. Na fase de entrada existe o "*Channel Handlers Module*", onde possui vários canais que envia requisições para o *core* do *gateway*, que pode ser diferenciado pelo protocolo de comunicação do dispositivo, por exemplo. Existe um "*Coordinator Module*", que trabalha como buffer para processar todas essas requisições que vem da entrada. E a fase da saída que é responsável por encaminhar as mensagens recebidas do núcleo do *gateway* para o *middleware* de destino.

O protótipo do *gateway* rodou em uma *Raspberry* Pi 3 e os testes foram realizados revezando entre o uso ou não do algoritmo sugerido, em três cenários: utilizando 20 sensores, 60 sensores e depois 120 sensores. Como resultado, mostrou que foi possível proporcionar uma melhor Serviço de QoS para tráfego TCP em detrimento do tráfego UDP, considerando que o TCP foi tratado como maior prioridade.

2.0.9.1 Semelhanças Destacadas

O escopo desse projeto é focado na otimização e análise do tráfego em uma arquitetura em *fog computing*, onde se baseia a semelhança do trabalho proposto, visto que objetivos e implementação se diferem.

2.0.10 Design of a fog controller to provide an iot middleware with hierarchical interaction capability

Esse projeto [52] foca na criação de um *middleware* em uma arquitetura IoT que é capaz de descentralizar responsabilidades, processamento e decisões dentro de uma estrutura proposta, ou seja, abordando parâmetros e fundamentos de *fog computing* para implementação dos componentes dentro da rede.

Dentro da sua composição, ele utiliza SBC's (*computers on a single card*), dispositivos de custo baixo, que são independentes um do outro, mas trabalham de forma otimizada com uma interface minimalista que orquestra toda a estrutura de forma centralizada, enquanto seu processamento e armazenamento são distribuídos dentro da arquitetura.

Os autores destacam os componentes dentro do *middleware*: o *gateway*, responsável pela ponte de comunicação dos dispositivos e do *middleware*; DIMS (*Database Interface Management System*) responsável por armazenar os dados; a interface de dispositivo é responsável pela comunicação do *middleware* com os dispositivos; o módulo de regras permite que o nó tenha capacidade de processamento local, analisando dados que chegam ao *middleware* e executam ações previamente estabelecidas no a coleção de regras DIMS e o controlador que é capaz de criar comunicação assíncrona entre diferentes *Middlewares* UIoT, possibilitando solicitar ou enviar informações entre diferentes nós de rede. Vale ressaltar que o controlador, a interface de dispositivos e o módulo de regras foram desenvolvidos pelos autores.

Para realização dos testes, foi proposto uma topologia com três *middlewares*: um central que representa a faculdade, que trabalha de forma central, e outros dois um departamento, que estão associados com o *gateway*. Para o primeiro cenário, as interações entre *middlewares* e os dispositivos IoT foram avaliados

usando com base regras estabelecidas dentro do ambiente do laboratório. Para o segundo cenário, foi testada a interação entre diferentes *Middlewares* de UIoT, com foco na troca de informações entre nós na hierarquia que foi proposta.

Os autores destacam que o uso de microsserviços para redução da complexidade do ambiente foi um diferencial da arquitetura, além do fato da adição do controlador, que permitiu a comunicação mais coordenada e eficiente entre os *middlewares* propostos. E trouxe alguns pontos de atenção, ao implementa-lo como segurança na comunicação, quantidade de dispositivos, tamanho da rede, etc que pode fazer diferença na efetividade da sua proposta.

2.0.10.1 Semelhanças Destacadas

O trabalho proposto visou implementar um sistema que automatize a troca de informações de dispositivos da arquitetura e realize microsserviços dentro do ambiente, utilizando premissas de uma arquitetura em *fog computing* e dispositivos IoT. Sendo a ideia da arquitetura a semelhança entre os dois projetos implementados.

2.0.11 A New IoT Combined Body Detection of People by Using Computer Vision for Security Application

O trabalho de identificação de corpos humanos por visão computacional combinados com dispositivos IoT do artigo [53] é um estudo que tem como objetivo a identificação de humanos com a ajuda de dispositivos IoT em conjunto com a aplicação do "Telegram" voltado para a segurança de ambientes de acesso restrito. Ele funciona com uma arquitetura centralizada em que o componente central se dá por uma *Raspberry* Pi com conexão à internet via Wifi e conectados a ela existem componentes associados sendo uma câmera e um sensor de movimento.

Para a identificação de humanos a *Raspberry* Pi foi carregada com dois algoritmos de visão computacional. O primeiro sendo voltado para a extração de características e o segundo voltado para a classificação de aprendizagem. Para a primeira parte é utilizado um algoritmo denominado histograma de gradientes orientados (HOG). Ele é responsável pela extração das informações mais importantes dos dados e sua função é a identificação generalizada de um ou mais corpos humanos em uma imagem, que se dá por um conjunto de técnicas de normalização de processamento da imagem, aplicados na imagem de diferentes maneiras como a imagem em alto contraste e depois ela em negativa.

Após o processamento completo da imagem, o resultado é submetido ao algoritmo de máquina de vetores de suporte (SVM) em que ela é treinada a partir desses resultados e por fim terá a função de classificar as identificações como "humanos" e "não humanos" finalizando o trabalho de reconhecimento de humanos.

Dessa forma, a arquitetura do projeto é constituída de um sensor de movimento que quando acionado ativará a câmera para captação de imagens, ambos os componentes conectados a *Raspberry* Pi, e em seguida roda os algoritmos de reconhecimento carregadas na própria *Raspberry*. Uma vez que a identificação de humanos da imagem é um sucesso, um *script* que consome a aplicação para *smartphones* "Telegram"

em que uma notificação a imagem com as identificações são enviadas ao telefone de uma pessoa, para que ela fique ciente de atividades humanas no local da implementação.

2.0.11.1 Semelhanças Destacadas

Parâmetros como uso de visão computacional e dispositivos IoT para realização do reconhecimento facial faz com que ambos trabalhos sejam relacionados, entretanto a maneira que é realizado esse processo torna um fator de diferenciação, visto que neste projeto utiliza-se de conceitos de *edge computing*, já que todo processamento é realizado na borda, além de utilizar de sensor de de movimento para início do processo.

2.0.12 Edge Artificial Intelligence: Real-Time Noninvasive Technique for Vital Signs of Myocardial Infarction Recognition Using Jetson Nano

O artigo [54] de reconhecimento de infarto do miocárdio em tempo real utilizando a Jetson nano tem como objetivo utilizar visão computacional para prever ocorrências de infarto do miocárdio em pacientes que estão em um ambiente controlado. Ele utiliza um algoritmo de *machine learning* chamado YOLO (*You Only Look Once*) para identificar quando pacientes fazem poses de dor no peito, sinal de levine, além de identifica quedas parciais e quedas totais dos pacientes. A arquitetura do projeto tem como objetivo ser descentralizada visando fazer o processamento como *edge computing* numa jetson nano.

Para a identificação de quedas e sinais de infarto nos pacientes, a jetson nano foi carregada com um algoritmo de detecção chamado YOLOv2, porém com a utilização de um *dataset* customizado com fotos e vídeos de pessoas fazendo poses de dores no peito, sinal de levine, quedas parciais e quedas totais. Uma vez que a inteligência artificial é treinada utilizando o *dataset* customizado ela obteve resultados satisfatórios, comprovando sua eficiência.

Dessa forma a arquitetura foi feita com o objetivo de ser mais descentralizado em que a inteligência tivesse seu processamento em *edge computing* através de uma jetson nano. Isso é possível utilizando o modelo *tiny* do YOLO, uma versão otimizada para a inteligência artificial ser mais leve capaz de ser processada em microprocessadores atuais mais potentes. Além do modelo treinado, na placa é acoplada uma câmera que é responsável por enviar imagens em tempo real para o modelo treinado fazer a identificação caso o paciente sofra algum tipo de queda ou apresente dores no peito.

2.0.12.1 Semelhanças Destacadas

Esse projeto traz o uso de algorítimos de inteligência artificial com uso de visão computacional em dispositivos IoT, se assemelhando do projeto proposto, entretanto possui objetivos e arquitetura diferentes, já que foco é prever ocorrências de infarto e utilizar *edge computing* para atingir esse propósito.

2.0.13 A new IoT combined face detection of people by using computer vision for security application

O projeto [55] de detecção de rostos de pessoas usando visão computacional voltado para segurança é um estudo que utiliza visão computacional com o objetivo a identificação de rostos humanos em uma foto com a ajuda de dispositivos IoT em conjunto com a aplicação "Telegram" visando a segurança de ambientes de acesso restrito. O projeto funciona em volta de uma arquitetura centralizada em que o dispositivo central se dá por uma *Raspberry* Pi com conexão à internet e associados a ela existem componentes para auxílio sendo uma câmera e um sensor de movimento.

Para o processamento da imagem, a *Raspberry* Pi foi carregada com um modelo treinado de uma inteligência artificial que é capaz de identificar rostos em imagens utilizando visão computacional. O algoritmo funciona em duas partes, sendo a primeira o posicionamento de quadrados recortados acima da linha dos ombros e pescoço, identificando um possível candidato para rosto humano, já a segunda consiste em testes de algumas características (*"features"*) de rostos humanos para que possa classificado ou não como um rosto humano.

A arquitetura do projeto dos autores consiste em uma arquitetura centralizada em que uma *Raspberry* Pi associada a componentes como sensor de movimento e câmera que auxiliam no processo de aquisição de imagens. Uma vez que o sensor de movimento é acionado ele ativa a câmera para aquisição de imagens em que serão processadas pelo modelo de visão computacional. Uma vez que o processamento é finalizado, a *Raspberry* Pi se comunica com o usuário enviando mensagens através da internet por meio da aplicação "Telegram", nessa mensagem o sistema avisa o usuário se foram detectados algum rosto humano e quantos junto com a imagem com os rostos identificados destacados.

2.0.13.1 Semelhanças Destacadas

O objetivo desse projeto, assim como alguns já expostos, baseia-se em realizar o reconhecimento facial utilizando visão computacional e inteligência artificial e notificar os usuários assim que for realizado a identificação, se assemelhando do proposto, exceto o fato de centralizar todas as ações na borda, caracterizando *edge computing*, além de não ter registro de horário.

2.0.14 Desenvolvimento de uma aplicação web com detecção e reconhecimento facial

Esse projeto [56] visou desenvolver uma aplicação *web* para estudo de métodos de detecção e reconhecimento facial com imagens de câmeras de segurança com objetivo de descrever a implementação da interface e suas características, assim como aplicar os métodos com imagens de câmeras de segurança e analisar os resultados.

Para validar métodos de detecção, que utilizam redes neurais, o autor fez uma análise entre os métodos de Viola-Jones, HOG (*Histogram of Oriented Gradients*), SFD (de Zhang) e o DeepIR, de diferentes autores, utilizando o FDDB (*Face detection data set and benchmark*) que disponibiliza os resultados de testes de detecção de imagem sobre o próprio conjunto de dados. A primeira conclusão que foi feita é que o método de Zhang apresentou o melhor resultado.

Tendo essas opções de detecção facial disponíveis ao implementar a aplicação *web* o autor permite que o administrador escolha o método de detecção e no momento do reconhecimento a aplicação destaca a data, a distância que levaram uma face a ser considerada uma desconhecida e se deu *match* com as pessoas previamente cadastradas no banco de dados do projeto.

Após a implementação, o autor realizou uma análise diante do desempenho dos métodos de detecção e reconhecimento, diante das imagens coletadas nos bancos de dados e de câmeras de segurança. Logo, percebeu-se que o método que utiliza redes neurais convolucionais tem um desempenho de detecção muito acima do método que não utiliza.

2.0.14.1 Semelhanças Destacadas

Esse projeto utiliza-se de algoritmos de reconhecimento facial e inteligência artificial para realizar a identificação de pessoas, além de implementação de uma interface *web*. A diferença do proposto é o fato de ser implementada em uma arquitetura em *fog computing* além dos objetivos estabelecidos.

2.0.15 Sistema de reconhecimento e identificação facial em dispositivo embarcado

Esse trabalho [57] propõe a construção de um sistema de reconhecimento e identificação facial usando um *Raspberry* Pi para processamento das imagens com objetivo de otimizar o processamento com hardware de baixo custo.

O projeto como um todo foi feito em cima da plataforma do tipo *Raspberry* Pi 3 Model B+, já que o processamento das imagens capturadas para a detecção e identificação facial é realizada nele. O autor faz uso do próprio sistema operacional da *Raspberry*, o Raspbian, e utiliza o OpenCV, que é uma biblioteca *open source* popularmente usada para aplicações de visão computacional. Dos recursos disponíveis dentro dessa biblioteca, dois deles são fundamentais para o sucesso do sistema, são eles o algoritmo HaarCascade e o Local Binary Patterns Histograms (LBPH), um para a detecção facial e outro para a identificação facial, respectivamente.

O fluxograma do projeto consiste em primeiro a entrada dos dados, que são as imagens onde queremos detectar e identificar as faces. Caso seja reconhecida a face, ela é submetida a identificação, sendo realizada a comparação das faces reconhecidas com as faces do banco de dados. Feito a identificação é mostrado ao usuário a imagem inserida no sistema com as faces marcadas com uma borda quadrada no seu entorno e o nome da pessoa a qual ela pertence logo abaixo da borda quadrada.

O sistema de detecção e identificação facial foi implementada com sucesso, porém foi destacado alguns pontos de melhorias para melhorar o sistema como um todo. Uma delas, foi a sugestão de implementação de algoritmos que usam sistemas neurais para melhor eficácia, outro ponto abordado foi a velocidade de transmissão dos dados, que foi visto como um problema visto que os dados que trafegam por sinal WiFi gera um *delay* na transmissão e uma perda da qualidade do sinal, o que afetou o reconhecimento.

2.0.15.1 Semelhanças Destacadas

Destaca-se as semelhanças com o projeto proposto, como utilização de dispositivos IoT, uso da biblioteca OpenCV e o algoritmo HaarCascade, porém percebe-se que todo o processamento é realizado na borda, configurando *edge computing*.

2.0.16 Sistema de monitoramento seguro por reconhecimento facial em internet das coisas

O trabalho [39] tem como finalidade criar, através de câmeras e processadores de baixo custo, um sistema que monitora o fluxo de pessoas, permitindo a interação direta do usuário através de plataformas simples, funcionais, responsivas e seguras.

Sendo assim, foi desenvolvido um sistema de segurança com reconhecimento facial, tendo como as principais características o *streaming* de vídeo associado à possibilidade de escolha entre duas plataformas (aplicativo para dispositivos móveis ou navegador web) para visualização, além da possibilidade de movimentação da câmera em dois eixos com liberdade de 180º cada e o sistema de reconhecimento facial em si.

A estrutura física do sistema consiste em uma *Raspberry* Pi 3 modelo B com um software embarcado, uma câmera conectada à interface serial da placa e dois servomotores para o controle de posição da câmera. A estrutura de software é composta pelos algoritmos de reconhecimento facial (algoritmo Viola Jones para detecção e *Local Binary Patterns Histograms* para reconhecimento) e pelo software embarcado (Raspbian Stretch Lite).

O sistema foi implementado com sucesso e seus objetivos dentro do escopo estabelecido foram realizados, um dos pontos destacados pelo autor sobre as dificuldades foi a determinação do algoritmo que melhor se adequasse e trouxesse maior confiabilidade. Outro ponto, foi o uso da *Raspberry* Pi, já que este possui certa limitação em processamento e isso trouxe problemas ao realizar treinamentos dos algoritmos, sendo necessário realizar essas ações externamente.

2.0.16.1 Semelhanças Destacadas

Esse projeto foi implementado com uso de dispositivos IoT, além de possuir uma interface de interação com usuário, se assemelhando com proposto, porém o objetivo desse projeto é monitoramento do fluxo de pessoas, não tendo assim o registro do horário como o projeto proposto. Vale ressaltar que apesar de não ter sido destacado pelo autor, mas o fato de ter realizado o treinamento do algoritmo externamente pode trazer conceitos de *fog computing*.

2.0.17 Reconhecimento facial com técnicas de Machine Learning

O projeto [58] em questão tem como objetivo mostrar a aplicação de técnicas de *machine learning* dentro do âmbito da biometria, em especial, do reconhecimento facial. O autor avalia a assertividade de

diferentes técnicas encapsuladas nos métodos da biblioteca *open source*, o OpenCV. A análise tem como foco avaliar o desempenho de diferentes métricas diante a alterações de iluminação e expressões faciais.

Para o desenvolvimento desse estudo, foi utilizado duas ferramentas: *PyCharm Community Edition*, que proporciona maior produtividade nos projetos em linguagem de programação Python e uma *Web-cam*, integrada ao *notebook*, para realização dos testes. Sendo assim, foram implementados na biblioteca OpenCV os seguintes algoritmos: *Eigenface, Fisherface* e LBPH (*Local Binary Patterns Histograms*) e utilizado o banco de imagens do pacote Yale Face.

Em sua análise, o autor constatou que o melhor desempenho foi obtido com o *Fisherface* e o algoritmo que obteve maior porcentagem no item de confiança foi o LBPH. Sendo assim, dentro do escopo proposto o autor conseguiu chegar em uma conclusão assertiva e atingir todos os objetivos com o trabalho proposto.

2.0.17.1 Semelhanças Destacadas

Apesar desse trabalho ser mais focado em estudo de comparação de algorítimos, é possível destacar a relação entre o projeto proposto devido ao uso da biblioteca OpenCV que auxiliou no desenvolvimento da aplicação.

2.0.18 Gateway redundante IoT com uso de *fog computing*

O projeto [59] em questão trabalhou no desenvolvimento de uma arquitetura baseada em *fog computing* para *gateway* IoT que ofereça alta disponibilidade e que permita uma distribuição da carga de processamento dos dados do ambiente, tendo em vista que a falha do *gateway* pode gerar indisponibilidade e atrapalhar a comunicação de dispositivos no sistema.

Para atuar como o *gateway* do sistema, o autor faz uso de duas unidades do *Raspberry* Pi 3 Model B V1.2 conectado com um pendrive com 8GB para armazenar os bancos de dados com informações dos dispositivos IoT da rede conectadas por uma rede WiFi. Na parte de software, foram utilizados software open source em conjunto para desenvolvimento, sendo: sistema operacional Debian, UIoT Middleware, Keepalived e MongoDB, alguns desenvolvidos pelo próprio departamento na UnB.

Os dois dispositivos trabalham em conjunto, sendo um *Gateway* Mestre e um *Backup*. Em um cenário sem falhas, os dados são recebidos pelo mestre, em caso de falha no mestre o backup assume as conexões com os dispositivos IoT. Foi realizado também testes de performance no ambiente, para o entendimento de até quanto os *gateways* aguentariam o fluxo de dados de forma satisfatória.

O autor retrata que os protótipos cumpriram seus papéis conforme proposto, porém visto limitações em hardware, pode ser implementado melhorias para garantir mais estabilidade ao sistema como um todo.

2.0.18.1 Semelhanças Destacadas

A semelhança entre esse projeto e o proposto é o uso da arquitetura em *fog computing* para implementar seus respectivos objetivos com dispositivos IoT.

2.0.19 Gerenciamento proativo baseado na análise computacional dos nodos fog que integram smart classroom

O projeto [16] em questão tem como o objetivo compor uma *Smart Classroom* com o gerenciamento proativo da *fog computing*, o qual é subsidiado por um modelo adaptativo de análise computacional dos nodos *Fog*.

Nesse projeto foi arquitetado toda uma estrutura focada nas salas de aula, onde o sistema em *fog computing* gerencia o leitor digital, onde ocorre a leitura biométrica para abertura da sala, assim como os dados dos alunos e professores, assim como a manipulação de temperatura, detecção de presença e funcionamento do projetor, sendo assim, quatro sensores com funções distintas dentro da *smart classroom*.

Em cada sensor, existe um processo próprio, como por exemplo o método de autenticação dos alunos e professores, realizado pelo sensor de biometria, nesse passo por exemplo, caso o professor não tenha sido autenticado ainda, o aluno não é autorizado a entrar, entre outras regras. Todo esse processo é processado no nó de *fog computing*, sendo que o banco de dados e o registro posterior de presença ou não do aluno é validado dentro do servidor local, atuando como estrutura de nuvem privada.

Como análise, a autora demonstra como a estrutura em *fog computing* foi importante para métricas de desempenho e latência e de forma simplificada auxiliar o gerenciamento do professor em sala de aula, conseguindo resultados positivos em sua implementação.

2.0.19.1 Semelhanças Destacadas

Esse projeto utiliza de conceitos de *fog computing* para realização de biometria, visando implementar uma *smart classroom*, com intuito de otimizar performance em seu objetivo final, assim como o projeto proposto.

2.0.20 Sistema inteligente para controle de acesso e monitoramento de múltiplos ambientes (*class control*)

Esse projeto [60], baseado em *fog computing*, traz um sistema de controle de acesso e monitoramento de múltiplos sistemas, utilizando uma trava inteligente que é programada a controlar e monitorar os ambientes com mais segurança e eficiência.

O projeto traz a ideia de *"class control"* que pode ser utilizado em salas de aula equipadas com ar-condicionado e projetor. O sistema é responsável por liberar o acesso do usuário ao ambiente, em horário programado, através de sua identificação por cartão RFID, e assim que constatar sua presença, monitorar o uso da energia e climatização do ambiente.

Ele trabalha com dois dispositivos: um microcontrolador com acesso Wifi (ESP8266) e um microcontrolador ARM (*Raspberry*). O primeiro trabalha como a trava inteligente e o segundo é responsável pelo processamento local liberando ou não o acesso e ainda monitorando o ambiente. Também faz uso de um servidor em nuvem, onde ocorre o processamento de banco de dados e um cliente *web* para a operação e configuração do sistema.

O servidor em nuvem tem a função de armazenar as informações das salas que são: usuários cadastrados, registros de acesso ou tentativa de acesso, horários de reserva, registros de alertas e condições do ambiente, onde possui uma aplicação web onde ocorre a orquestração e gerenciamento dessas informações.

Os autores destacam que a implementação foi bem-sucedida em relação ao controle de acesso, conseguindo de forma satisfatória trocar as informações necessárias por meio de comunicação Wifi e gerenciar o funcionamento dos dispositivos IoT nas bordas.

2.0.20.1 Semelhanças Destacadas

O uso de elementos de *fog computing*, como a descentralização do processamento, e a utilização de dispositivos IoT permitiu estabelecer semelhanças entre os projetos, entretanto possuem propósitos de implementação diferentes, visto que esse foca na autenticação e automatização de dispositivos no ambiente.

3 FUNDAMENTAÇÃO TEÓRICA

3.1 INTERNET DAS COISAS (INTERNET OF THINGS - IOT)

O conceito de Internet das coisas faz referência a conexão objetos do dia a dia que conseguem se comunicar com a internet. Isto implica dizer que não só computadores convencionais estão conectados à grande rede, como também uma grande heterogeneidade de equipamentos tais como TVs, *Laptops*, automóveis, *smartphones*, consoles de jogos, *webcams* e a lista aumenta a cada dia [61].

Essa tecnologia passou a ter bastante relevância no mundo atual, não somente em assuntos técnicos, mas também em âmbitos sociais e econômicos. As projeções para o impacto da IoT na Internet e na economia são impressionantes, com previsões de impacto e crescimento de até 83 bilhões de dispositivos IoT conectados até 2024, gerando um superavit econômico de mais de US$ 1,4 trilhões na economia global até 2027, conforme matéria do ConnectData [62].

Essa comunicação de dispositivos totalmente heterogêneos pode ser realizada de dois modos distintos: entre humanos e objetos ou entre objetos em si. A comunicação de humanos e objetos ocorre quando as pessoas interagem com os dispositivos a fim de se obter alguma informação, já a comunicação entre objetos abrange a troca de informações entre eles, podendo existir ou não, a interação direta de pessoas [63].

Além disso, a unidade básica de hardware apresentará ao menos uma das seguintes características: unidade(s) de processamento; unidade(s) de memória; unidade(s) de comunicação e; unidade(s) de sensor(es) ou atuador(es). Aos dispositivos com essas qualidades é dado o nome de objetos inteligentes (*Smart Objects*) [61].

A conexão de objetos a diferentes recursos da internet tem como consequência o surgimento de novas aplicações, como óculos virtuais, relógios inteligentes para monitoramento de gasto calórico, monitoramento de tráfego, chaves eletrônicas, sensores de umidade, além das diversas áreas de atuação que potencializa a heterogeneidade da Internet das coisas.

Diante dessa combinação de tecnologias e aplicabilidade, em [61] os autores apresentam os blocos básicos de construção da IoT sendo eles:

- Identificação: é importantíssimo no mundo altamente conectado a identificação dos objetos de forma única afim de conectá-los a Internet. Tecnologias como RFID e endereçamento IP se aplicam nesse critério.
- Sensores: coletam informações pertinente ao contexto inserido e encaminham ou armazenam essas informações. Podem ter função de atuadores também, onde podem manipular ou reagir aos dados coletados.
- Comunicação: desempenha papel importante na conexão de objetos inteligentes. Tecnologias como Wifi e Bluetooth.
- Computação: unidade de processamento responsável por executar os algoritmos locais.

- Serviços: a IoT pode prover diversas classes de serviço, como serviço de identificação, serviços de colaboração e inteligência, serviço de ubiquidade etc.

- Semântica: trata-se da extração de conhecimentos dos objetos na IoT, de forma que potencialize o uso eficiente dos recursos a partir de dados coletados, a fim de prover determinado serviço.

O uso de tecnologias IoT e sua heterogeneidade permitiu a melhor execução e resultados para elaboração do presente projeto.

3.2 FOG COMPUTING

Com o crescimento constante do uso de tecnologias IoT, que proporciona um aumento substancial no processamento de dados, foi necessário o desenvolvimento de tecnologias que atendessem suas demandas provendo ainda pouca latência. Nesse contexto foi idealizado arquiteturas de *fog computing*, *edge computing* e *mist computing* [2], sendo o conceito de *fog computing* o foco desse trabalho.

Fog computing é um paradigma de recurso horizontal, físico ou virtual que reside entre dispositivos finais e a arquitetura em nuvem tradicional ou centros de dados, como servidores físicos. Esse paradigma oferece suporte a aplicativos sensíveis à latência e isolados verticalmente, fornecendo serviços onipresentes, escaláveis, em camadas e com computação, armazenamento e conectividade de rede distribuídos [2].

Os nós de *fog computing*, que fazem parte dessa estrutura, são elementos de computação intermediários da rede de acesso de dispositivos finais inteligentes que são situados entre a nuvem e os dispositivos finais. Os nós de *fog* podem ser físicos ou virtuais elementos e estão fortemente acoplados com os dispositivos finais inteligentes ou redes de acesso.

Quando se trata de *fog computing*, de acordo com o NIST, é possível destacar as seguintes características:

- Baixa latência: por possuir unidades de processamento mais próximo dos dispositivos IoT, o tempo de análise e resposta dos dados gerados pelos próprios *endpoints* é mais rápida do que se comparado com uma arquitetura de nuvem centralizada.

- Distribuição geográfica: diferente da estrutura da nuvem centralizada, para atender as demandas de dispositivos heterogêneos e distribuídos, os nós de *fog computing* são implantados de forma distribuída.

- Suporte a mobilidade: é essencial que muitas aplicações *fog* se comuniquem diretamente com dispositivos móveis e, portanto, suportam técnicas de mobilidade.

- Interações em tempo real e comunicação sem fio: é ideal que as aplicações em *fog* envolvam interações em tempo real em vez de processamento em lotes, além de priorizar comunicações em fio, visto a maioria dos dispositivos IoT se comunicam dessa forma.

- Heterogeneidade: Os dispositivos em *fog* vêm em diferentes formatos e serão implantados em uma ampla variedade de ambientes, e os dispositivos dos quais eles coletam dados também podem variar em fator de forma e capacidade de comunicação de rede.
- Suporte para análise em tempo real e interação com a nuvem: a arquitetura em *fog computing* está posicionada para exercer um papel significativo na ingestão e processamento dos dados próximos à fonte à medida que são produzidos. Enquanto os nós *fog* fornecem localização, permitindo baixa latência, a nuvem fornece centralização global.

Já em termos de arquitetura, muitos trabalhos tratam de três camadas[64], quatro[65], cinco [18] e até seis [66] para se estudar *fog computing*[67]. Entretanto, de forma padronizada, sempre reporta pelo menos as seguintes camadas, conforme [68]:

1. Camada final: esta é a camada mais próxima do usuário final e do ambiente. Consiste em vários dispositivos IoT, entre eles, sensores, celulares, cartões inteligentes, leitores e assim por diante. Especialmente, embora os esses equipamentos tenham o poder de computação, é apenas utilizado como dispositivos de detecções inteligentes. Eles são responsáveis por processar os dados transmitir esses dados detectados para a camada superior para processamento e armazenamento.
2. Camada Fog: essa é a camada que traz o diferencial dentro da estrutura de *fog computing*. É a camada que fica na ponta da rede, possui a finalidade de realizar o processamento, transmissão e armazenar temporariamente os dados recebidos pela camada inferior, cuja objetivo é reduzir a latência e melhorar a qualidade do serviço prestado. Os dispositivos finais podem se conectar convenientemente com nós de fog para obter serviços. Ela também se mantém conectada com a camada superior, onde terá a interação e cooperação para a necessidade de processamento e armazenamento mais potentes.
3. Camada de nuvem: A camada de computação em nuvem consiste em várias camadas de alto desempenho servidores e dispositivos de armazenamento. Possui recursos de computação e armazenamento para suportar computação, análise e armazenamento de uma enorme quantidade de dados. No entanto, diferente da arquitetura de computação em nuvem tradicional, nem todas as tarefas de computação e armazenamento passam pela nuvem, apenas as solicitações necessárias de armazenamento e *storage* passarão por essa centralização.

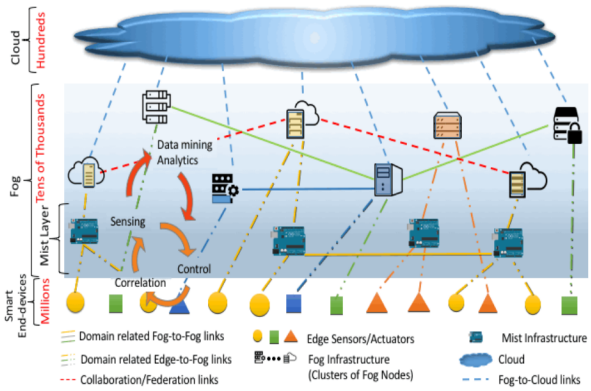

Figura 3.1: Ambiente *Fog comuting* suportando um ecossistema baseado em nuvem para dispositivos finais inteligentes - Fonte: [2]

3.3 CLOUD COMPUTING

Conforme exposto no artigo [17], publicado pela organização NIST, a computação em nuvem é um modelo para permitir o acesso onipresente e sob demanda a uma rede compartilhada conjunto de recursos de computação configuráveis, como armazenamento, aplicativos, serviços e redes, que podem ser provisionados e liberados rapidamente com o mínimo de esforço de gerenciamento ou interação com o provedor de serviços.

O modelo de computação em nuvem tem diversos benefícios econômicos [69], pois libera o consumidor da necessidade de investir em infraestrutura (CAPEX), permitindo que ele alugue recursos de acordo com as necessidades e pagando apenas pelo uso. Além disso, permite diminuir os custos operacionais (OPEX), além da terceirização do serviço de infraestrutura para as nuvens, transferindo o risco do negócio para o provedor de infraestrutura, geralmente mais bem equipado para gerenciá-lo. Além dos benefícios técnicos [69] como eficiência energética, otimização da utilização de recursos de hardware e software, elasticidade, isolamento de desempenho e flexibilidade.

Esse modelo de nuvem é composto por cinco características essenciais, três modelos de serviço e quatro modelos de implantação modelos [17].

Sobre as características essenciais, destaca-se o *"on-demand self-service"* que caracteriza que a requisição de mais recursos seja de forma automatizada a medida que for necessário o consumo; o amplo acesso à rede e o *"pool"* de recursos, ou seja, todos os recursos estão disponibilizados pela rede com o provedor possuindo um *"pool"* de recursos que atende os vários consumidores, sem dar visibilidade de localidade de forma muito precisa ao consumidor; uma rápida elasticidade permitindo aumentar ou diminuir os recursos disponibilizados por demanda e monitoramento, controle e relatório do que foi consumido.

Os três modelos de serviço são: SaaS (*Software as a Service*), PaaS (*Plataform as a Service*) e IaaS (*Infrastructure as a Service*). O SaaS fornece ao consumidor uma aplicação do provedor hospedada em uma infraestrutura da nuvem, o consumidor nesse caso, não gerencia ou controla a infraestrutura de nuvem, incluindo rede, servidores, sistemas operacionais, armazenamento ou até mesmo recursos de aplicativos individuais. Já com o PaaS, o consumidor continua sem gerenciar ou controlar a infraestrutura de nuvem, mas tem controle sobre os aplicativos implantados e possivelmente definições de configuração para o ambiente de hospedagem de aplicativos. Enquanto com o IaaS, o consumidor não gerencia ou controla a nuvem infraestrutura, mas tem controle sobre sistemas operacionais, armazenamento e aplicativos implantados.

E, por fim, os quatro tipos de implantação que diz respeito a forma que vai ser provisionada a infraestrutura de nuvem, sendo a primeira opção a nuvem privada que é para uso exclusivo de uma única organização, a segunda opção a nuvem comunitária que é utilizada por um grupo específico em comum, a terceira, nuvem publica onde é "aberta" ao público e a quarta opção nuvem híbrida que faz proveito de mais de um tipo de provisionamento de nuvem.

3.4 VISÃO COMPUTACIONAL

Em 1982, Ballard e Brown, na obra *Computer Vision*, definiram Visão Computacional como a ciência que estuda e desenvolve tecnologias que permitem que máquinas enxerguem e extraiam características do meio, através de imagens capturadas por diferentes tipos de sensores e dispositivos [70].

[3] Define a visão computacional como sendo o domínio da ciência da computação que estuda e aplica métodos que permitem aos computadores "compreenderem" o conteúdo de uma imagem. Isto pode ser feito quando as informações capazes de caracterizar os elementos que compõem uma cena podem ser padronizadas e extraídas a partir sua representação digital.

O Processamento de imagens e o reconhecimento de padrões são dois campos que estão fortemente relacionados com a visão computacional. Enquanto estudos sobre Processamento de Imagens apresentam técnicas para manipular informações representadas na imagem, os estudos sobre reconhecimento de padrões são realizados a fim de identificar e classificar os objetos representados [3].

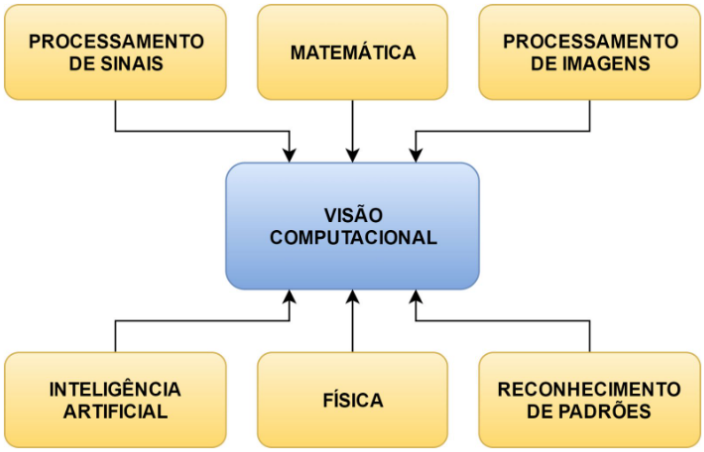

Figura 3.2: Campos de estudo interligados. - Fonte: [3]

Os sistemas de visão computacional geralmente apresentam um fluxo comum, mesmo que esteja presente em diversas áreas [3].

Figura 3.3: Fluxo de um sistema baseado em visão computacional - Fonte: [3]

Na etapa de primeira etapa é quando se adquire e digitaliza uma imagem que será trabalhada e isso pode variar entre imagem bidimensional, tridimensional ou sequencia de imagens (quando se trata de vídeos). Na etapa de pré-processamento ocorre análise da imagem, quando ocorre a identificação e obtenção de

informações necessárias. Já na etapa de segmentação, as informações focos são segmentadas em mais de uma imagem para facilitar a extração de características desses objetos.

Após isso, ocorre a extração de características do objeto, onde o algoritmo foca em informações que podem ser uteis para diferenciação e, por fim, faz o processamento com objetivo de reconhecer e determinar a classe ou grupo que aquela imagem pertence a partir dos padrões reconhecidos.

3.5 WEB SERVER

Um *Web Server* é basicamente responsável por receber solicitações de clientes (como navegadores da *Web*) e responder a elas entregando páginas da *Web* ou outros recursos pela Internet.

Para o entendimento do conceito de *Web Server*, é interessante se ter algumas definições importantes, tais como:

- **Requisição:** Uma requisição ou solicitação é uma mensagem enviada por um cliente (por exemplo, um navegador da *Web*) para um *Web Server*, solicitando um recurso específico (como uma página da Web ou imagem);
- **Resposta:** Uma resposta é uma mensagem enviada por um *Web Server* a um cliente em resposta a uma solicitação. Ele normalmente contém o recurso solicitado, juntamente com outras informações, como cabeçalhos HTTP e códigos de status;
- **HTTP:** O HTTP (*Hypertext Transfer Protocol*) é um protocolo utilizado por *Web Servers* e clientes para se comunicar uns com os outros através da Internet. Ele define o formato de solicitações e respostas, bem como vários códigos de status que indicam o sucesso ou a falha de uma solicitação;
- **Endereço IP:** Um endereço IP (*Internet Protocol*) é um identificador exclusivo atribuído a todos os dispositivos na Internet. Quando um cliente envia uma solicitação para um *Web Server*, ele usa o endereço IP do servidor para direcionar a solicitação para o local correto;
- **Página Web:** uma página da Web é um documento ou recurso que pode ser acessado por meio da *World Wide Web*. Ele normalmente contém texto, imagens e outros elementos multimídia e é exibido por um navegador da *Web*.

Mais especificamente, um *Web Server* recebe solicitações de navegadores cliente usando algum tipo de protocolo, sendo ele HTTP ou HTTPS e, em seguida, responde a essas solicitações enviando de volta as páginas da *Web*, imagens ou outros recursos solicitados, podendo também executar outras tarefas tais como processar formulários, executar *scripts* e gerenciar bancos de dados.

Os *Web Servers* podem ser dispositivos de Hardware ou Software (aplicações) executados em um *Host*. Softwares populare de *Web Server* incluem o Apache, Nginx, IIS e GWS (*Google Web Server*), que podem ser configurados e personalizados com vários recursos para otimizar o desempenho, a segurança e a escalabilidade.

Além de servir páginas da *Web*, os *Web Servers* também podem lidar com outros tipos de tráfego da Internet, como solicitações de transferência de e-mail ou arquivos, e podem ser usados para executar vários tipos de aplicativos, incluindo CMS (*Content Management System*), sites de comércio eletrônico e fóruns on-line. [71]

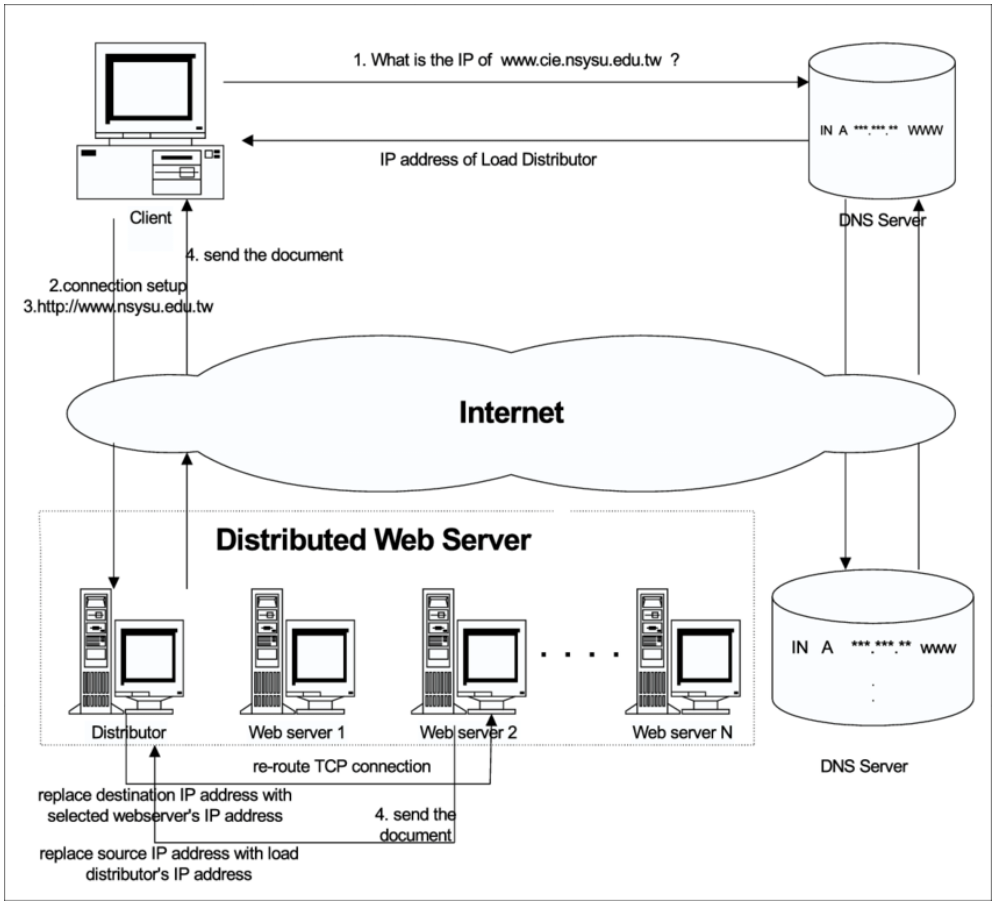

Figura 3.4: Visão Geral de um Web Server Distribuído - Fonte: [4]

3.6 ALGORÍTIMOS DE DETECÇÃO E RECONHECIMENTO FACIAL

3.6.1 Haar Cascade

Para o presente projeto foi utilizado o algoritmo Haar Cascade para realizar a identificação dos rostos e posteriormente o reconhecimento facial. A detecção utilizando classificadores em cascata baseados em recursos Haar é um método eficaz de detecção de objetos proposto por Paul Viola e Michael Jones. É uma abordagem baseada em aprendizado de máquina em que a função cascata é treinada a partir de muitas imagens positivas e negativas [5].

O algoritmo pode ser explicado em quatro etapas [6]:

1. Cálculo das características de Haar: inicialmente, o algoritmo precisa de muitas imagens positivas (imagens de rostos) e imagens negativas (imagens sem rostos) para treinar o classificador.

Em seguida, é preciso extrair recursos Haar. Cada recurso é um valor único obtido pela subtração da soma dos *pixels* sob o retângulo branco da soma dos *pixels* sob o retângulo preto. Alguns tipos de recursos encontram-se na imagem 3.5:

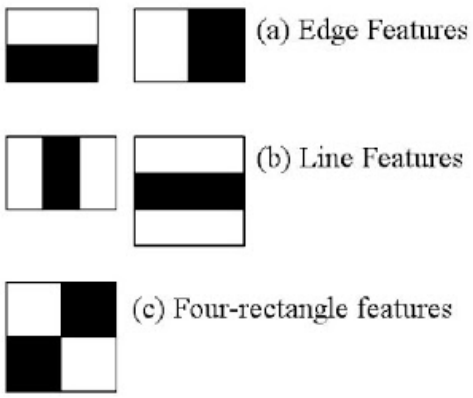

Figura 3.5: Tipos de Recursos Haar - Fonte: [5]

2. Criando Imagens Integrais: para não precisar computar cada *pixel*, o algoritmo cria sub-retângulos e cria referências de matriz para cada um desses sub-retângulos, sendo estes usados para calcular os recursos de Haar.

3. Treinamento Adaboost: para determinar as melhores características em um objeto entre as centenas de milhares de características de Haar utiliza-se o Adaboost. O Adaboost basicamente escolhe os melhores recursos e treina os classificadores para usá-los. Ele usa uma combinação de "classificadores fracos" para criar um "classificador forte" que o algoritmo pode usar para detectar objetos.

   O classificador final é uma soma ponderada desses classificadores fracos. É chamado de fraco porque sozinho não consegue classificar a imagem, mas junto com outros forma um classificador forte.

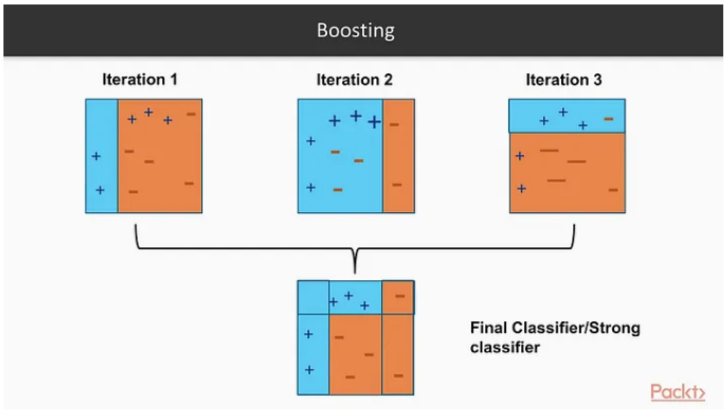

Figura 3.6: Representação de um algoritmo de boosting - Fonte: [6]

4. Implementação de Classificadores em Cascata. essa última etapa combina esses classificadores fracos em um classificador forte usando classificadores em cascata.

Os recursos são agrupados em diferentes estágios de classificadores e aplicados um a um. Se uma janela falhar no primeiro estágio, será descartada. Não considerando os recursos restantes nele. Se passar, será aplicada a segunda etapa de recursos e continuar o processo. A janela que passa por todos os estágios é uma região de face.

3.6.2 K-Nearest Neighbors (KNN) Classification

KNN (K-Nearest Neighbours) é um dos algoritmos de classificação mais básicos em aprendizado de máquina. Pertence à categoria de aprendizado supervisionado de aprendizado de máquina. O KNN costuma ser usado em aplicativos de pesquisa em que você procura itens "semelhantes" [72], como no reconhecimento facial.

Essa classificação é baseada em vizinhos sendo um tipo de aprendizagem baseada em instâncias, ou seja, ela não tenta construir um modelo interno geral, mas simplesmente armazena instâncias dos dados de treinamento.

A classificação é calculada a partir de uma maioria simples de votos dos vizinhos mais próximos de cada ponto: um ponto de consulta é atribuído à classe de dados que possui mais representantes dentro dos vizinhos mais próximos do ponto.

O scikit-learn, biblioteca em python utilizada nesse projeto, implementa dois classificadores de vizinhos mais próximos diferentes: KNeighborsClassifier, que implementa o aprendizado baseado no vizinhos mais próximos de cada ponto de consulta, onde é um valor inteiro especificado pelo usuário e o RadiusNeighborsClassifier implementa o aprendizado com base no número de vizinhos dentro de um raio fixo de cada ponto de treinamento, onde é um valor de ponto flutuante especificado pelo usuário [73]. No escopo desse projeto é utilizado a primeira opção.

A classificação básica dos vizinhos mais próximos usa pesos uniformes: ou seja, o valor atribuído a um ponto de consulta é calculado a partir de uma maioria simples de votos dos vizinhos mais próximos.

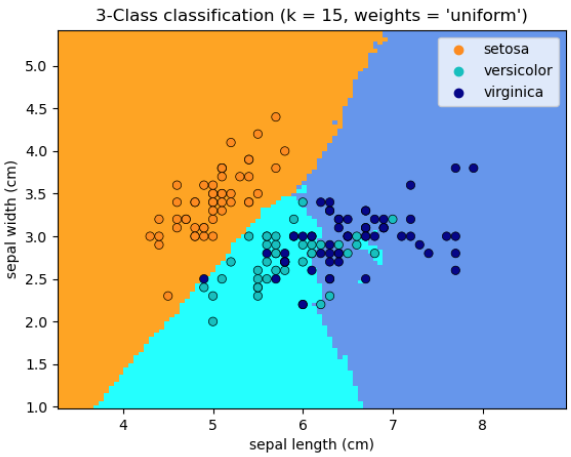

Figura 3.7: Exemplo de uso da classificação de vizinhos mais próximos cujo qual será feito uma tomada de limites de decisão para cada classe - Fonte: [7]

4 FERRAMENTAS UTILIZADAS

A descrição deste capítulo tem como foco detalhar as ferramentas utilizadas para o desenvolvimento deste projeto. Neste, será possível entender o funcionamento e as principais funcionalidades e aplicabilidade com base no cenário proposto.

4.1 BIBLIOTECAS

4.1.1 OpenCV

O **OpenCV** (*Open Source Computer Vision Library*) é uma biblioteca de software de visão computacional e *machine learning* de código aberto. O **OpenCV** foi desenvolvido para fornecer uma infraestrutura comum para aplicativos de visão computacional e acelerar o uso da percepção da máquina nos produtos comerciais. Sendo um produto licenciado Apache 2, o **OpenCV** torna mais fácil para as empresas utilizar e modificar o código [74].

A biblioteca possui mais de 2.500 algoritmos otimizados, que inclui um conjunto abrangente de visão computacional clássica e de última geração, além de algoritmos de aprendizado de máquina. Esses algoritmos podem ser usados para detectar e reconhecer rostos, identificar objetos, classificar ações humanas em vídeos, rastrear movimentos de câmeras, rastrear objetos em movimento, extrair modelos 3D de objetos, produzir nuvens de pontos 3D a partir de câmeras estéreo, unir imagens para produzir alta resolução imagem de uma cena inteira, encontrar imagens semelhantes em um banco de dados de imagens, remover olhos vermelhos de imagens tiradas com *flash*, seguir movimentos oculares, reconhecer cenários e estabelecer marcadores para sobrepô-los com realidade aumentada, etc [74].

Python, Java, C++ e outras linguagens de programação podem ser utilizadas com **OpenCV**, pois é plataforma cruzada. Além da ampla variedade de sistemas operacionais que pode ser usada como Android e *Windows*, bem como Linux, BlackBerry OpenBSD, iOS, Maemo e OS X [75].

4.1.1.1 Motivação para a escolha da biblioteca **OpenCV**

O **OpenCV** tem mais de 47 mil pessoas na comunidade de usuários e um número estimado de downloads superior a 18 milhões. A biblioteca é amplamente utilizada em empresas, grupos de pesquisa e por órgãos governamentais.

Juntamente com empresas bem estabelecidas como *Google*, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota que empregam a biblioteca, existem muitas startups, como Applied Minds, VideoSurf e Zeitera, que fazem uso extensivo do **OpenCV** [74].

Sendo assim, uma ferramenta amplamente utilizada e atual dentro do mercado de TI, além da ampla comunidade de apoio, com diversos tutoriais e documentações para auxilio e entendimento dessa biblio-

teca.

4.1.2 Flask

Flask é um *framework* escrito em **Python**. Foi desenvolvido por Armin Ronacher, que liderou uma equipe internacional do **Python** chamada Poocco. O **Flask** é baseado no kit de ferramentas Werkzeg WSGI e no mecanismo de modelo Jinja2.

WSGI (*Web Server Gateway Interface*) é a uma especificação para uma interface simples e universal entre servidores *web* e aplicações *web*, onde Werkzeg é uma biblioteca que não impõe nenhuma dependência. Cabe ao desenvolvedor escolher um mecanismo de modelo, um adaptador de banco de dados e até mesmo como lidar com as solicitações [76]. Já o Jinja é um mecanismo de template da web para a linguagem de programação **Python**. O **Flask** depende do mecanismo de modelo Jinja e do kit de ferramentas Werkzeug WSGI [77].

Como o **Flask** é frequentemente denominado como uma estrutura de prototipagem, ou seja, ele não inclui a camada de abstração para o banco de dados ou qualquer tipo de validação e segurança. Portanto, o **Flask** da total flexibilidade ao implementador para adicionar os requisitos que acharem necessário [78], fornecendo ferramentas poderosas para tarefas comuns de desenvolvimento da *Web* e incentiva a abordagem de "trazer sua própria biblioteca" para todo o resto, permitindo aos programadores a flexibilidade de escolher os melhores componentes para seus aplicativos [79].

4.1.2.1 Motivação para a escolha do framework **Flask**

O **Flask** está entre os 20 *frameworks* mais procurados de acordo com a Pesquisa *Stack Overflow* de 2022 [80], mesmo sendo considerado um microframework.

É considerado uma ferramente leve - há poucas partes constituintes que precisam ser montadas e remontadas e não depende de um grande número de extensões funcionar. Além de ser considerada flexível e escalável, o que se encaixa perfeitamente desse projeto, diante da proposta de *fog computing*.

4.1.3 Pandas

É uma biblioteca **Python** open-source especializada em análise e manipulação de dados, ele tem capacidade de trabalhar com dados do tipo planilha. Ela traz estrutura de dados e funções projetadas para tornar o projeto com dados estruturados mais rápido e expressivo. Possui funcionalidades para ler e escrever dados, lidar com perda de dados, remodelação do dataset e até modulação, podendo dividir, inserir e excluir variáveis [81].

Pandas tem uma ampla gama de recursos para entrada/saída formatos de dados, como Excel, .csv, Python/NumPy, HTML, SQL etc. Além disso, os **Pandas** têm possibilidades poderosas de consultas, cálculos estatísticos e visualizações básicas [82].

Os recursos do **Pandas** podem ser categorizados como: exploração e análise dos dados rapidamente,

leitura de vários formatos de arquivo e manipulação de dados. Ele trabalha com objetos chamados *"Data Frame"*. O *Data Frame* é um dado estruturado bidimensional onde os dados são armazenados de forma tabular na forma de linhas e colunas [83].

4.1.3.1 Motivação para a escolha da biblioteca **Pandas**

Conforme artigos como [82] e [84], para pré-processamento e manipulação de dados, é recomendado o uso de **Pandas**. Tem uma comunidade forte de suporte, uma rica oferta de funcionalidades e nenhuma concorrência.

Algumas empresas usam o **Pandas** como sistema de recomendação, como a *Netflix* que usa em sua grande coleta de dados sobre as preferências de seus clientes para fornecer sugestões aos seus usuários. A *Amazon* realiza extensa análise de dados para criar sistemas de recomendação poderosos. O YouTube usa análise de dados para recomendar vídeos a seus usuários [83].

4.1.4 scikit-learn

scikit-learn é uma biblioteca de aprendizado de máquina de código aberto escrita em **Python**. Ele permite a integração rápida de métodos de aprendizado de máquina em **Python**. A biblioteca **scikit-learn** compreende uma ampla variedade de métodos para classificação, regressão, estimativa de matriz de covariância, redução de dimensionalidade, pré-processamento de dados [81].

O pacote **scikit-learn** abrange quatro tópicos principais relacionados ao aprendizado de máquina: transformação de dados, aprendizado supervisionado, aprendizado não supervisionado e avaliação e seleção de modelos [85].

A transformação de dados é uma etapa crucial para análise de todos os dados. Existem transformações comuns (por exemplo, diferenciação e *log-odds ratio*) frequentemente usados para várias variáveis de entrada. O **scikit-learn** fornece várias funções para executar esses métodos "paratransformação"de dados e pré-processamento.

Aprendizagem supervisionada refere-se a um subconjunto de máquinas algoritmos de aprendizado que estabelecem um mapeamento entre as variáveis de recurso e suas variáveis de destino correspondentes, já no aprendizado não supervisionado os dados de treinamento consistem em um conjunto de vetores de entrada sem nenhum valor alvo correspondente.

E na seleção de modelos, o **scikit-learn** fornece métricas simples para avaliar a concordância entre os rótulos previstos e os rótulos verdadeiros dentro do aprendizado de máquina, evitando sobre ajuste nos métodos aplicados.

Sua construção é baseada na interação com outras bibliotecas tecnologias, numéricas e científicas, como NumPy e SciPy e Cython. A estrutura de dados base usada para dados e parâmetros de modelo é Numpay. O Scipy agrega com algoritmos eficientes para álgebra linear, representação de matriz esparsa, funções especiais e funções estatísticas básicas e o Cython torna mais fácil alcançar o desempenho de linguagens compiladas com sintaxe semelhante ao **Python** e operações de alto nível [86].

4.1.4.1 Motivação para a escolha da biblioteca **scikit-learn**

O **scikit-learn** difere de outras ferramentas de aprendizado de máquina em **Python** por várias razões: i) é distribuído sob a licença BSD ii) incorpora código compilado para eficiência, ao contrário do MDP (*Markov Decision Process*) e pybrain, iii) depende apenas de numpy e scipy para facilitar a distribuição fácil, ao contrário de pymvpa que tem opcional dependências como R e shogun, e iv) foca na programação imperativa, ao contrário do pybrain que usa uma estrutura de fluxo de dados. Embora o pacote seja escrito principalmente em **Python**, ele incorpora as bibliotecas C++ LibSVM e LibLinear que fornecem implementações de referência de SVMs e modelos lineares generalizados com licenças compatíveis [86].

Além disso, possui forte suporte da comunidade para documentação, rastreamento de bugs e garantia de qualidade e seu algoritmo de implementação dos métodos de aprendizado de máquina é otimizada para eficiência de computação [85].

4.2 RASPBERRY PI

Uma *Raspberry* Pi 4 é uma placa baseada em microprocessador, que pode ser definido como um mini-computador, já que possui todas as suas funções básicas [59], foi originalmente concebido como uma ferramenta educacional para promover o ensino de informática básica nas escolas. No entanto, rapidamente se tornou muito popular como um computador barato de uso geral sem periféricos [87], principalmente com o crescimento da popularidade do IoT, sendo um elemento importante dentro da arquitetura de *fog computing*.

Nessa placa é possível instalar um sistema operacional, fazer suporte à comunicação através de cabo Ethernet, Wi-Fi e Bluetooth, além de possuir entradas USB e HDMI, para conexões de teclado, mouse e tela ou aumento de capacidade de armazenamento ou agilidade no sistema. O modelo utilizado possui 32 GB de armazenamento e 8 GB de memória RAM embarcado, foi adicionado um Pen Drive com 32 GB para aumento de recurso na placa.

Uma possibilidade bastante explorada nesse projeto, é o fato da *Raspberry* Pi 4 poder ter a funcionalidade de servidor [57]. Sendo configurado dessa forma, ele consegue suportar diversas linguagens, aplicações e serviços. Além disso, pode ser acessado por meio de monitores, teclados e mouse, mas também pelo protocolo SSH, de forma remota, sendo possível manipular suas configurações da mesma maneira que é feita em servidores.

4.2.1 Motivação para a escolha da Raspberry

Em uma arquitetura de *fog computing*, o uso da *Raspberry* Pi se torna muito benéfico pela possibilidade trazer uma componente pequeno mas poderoso em processamento perto das aplicações de fato a um custo acessível.

Além da sua versatilidade, foi possível configurá-la conforme a necessidade do projeto de forma simples com bastante informação e material disponibilizado de fácil acesso, ajudando em seu manuseio e em

uma possível replicação do escopo proposto.

Figura 4.1: Visão superior de uma RaspBerry Pi - Fonte: [8]

4.3 *WINDOWS* OS

O *Windows* OS foi pensado e criado em meados de 1981 mediante a identificação, por parte de sua detentora (Microsoft), da necessidade da criação de uma interface gráfica em que fosse possível exibir informações e acessar informações através de periféricos (teclado e mouse) [88].

Apesar do seu desenvolvimento em 1981, o *Windows* só foi efetivamente considerado um Sistema Operacional apenas dois anos depois, em 1983. Esta demora ocorreu devido ao fato da construção do Sistema anteriormente a esta data, ser baseada apenas em sistemas gráficos que, de acordo com [88], eram executados sobre alguma versão dos sistemas compatíveis com DOS (*Disk Operating System*), como por exemplo:

- MS-DOS (*MicroSoft Disk Operating System*);
- PC-DOS (*Personal Computer Disk Operating System*;
- DR-DOS (*Digital Research's Disk Operating System*)

O *Windows* SO passou por várias versões de aprimoramento, tendo destaque para as versões comentadas a seguir:

- ***Windows* XP (NT 5.1):** Considerada por muitos a melhor versão do SO, foi lançada em meados de 2001 e descontinuada pela Microsoft somente em 2014 (*End of Support*), sendo baseado no antigo OS/2 da IBM cujo direitos foram adquiridos pela Microsoft. A partir do *Windows* XP foram introduzidas novas experiências de interface, apresentando melhorias de desempenho de inicialização.
- ***Windows* 7:** Devido a sua inovação no quesito de design, recursos e ferramentas, se tornou um dos sistemas mais populares da Microsoft, tendo seu lançamento em 2009 e sua descontinuação apenas em janeiro de 2020 (*End of Support*). O *Windows* 7 foi um sistema de 32 e 64bits que trouxe maior compatibilidade com ferramentas e aplicações, tais como *Aero Shake*, *Aero Peek* e *Aero Snaps*, que foram novos recursos de navegação do sistema.

- ***Windows* 10:** Lançado em 2014, simbolizou o retorno do menu "Iniciar"após fracassadas tentativas de modernização visual do sistema trago pelas versões 8 e 8.1 do Sistema Operacional por parte da Microsoft. O *Windows* 10 trouxe a possibilidade de unificação de sistemas e aplicativos, através da *Windows Store*. Contudo, pelo pouco sucesso das modalidades *Windows Phone* e *Windows RT*, esta unificação se concentrou posteriormente apenas nas aplicações para Computadores. O *Windows* 10 será a versão utilizada para embarcação do sistema proposto por este trabalho.

4.3.1 Motivação para a escolha do *Windows* OS

A Microsoft é a empresa lider do quadrante mágico do Gartner no quesito de Plataformas de Serviços e Conteúdos, como citado por [9], pelo quinto ano consecutivo (em 2021).

Figura 4.2: Quadrante Mágico da Gartner® para Plataformas de Serviços de Conteúdo. - Fonte: [9]

Não somente, é o sistema operacional mais utilizado do mundo que, conforme citado por [89], compreende extraordinários 74,8% do mercado. Este resultado tem muito impacto devido ao suporte oferecido pelos desenvolvedores, com atualizações semanais, e uma imensa compatibilidade com programas, jogos e hardwares.

Por fim, apresentamos de forma sintética algumas das vantagens de usar o *Windows* OS:

- **Interface amigável:** O *Windows* OS tem uma interface gráfica do usuário (GUI) amigável que facilita a navegação e o uso do sistema operacional pelos usuários;
- **Compatibilidade:** O *Windows* OS é compatível com uma ampla gama de hardware e software, facilitando a localização de dispositivos e aplicativos compatíveis pelos usuários;
- **Ferramentas de produtividade:** O *Windows* OS vem com uma variedade de ferramentas de produ-

tividade, como o Microsoft Office Suite, que inclui Word, Excel, PowerPoint e outras ferramentas de software úteis;

- **Suporte:** O *Windows* OS tem uma grande base de usuários e uma vasta rede de recursos de suporte, incluindo fóruns on-line, tutoriais e canais de suporte ao cliente;

- **Integração:** O *Windows* OS se integra bem a outros produtos da Microsoft, como o Azure e o OneDrive e também com outros produtos de outros fabricantes, como o *Google* Cloud da *Google*, permitindo uma integração perfeita com serviços e aplicativos baseados em nuvem.

No geral, o *Windows* OS oferece uma série de vantagens que o tornam uma escolha popular para os usuários, incluindo sua interface amigável, ampla compatibilidade com hardware e software, ferramentas de produtividade, suporte a jogos e grande rede de suporte.

4.4 *GOOGLE CLOUD PLATAFORM*

Um dos maiores provedores no espaço da computação em nuvem é o *Google*, com sua nuvem oferta de recursos denominada "*Google Cloud Platform*", popularmente conhecida como GCP. O *Google* também é um dos principais líderes de tecnologia no espaço da Internet, com uma variedade dos principais produtos da web, como Gmail, YouTube e *Google* Maps. Lançada em 2011, a plataforma tem, pelo menos, 21 zonas de disponibilidade espalhadas pelo mundo [90].

Esses produtos geram, armazenam e processam toneladas de terabytes de dados todos os dias de usuários da Internet ao redor do mundo. Para lidar com esses dados significativos, o *Google* ao longo dos anos investiu pesadamente na infraestrutura de processamento e armazenamento, em consequência disso, atualmente encontra-se no quadrante mágico de líderes do Gartner [10]. Ele possui alguns dos designs e tecnologia de data center mais impressionantes do mundo para suportar suas demandas computacionais e serviços de computação [91].

As opções de armazenamento em nuvem do *Google* fornecem acesso de armazenamento escalonável e em tempo real para viver e dados de arquivamento dentro do perímetro da nuvem. O Cloud Storage, por exemplo, utilizado no escopo desse projeto, é configurado para atender para qualquer demanda de armazenamento concebível. Os dados armazenados no armazenamento em nuvem estão disponíveis a qualquer momento e de qualquer lugar do mundo.

O Cloud Storage oferece recursos de design seguro para proteção de dados e controles e recursos avançados para manter os dados privados e protegidos contra vazamentos ou comprometimentos. Os recursos de segurança incluem políticas de controle de acesso, criptografia de dados, políticas de retenção, bloqueios de política de retenção e URLs assinados. [92].

Figura 4.3: Quadrante Mágico da Gartner® para Infraestrutura de nuvem e serviços de plataforma. - Fonte: [10]

4.4.1 Motivação para a escolha da GCP

O *Google Cloud Platform (GCP)* oferece uma série de vantagens para indivíduos e organizações que procuram soluções de computação em nuvem. Ele oferece oferece uma ampla gama de serviços, incluindo máquinas virtuais, armazenamento, rede e bancos de dados, o que permite que as organizações escolham os serviços mais adequados às necessidades, com um crédito inicial de $300 doláres, permitindo realizar testes de forma confiável e simples.

Além disso, a GCP possui uma interface interativa e intuitiva, além de documentações e tutoriais de fácil acesso, permitindo atingir os objetivos desse projeto de forma otimizada. Além de integração a uma variedade de outras ferramentas e serviços, incluindo *Google* Analytics, BigQuery, **OpenCV** e TensorFlow, tornando mais fácil para as organizações como um todo incorporarem o GCP em seus fluxos de trabalho existentes, com uma infraestrutura altamente confiável projetada para ser altamente disponível, com vários data centers e redundância integrada para minimizar o tempo de inatividade.

5 ARQUITETURA PROPOSTA

Este capítulo visa descrever toda a arquitetura proposta bem como a metodologia de implementação utilizada. O objetivo da solução proposta nesse trabalho é implementar uma arquitetura de reconhecimento facial onde seja possível realizar o mapeamento do horário de registro de um usuário, baseando-se em uma arquitetura com processamento local com auxílio da nuvem para envio e controle dos dados coletados e registrados. Logo, esse trabalho tem o propósito de prover um protótipo de sistema de biometria com baixo custo e processamento local, abordando paradigmas de uma arquitetura em *fog computing*.

Baseado em [68], sua arquitetura é derivada de uma estrutura de três camadas, trazendo uma camada adicional entre os dispositivos finais e a nuvem, sendo elas: **Camada de Usuário Final**, **Camada *Fog Computing*** e **Camada Nuvem**.

A primeira camada (**Camada de Usuário Final**) é composta por dispositivos que os usuários terão acesso direto (neste caso, dispositivos IoT), sendo estes câmeras, celulares ou computadores, para acesso do serviço. É nessa camada onde será feito a requisição da aplicação, em que os indivíduos realizarão o cadastro e serão reconhecidos pelo sistema por meio de interfaces web. Apesar de muitos dispositivos dessa camada possuírem poder de computação, o objetivo dele é tanto consumir quanto fornecer informações, podendo detectar pessoas e transmitir esses dados detectados para as camadas superiores de processamento e armazenamento.

A segunda camada (**Camada *Fog Computing***) é composta por dispositivos que permitem processamento local, que nesse escopo corresponde à *RaspBerry Pi* cujo definições foram abordadas na seção 4.2. Vale destacar que, para validação sistêmica, foram realizados testes onde computadores pessoais foram configurados como *hosts* da aplicação.

Essa camada é o ponto central do projeto proposto, ela é responsável por hospedar a aplicação, onde será realizado todo processamento de identificação e registro dos indivíduos, pelo armazenamento inicial dos dados de treinamento e registro e conexão com os usuários e periféricos da camada anterior.

A terceira camada (**Camada Nuvem**) é composta por provedores de nuvem (pública, privada ou híbrida) ou até mesmo de forma *on-premises* em servidores em *Datacenters*, onde o administrador queira centralizar os registros de dados. Para este projeto foi utilizado a plataforma de núvem púbica *Google Cloud Plataform*.

Essa camada receberá os arquivos de treinamento e registro em formato de Planilha com extensão **.xmls** (*Microsoft Office Excel*). Diferente de uma estrutura em *Cloud Computing*, nem todos as solicitações de armazenamento e *storage* passarão por essa centralização mas apenas aquelas mapeadas que não necessitam de uma resposta tão otimizada dentro do escopo do nosso projeto.

De forma geral, na **Camada de Usuário Final**, os usuários irão realizar a requisição de uma página HTML por meio do IP do dispositivo e então o servidor *web* hospedado na **Camada *Fog Computing*** realizará o envio de uma resposta com o acesso à interface *web* do sistema solicitado. A interação com os usuários permitem que eles realizem seu cadastro e/ou façam registro do horário das pessoas cadastradas,

sendo esses arquivos inicialmente hospedados na **Camada *Fog Computing*** e posteriormente enviados para **Camada Nuvem**, onde terá mais recursos para armazenamento.

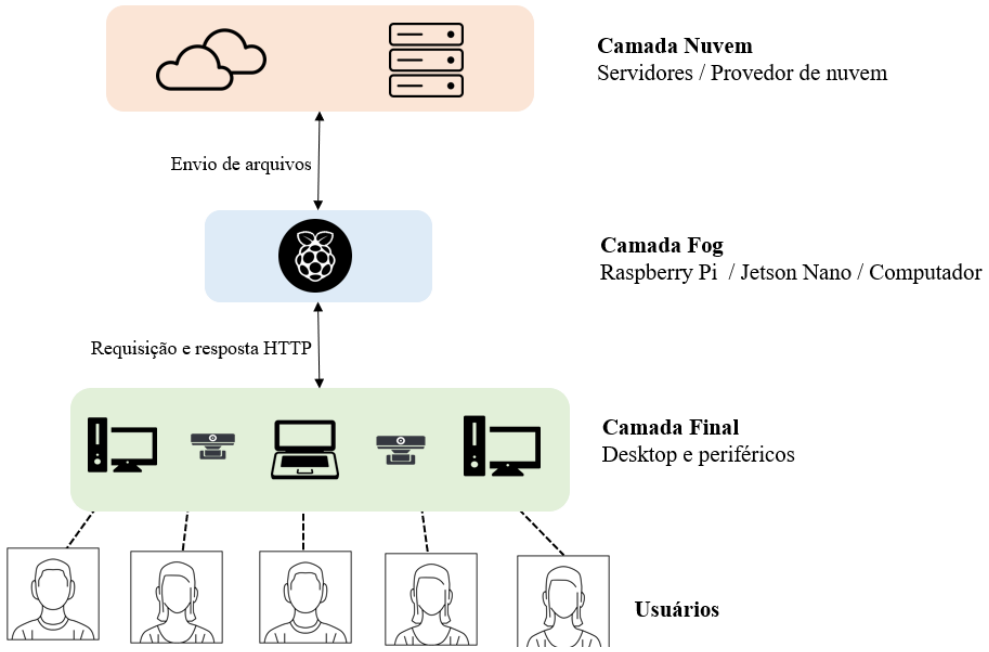

Figura 5.1: Visão macro da arquitetura implementada. - Fonte: Autores

5.1 DESENHO DA APLICAÇÃO

Para o desenho da aplicação e lógica do sistema, foi utilizado a linguagem em ***Python***. Esta lógica passa pelo registro dos dados e a implementação do servidor web em conjunto com um código HTML, utilizado para a implementação da interface do usuário. A aplicação foi hospedada em uma *Raspberry Pi*, onde foi configurado e instalado o sistema operacional Windows [93].

A arquitetura funciona por meio de um *backend* que utiliza o micro *framework* ***flask*** para transformar o código em um servidor web e poder gerenciar as ferramentas dentro da aplicação. Sendo as ferramentes uma plataforma para o usuário em HTML, os algorítimos de reconhecimento facial, o registros dos dados de cadastro e de frequência dos usuários e os algoritmos para a interação com o *Google Cloud Storage*.

O *backend* possui rotas para a disponibilização das ferramentas do sistema ao usuário através da interface no *frontend*. Essas rotas são responsáveis pelo cadastro dos usuários através da identificação do rosto do usuário cadastrado com o modelo treinado em *machine learning* com o algoritmo "*Haar Cascade*"[94] e para o reconhecimento do usuário cadastrado e registro da frequência foi utilizado o modelo baseado no algorítimo "k-nearest neighbor"[95], e por fim, para o registrar, baixar e deletar dados no *bucket* do *Google Cloud Storage* foi feita uma integração por meio de API JSON.

Por se utilizar uma arquitetura que disponibiliza os serviços através de um servidor *web*, é possível acessar o sistema por celulares e outros dispositivos que conseguem se comunicar através do protocolo HTTP.

5.1.1 Reconhecimento Facial

O reconhecimento facial foi feito com base na biblioteca **scikit-learn** em conjunto com a biblioteca openCV, onde a biblioteca **scikit-learn** foi utilizada para realizar a classificação e identificação do usuário que está registrando uma frequência no sistema sendo utilizado o algoritmo k-ésimo vizinho mais próximo (k-nearest neighbors algorithm). Através de testes de desempenho utilizando o algoritmo, foi utilizado 5 vizinhos para a identificação de um usuário previamente registrado.

O algoritmo "*Haar Cascade*", que também faz parte da biblioteca do OpenCV, identifica o rosto de uma pessoa, que através de características visuais é capaz de fazer classificações e identificações em imagens que no caso desse trabalho foi aplicado para a identificação de rostos[96].

5.1.2 Configuração para integração do ambiente em nuvem

Para execução da integração com os serviços em nuvem, é necessária uma chave de acesso com credenciais disponibilizada pela *Google Cloud* por meio de um arquivo JSON. Este arquivo foi adicionado no projeto e referenciado para que seja possível utilizar os recursos atrelados a essa chave por meio da biblioteca "google-cloud-storage".

Primeiro verifica-se se existe um *bucket* no *storage* com o nome definido no "bucket_name", sendo verificado o nome de cada *bucket* presente. Caso não exista nenhum *bucket* com nome da variável definida anteriormente, um novo *bucket* é criado.

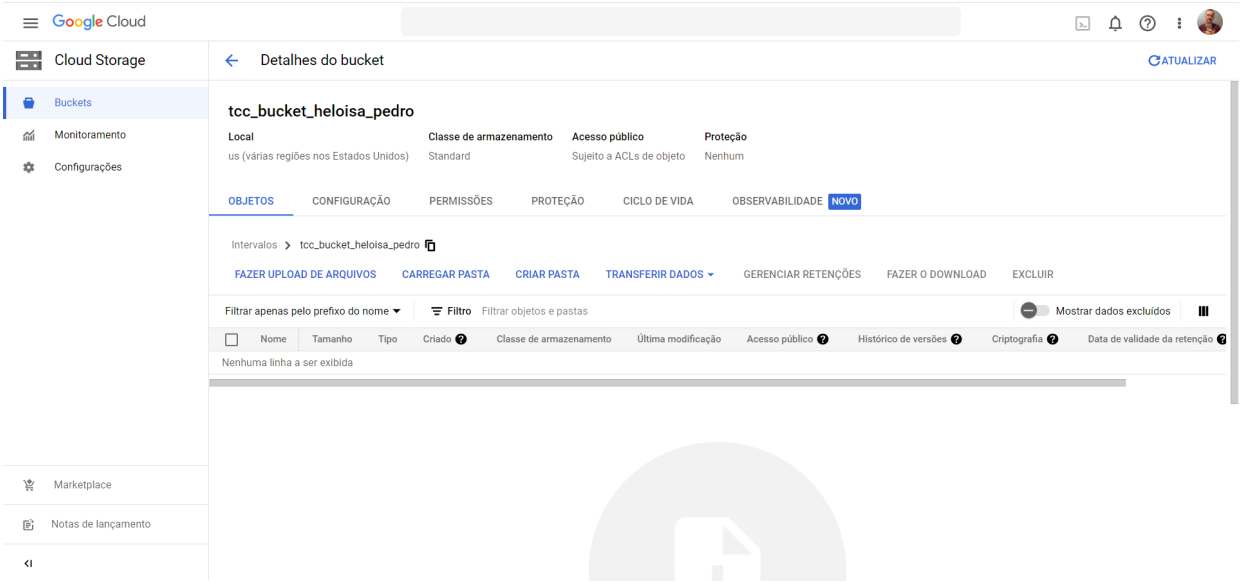

Figura 5.2: *bucket* do *Google Cloud Storage* criado pela função create_bucket() - Fonte: Autores

Existem 4 classes que podem ser classificados esse *bucket*, nesse projeto é utilizado a classe *Standard* e a localização foi definido como "US"por ser o mais próximo do Brasil e recomendado pela própria ferramenta. Caso já exista o *bucket*, ele é recuperado. Na imagem 5.3 observa-se o *bucket*s com os arquivos recuperados da camada *fog*. O detalhamento do processo é descrito nos anexos e na próxima seção, onde

demostra o processo de deletar, fazer o *upload* e *download* por meio dessa integração.

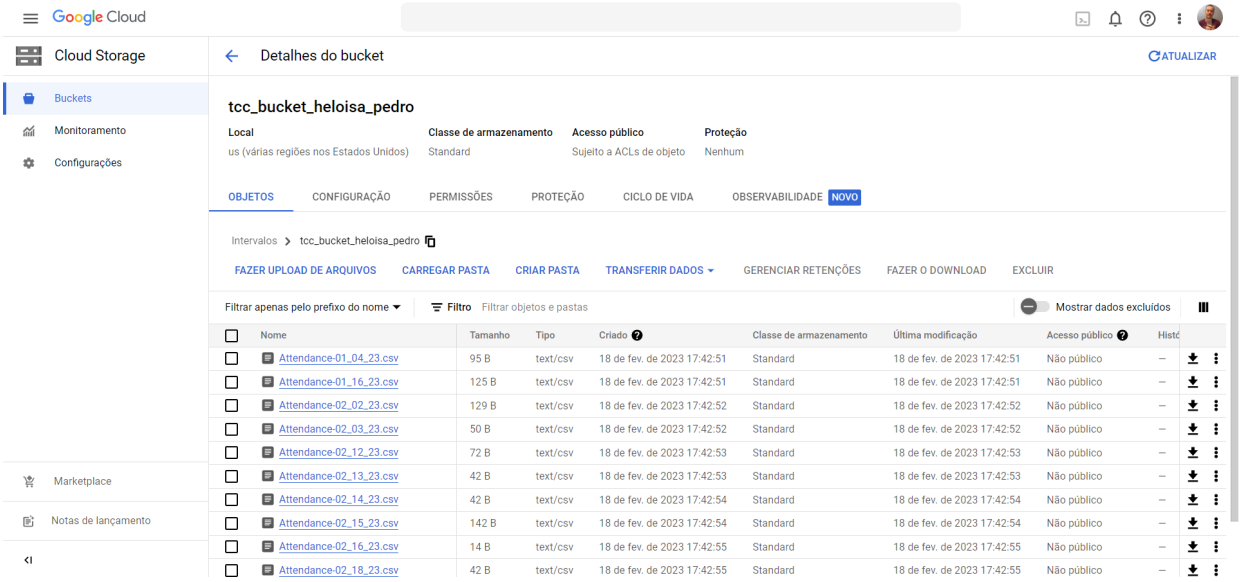

Figura 5.3: Upload de arquivos para o *bucket* do Google Cloud Storage - Fonte: Autores

5.1.3 Configuração da aplicação

A estrutura do sistema foi estruturada com os seguintes arquivos e diretórios:

- Arquivo ***Python*** "app.py": responsável por executar toda a lógica e funções do sistema
- Pasta "*templates*": onde consta um arquivo html "home.html"que monta a interface de usuário do projeto.
- Pasta "*Attendance*":onde encontra-se todos os arquivos de frequência no formato de planilhas do excel.
- Pasta "*Downloaded*": onde fica armazenada todas as frequências resgatadas da nuvem.
- Pasta "*static*": que contém os modelos treinados.
- Pasta "faces": com os usuários registrados no sistema. A padronização dessa pasta é feita como nome do usuário e o identificador separados pelo caractere "_", e dentro delas um total de 50 fotos de cada usuário.

Para configurar o servidor *web* foi utilizado o micro-framework ***flask*** do ecossistema ***Python***. Sendo assim, possível criar web sites e executá-los como servidor web em qualquer porta desejada, no caso desse projeto, será utilizada a porta 8500.

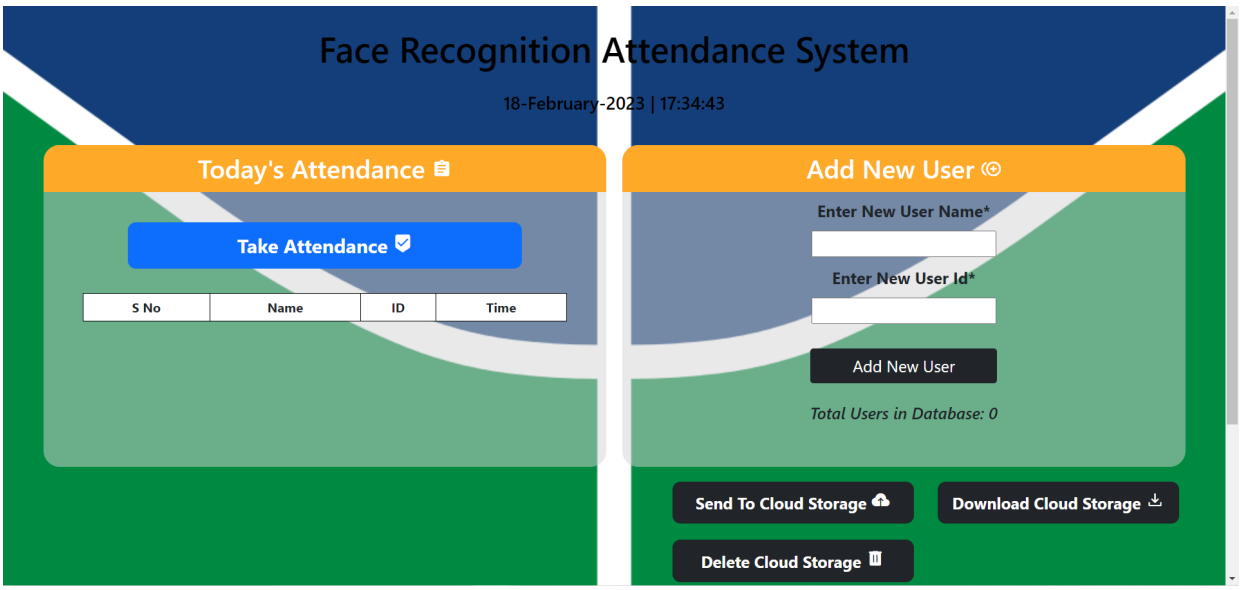

Figura 5.4: Site Carregado - Fonte: Autores

Além disso, a ferramenta ***flask*** possibilita criar rotas chamadas HTTP, no caso desse projeto, utiliza-se rotas com diferentes aplicações que serão descritas a seguir [97].

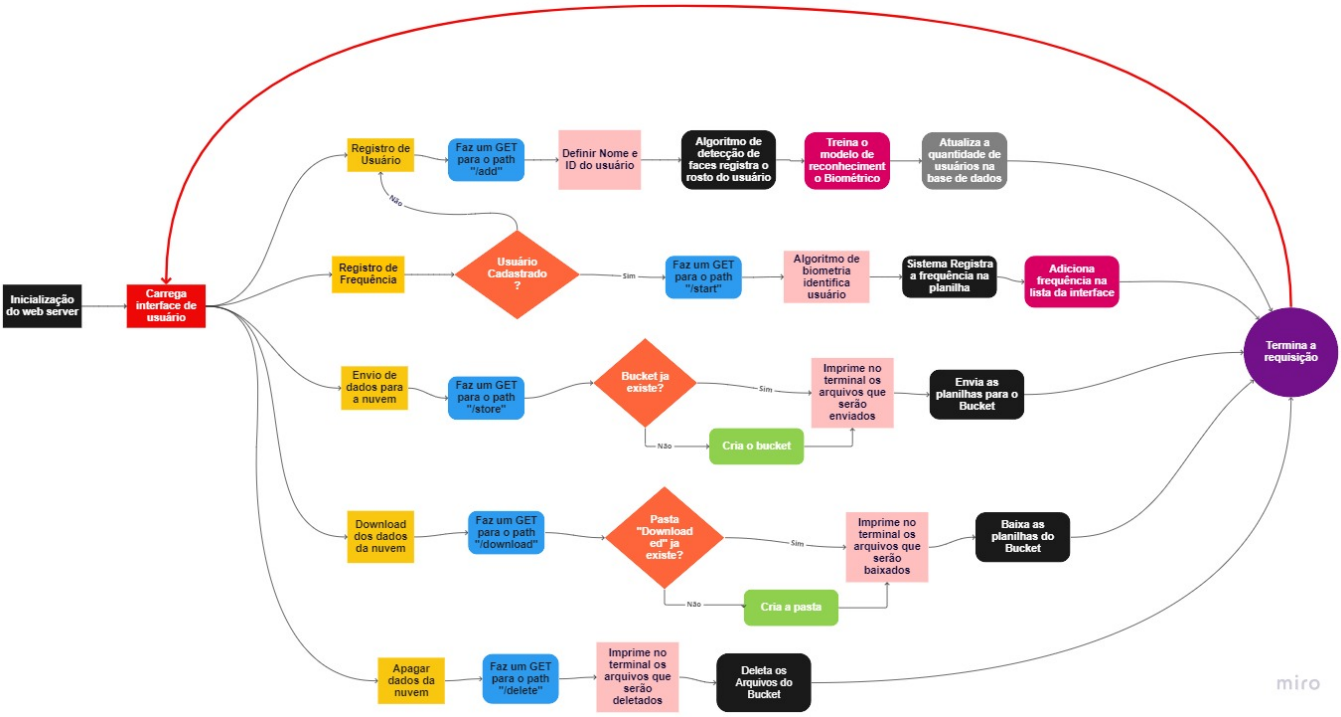

Figura 5.5: Fluxo das rotas apresentadas - Fonte: Autores

A primeira é a rota se refere ao momento em que o site é carregado, primeiro é verificado se já existe alguma frequência registrada, onde é checado na pasta "*Attendance*"se existe algum arquivo ".csv"com registros de frequência do dia atual, onde a padronização dos registros encontra-se como "*Name*", "*Roll*",

"*Time*"que respectivamente equivalem ao nome, identificador, e horário que foi feita a frequência.

Caso exista alguma frequência dentro do arquivo do dia, o conteúdo do arquivo é extraído e armazenado em variáveis, juntamente com a quantidade de registros da planilha.

A segunda rota é a rota é responsável pelo registro de usuários dentro da plataforma. É a única rota dentro da arquitetura que pode ser acessada através de um método POST, e isso ocorre quando o usuário digita o nome e o identificador nas caixas de texto devidamente preenchidas e clica no botão "*Add New User*", para adição de novo cadastro.

A partir dessa ação, o código recupera o nome e o identificador enviados e checa primeiramente se o registro do usuário já existe, caso o usuário já exista o código irá sobrescrever os dados cadastrados anteriormente. Caso não exista, uma nova pasta é criada a recebendo os dados do cadastro como parâmetro. E então, a captura de vídeo pela câmera é iniciada e o código entra em um *loop* para a captura do rosto do usuário até que tenha sido registrado um total de 50 fotos ou se a tecla "esc"for apertada.

Dentro do *loop*, a câmera captura um *frame* de imagem por cada instante de tempo e em seguida aplica-se a detecção do rosto utilizando o modelo treinado do algoritmo. Depois de registrar todas as fotos, o *loop* é quebrado e a captura de *frames* pela câmera é cessado.

E então, após o treinamento do modelo, a página inicial é carregada novamente utilizado o mesmo método utilizado na rota anterior [98].

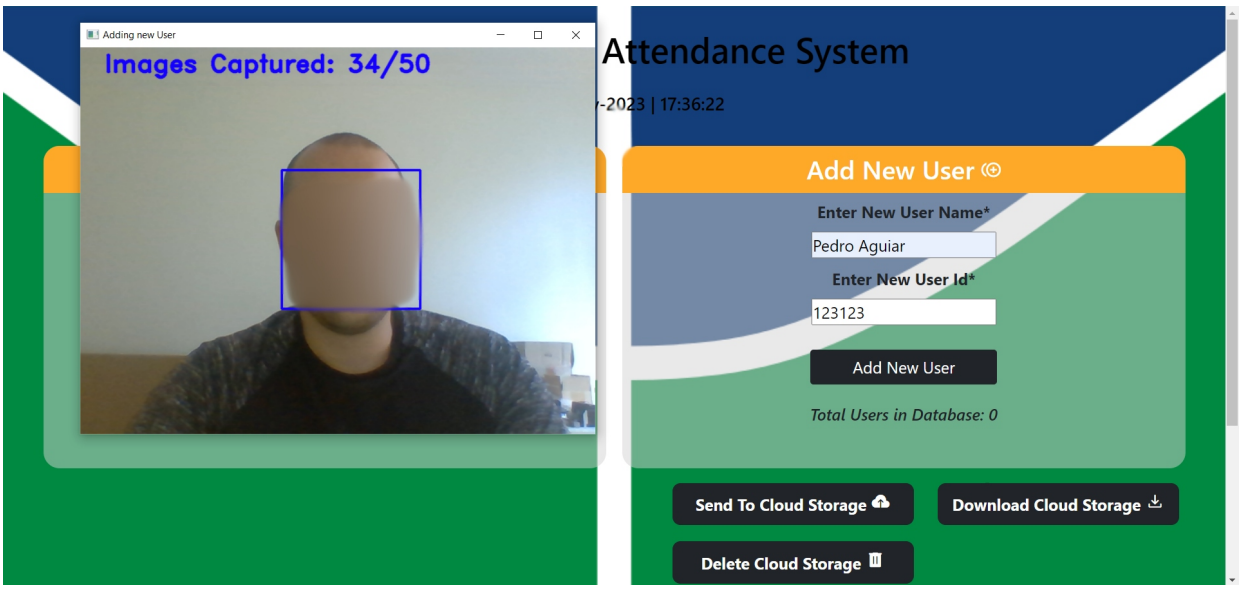

Figura 5.6: Cadastro de usuário - Fonte: Autores

A terceira rota é utilizada para fazer o reconhecimento facial do usuário a partir do modelo treinado e registrar a frequência no arquivo ".csv". Quando essa rota é chamada, primeiramente é checado se já existe um modelo treinado, e se não existir a mensagem "*There is no trained model in the static folder. Please add a new face to continue.*"é mostrada na interface do usuário.

Caso contrário a captura de vídeo é iniciada e entra num *loop* infinito em que os mesmos passos para a captura de *frames*, identificação do rosto e amostragem dos retângulos da segunda rota são feitos nova-

mente. Ao passo que a identificação de rosto seja feita e em sequência retorna o resultado da classificação.

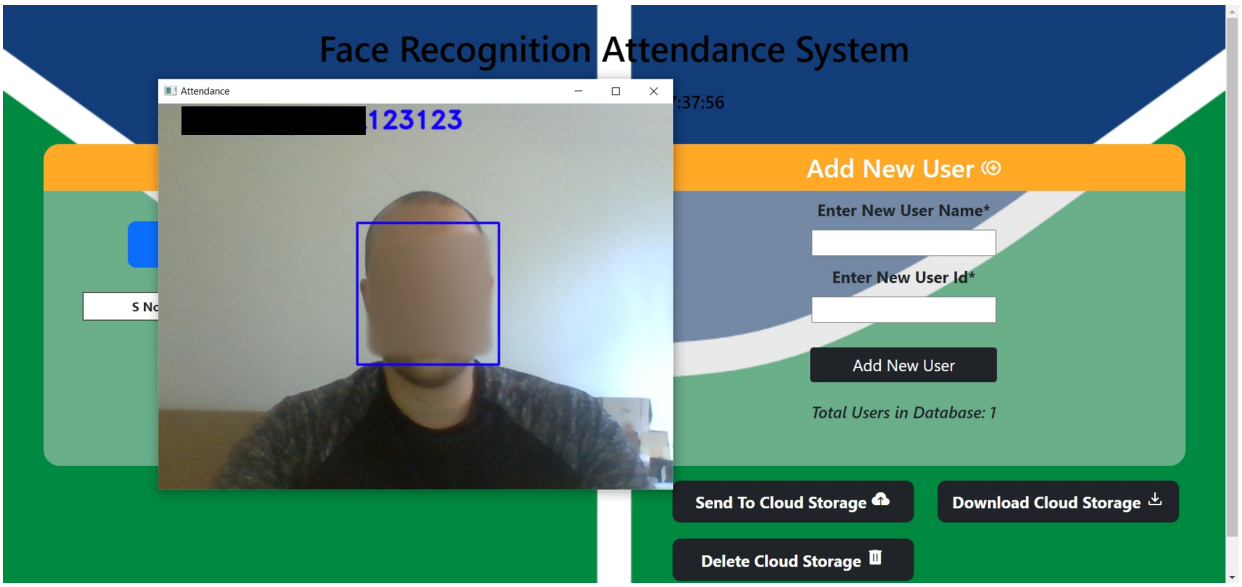

Figura 5.7: Reconhecimento utilizando modelo treinado - Fonte: Autores

Após feita a classificação e identificação do usuário, o código adiciona a caixinha de identificação de rosto, o nome do usuário que foi identificado e usa eles em conjunto com o um método que determina data e hora atual para fazer o registro da frequência em um arquivo ".csv". Para isso checa se o identificador da pessoa está dentro do arquivo ".csv"do dia atual "e caso não esteja o nome, o identificador e a hora atual são registrados no arquivo.

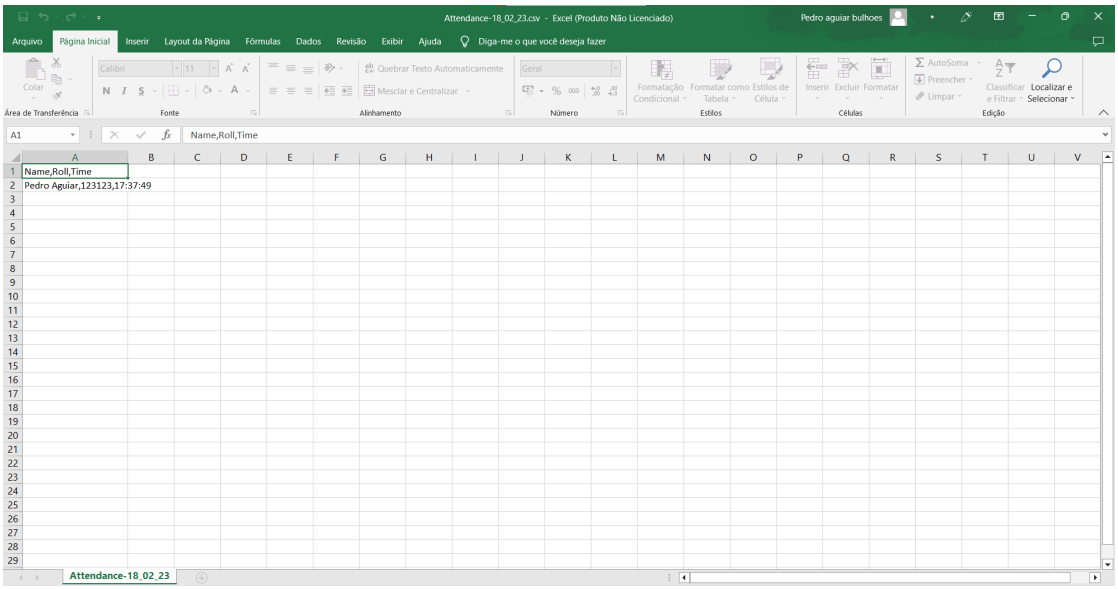

Figura 5.8: Arquivo Excel com o registro atualizado - Fonte: Autores

Uma vez feito isso, é importante ressaltar que o *loop* de identificação só é quebrado se a tecla "esc"for apertada, assim, possibilitando que múltiplas identificações sejam feitas. Por fim, depois que encerra-se

o *loop*, encerra-se também todo tipo de captura feito pela câmera e recarrega novamente a página inicial, porém atualizando a lista de frequência que é mostrada na interface de usuário.

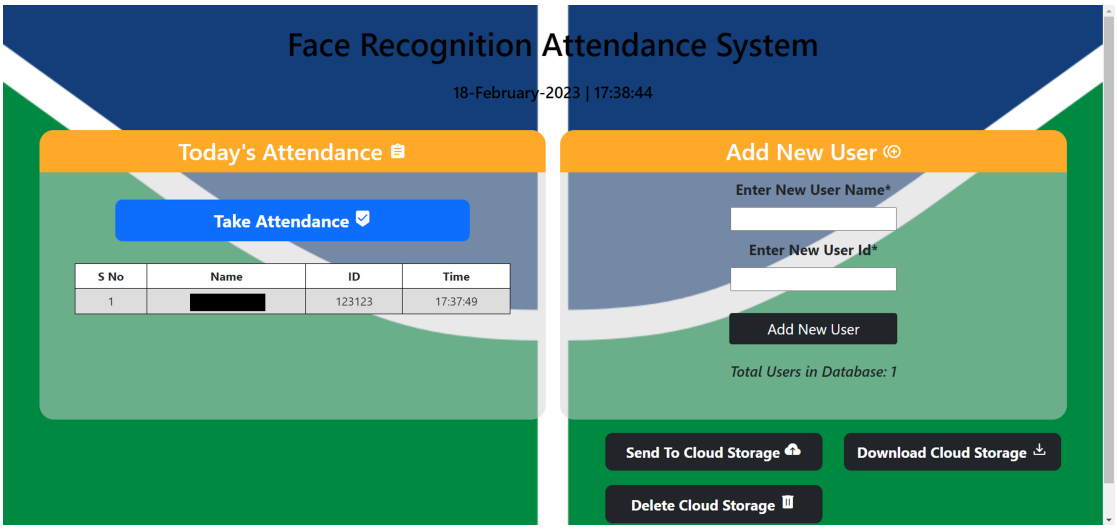

Figura 5.9: Interface de usuário com o registro atualizado - Fonte: Autores

A quarta rota tem como objetivo registrar os arquivos das frequências ".csv"localizados na pasta "*Attendance*"no ambiente da nuvem. Quando a rota é chamada, entra-se num laço de repetição selecionando cada um dos arquivos presentes na pasta, além disso, o programa imprime no terminal todos os arquivos localizados no *storage*. Após esse processo a página inicial é recarregada.

```
D:\unb\11-semestre\tcc2-attendance\face-recognition-based-attendance-system-master\codigo-tcc\Attendance\Attendance-18_02_23.csv
Attendance-01_04_23.csv
Attendance-01_16_23.csv
Attendance-02_02_23.csv
Attendance-02_03_23.csv
Attendance-02_12_23.csv
Attendance-02_13_23.csv
Attendance-02_14_23.csv
Attendance-02_15_23.csv
Attendance-02_16_23.csv
Attendance-02_18_23.csv
Attendance-18_02_23.csv
192.168.0.5 - - [18/Feb/2023 17:42:56] "GET /store HTTP/1.1" 200 -
```

Figura 5.10: Upload de arquivos para o *bucket* na nuvem - Fonte: Autores

A penúltima rota tem a função de fazer o *download* de todos os arquivos presentes dentro do *bucket* no *Google Cloud Storage*. Uma vez que essa rota é chamada é feita a listagem de todos os *blob*s do *bucket*. Em seguida é feito um laço de repetição selecionando cada arquivo contido no *storage*, e então checa-se se a pasta "*Downloaded*"existe, e caso não exista, ela é criada. Por fim, depois de todos os arquivos baixados a página inicial é recarregada.

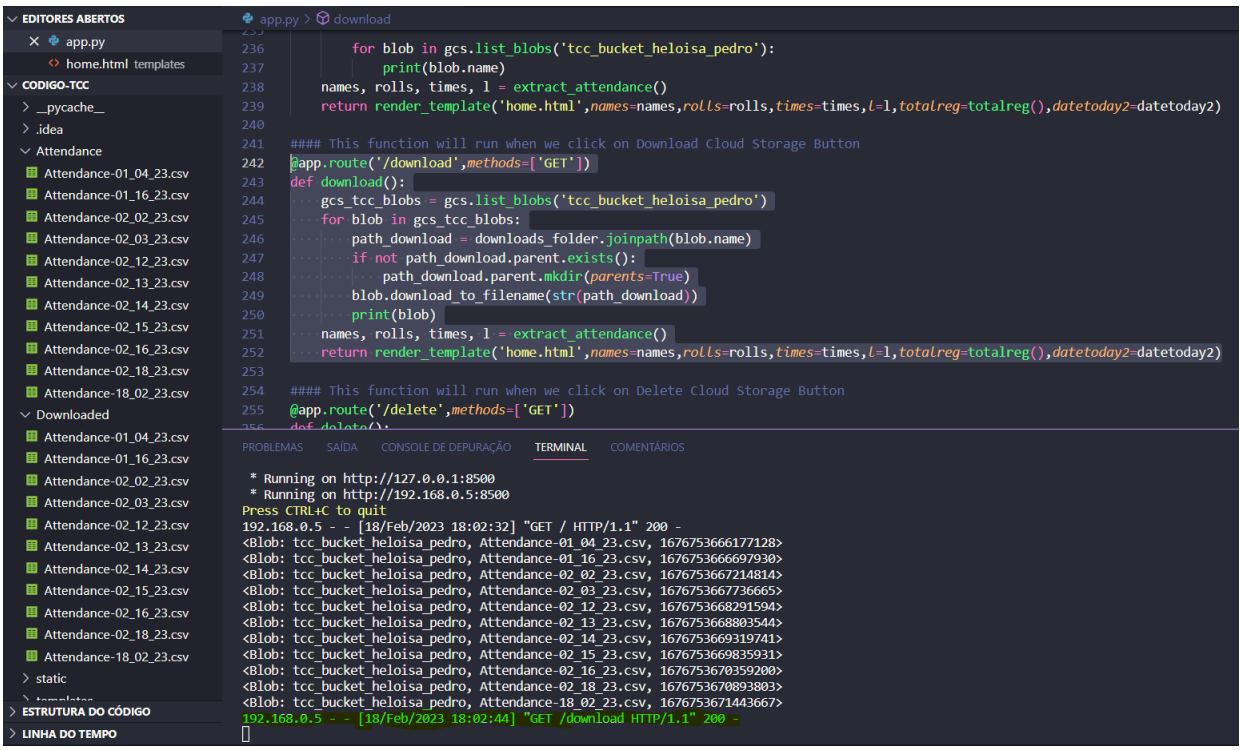

Figura 5.11: Download de arquivos contidos no *bucket* na nuvem - Fonte: Autores

Finalizado as rotas do servidor web, a última foi feita para apagar todos os arquivos dentro do *bucket* do *storage*. Feita de forma semelhante à ultima rota explicada, nessa rota é feita a listagem de todos os *blob*s presentes no *bucket* e entra em um laço de repetição, em que é selecionado e deletado cada *blob* do *Cloud Storage*. Uma vez que todos os *blob*s foram deletados, novamente a página inicial recarrega.

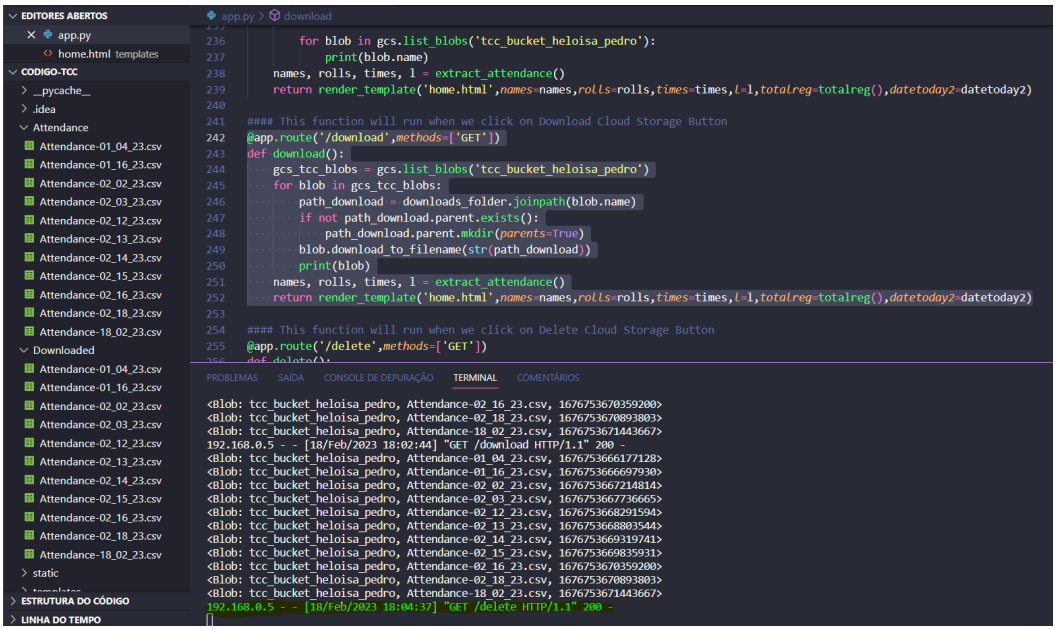

Figura 5.12: Request para deletar todos os arquivos do *bucket* na nuvem - Fonte: Autores

5.1.3.1 Interface para o usuário

Através da contextualização anterior, será descrito como funciona a interface de usuário.

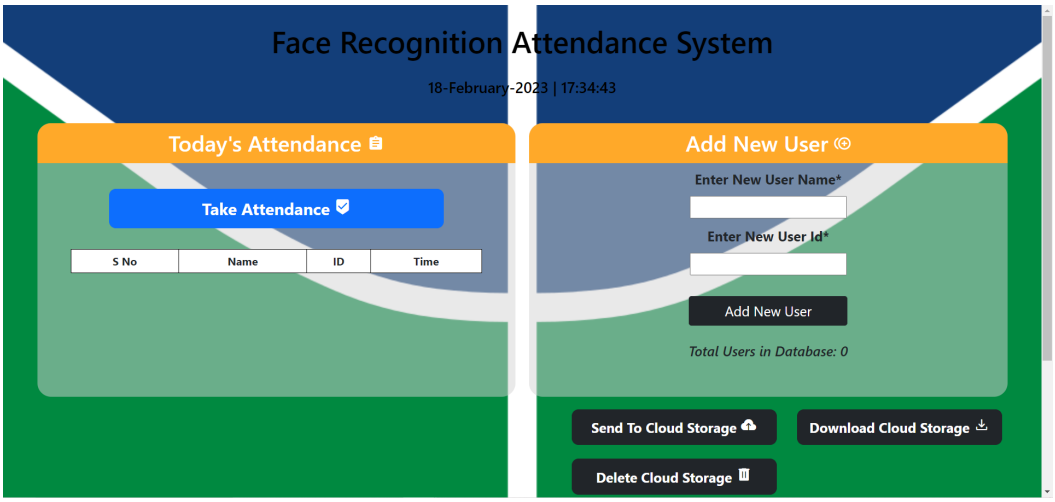

Figura 5.13: Interface de usuário - Fonte: Autores

A partir da imagem acima é possível observar a interface completa do sistema montada pelo arquivo "home.html"localizado dentro da pasta "*templates*". Sua estrutura é composta pela visualização da data e hora em tempo real no padrão dia, mês e ano, sendo um elemento importante para o registro da frequência dos usuários, feito a partir do módulo datetime do ***Python***.

É possível visualizar a seção "*Today's Attendance*", composta por um botão "*Take Attendance*"que faz uma requisição GET para a terceira rota do servidor e por uma lista de frequências retornadas do dentro do arquivo ".csv"do dia atual localizado na pasta "*Attendance*". Ao lado, direito a seção "*Add New User*", composta de um formulário com duas caixas de texto "*Enter New User Name*"e "*Enter New User Id*"em que é inserido o nome e o número identificador do usuário respectivamente, em baixo um botão "*Add new User*"que faz uma requisição GET para a segunda rota e um parágrafo "*Total Users in Database*"que mostra o número de usuários cadastrados.

E por fim, na parte inferior, encontra-se 3 botões "*Send To Cloud Storage*", "*Download Cloud Storage*"e "*Delete Cloud Storage*"que fazem requisições GET para a quarta, quinta e sexta rota, respectivamente.

6 TESTES E RESULTADOS

Este capítulo tem como objetivo apresentar os testes realizados e os resultados obtidos a partir da solução apresentada por meio do capítulo 5. Estes testes foram realizados de forma coordenada, de modo a obter a melhor acurácia possível na captura das imagens.

Reforçamos que todas as imagens utilizadas para este experimento são reais e foram coletadas com total consentimento dos voluntários, contudo foram anonimizadas para validação inicial das possibilidades do software, sem constituir uma pesquisa sistemática para efeito de desenvolvimento e operação de um sistema real.

O objetivo deste capítulo é trazer um resultado comparativo, isto é, objetiva-se expor a arquitetura apresentada pela seção 5 a quatro diferentes cenários de testes após a confecção de um banco de imagens em dois cenários diferentes de luminosidade.

6.1 PROCESSO DE COLETA

6.1.1 Cenário de Luminosidade 1: Coleta em condições de luminosidade reforçada

O primeiro cenário de luminosidade foi construído em ambiente com portas de vidro, possibilitando luz natural, com reforço de lâmpadas focadas, a fim prover a melhor luminosidade. Foi realizado o cadastro de três indivíduos distintos com captura a resoluções de 1080p pela câmera USB da marca Laojia, que possibilitou alta definição.

Foram feitas um total de 50 capturas para cada indivíduo de modo a armazenar a informação da maneira mais fidedigna possível. Tendo em vista que essa volumetria não seria adequada de se anexar neste documento, apresentamos a seguir o exemplo de captura de cada um dos indivíduos cadastrados neste cenário:

- Individuo 1: ID 21022000

Figura 6.1: Exemplo de captura do Indivíduo 1 em condições de luminosidade considerada boa a uma resolução de captura de 1080p - Fonte: Autores

- Individuo 2: ID 210998

Figura 6.2: Exemplo de captura do Indivíduo 2 em condições de luminosidade considerada boa a uma resolução de captura de 1080p - Fonte: Autores

- Individuo 3: ID 14031996

Figura 6.3: Exemplo de captura do Indivíduo 2 em condições de luminosidade considerada boa a uma resolução de captura de 1080p - Fonte: Autores

6.1.2 Cenário de Luminosidade 2: Coleta em condições de luminosidade artificial

Já o segundo cenário de luminosidade foi construído de forma artificial e dificultada propositalmente, utilizando a captura e cadastro de dois novos indivíduos distintos em ambiente interno, com captura a resoluções de 720p pela câmera integrada ao computador de modelo Lenovo (ThinkPad E14).

Esta condição foi propositalmente estabelecida com a finalidade de posteriormente, nos cenários propostos pela seção 6.2, realizar um comparativo com o Cenário de Luminosidade 1 apresentado na seção 6.1.1 e tirar as conclusões sobre a influencia dessa condição na qualidade de detecção e reconhecimento do sistema.

Foram feitas um total de novas cinquenta capturas para cada novo indivíduo de modo a armazenar a informação da maneira mais fidedigna possível. Tendo em vista que essa volumetria não seria adequada de se anexar neste documento, apresentamos a seguir o exemplo de captura de cada um dos indivíduos cadastrados neste cenário:

- Individuo 4: ID 041195

Figura 6.4: Exemplo de captura do Indivíduo 4 em condições de luminosidade precárias a uma resolução de captura de 720p - Fonte: Autores

- Individuo 5: ID 09041966

Figura 6.5: Exemplo de captura do Indivíduo 4 em condições de luminosidade precárias a uma resolução de captura de 720p - Fonte: Autores

6.2 ANÁLISE DE CENÁRIOS DE TESTE E RESULTADOS

A partir das coletas obtidas nos processos anteriormente descritos pelas seções 6.1.1 e 6.1.2 foi possível a criação de um banco de imagem de duzentos e cinquenta imagens sendo cinquenta imagens por indivíduo. Com esse banco de dados, será possível realizar o teste de reconhecimento facial do sistema nos cinco indivíduos cadastrados e em quatro cenários distintos, listados a seguir:

1. **Cenário de teste 1:** Câmera de resolução 1080p com condições de luminosidade favoráveis;
2. **Cenário de Teste 2:** Câmera de resolução 1080p com condições de luminosidade artificial;
3. **Cenário de teste 3:** Câmera de resolução 720p com condições de luminosidade favoráveis;
4. **Cenário de teste 4:** Câmera de resolução 720p com condições de luminosidade artificial.

Desta forma, esta seção visa apresentar os resultados obtidos após a realização dos testes de reconhecimento facial propostos pelos itens 6.2.1, 6.2.2, 6.2.3 e 6.2.4 tendo como base de dados seis indivíduos

cadastrados em dois cenários diferentes, que foram anteriormente descritos pelas seções 6.1.1 e 6.1.2. Foram realizados 40 testes no total, sendo 10 testes em cada cenário utilizando os 5 indivíduos do processo de coleta.

6.2.1 Cenário de Teste 1

No primeiro cenário, foi realizado os testes no cenário mais favorável que esse projeto conseguiu propor, onde a resolução da câmera possui 1080p e o ambiente se encontra totalmente iluminado.

Quando foi simulado o reconhecimento, independente da maneira que foi realizado o cadastro, foi possível perceber uma efetividade próxima ao 80%. Quando o indivíduo manteve-se parado, o sistema realizou o reconhecimento correto conforme demostrado na figura 6.6, conseguindo se ajustar de forma rápida.

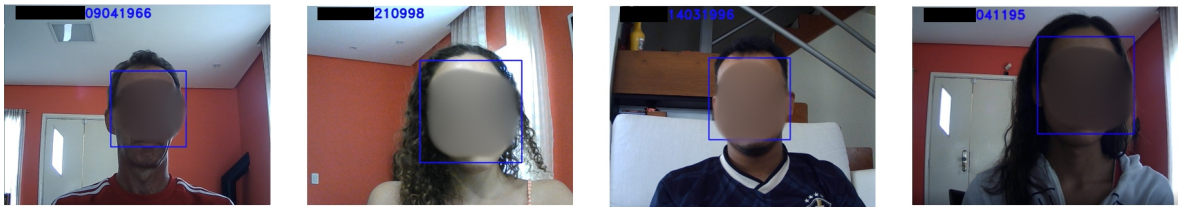

Figura 6.6: Alguns resultados da classificação utilizando câmera com resolução de 1080p e condições favoráveis

6.2.2 Cenário de Teste 2

No segundo cenário, foi alterado as condições de iluminação mantendo a resolução da câmera de 1080p. Nessas condições, percebe-se algumas instabilidades em sua análise. Em alguns indivíduos analisados, foi necessário a manipulação do ambiente para a realização da identificação correta, conforme observado em 6.11.

Figura 6.7: Alguns resultados da classificação utilizando câmera com resolução de 1080p e luz artificial

Entre os cenários propostos, este apresentou 60% de efetividade, aproximadamente, no primeiro momento, já que em 10 tentativas 4 apresentaram erros, sendo necessário intervenção no ambiente em termos de luminosidade para alcançar a resposta correta. Conforme observado na figura 6.8.

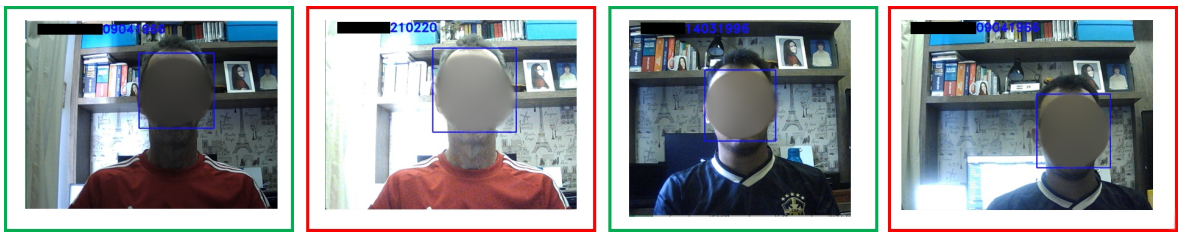

Figura 6.8: Alguns erros observados no cenário 2

6.2.3 Cenário de Teste 3

Nesse cenário, foi proporcionado uma luminosidade favorável porém com uma resolução menor na câmera de teste. Nessas condições, foi percebido um resultado positivo de forma geral.

No primeiro momento, entre os 10 testes realizados, 7 apresentaram o resultado correto. Sendo que nos 3 erros detectados, em 2 casos o sistema conseguiu se ajustar e identificar de forma correta o usuário.

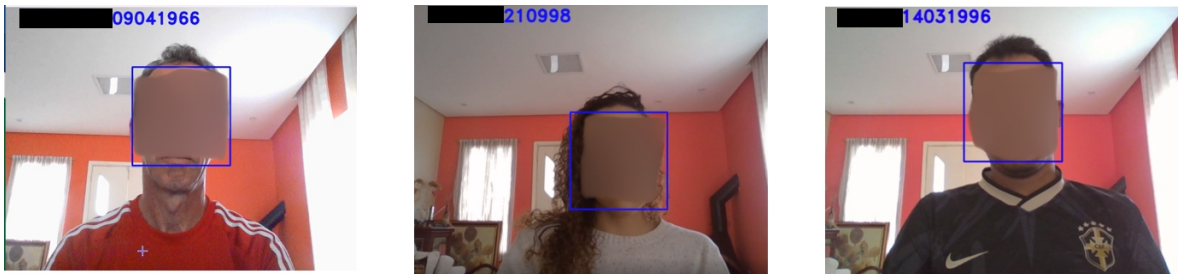

Figura 6.9: Alguns resultados da classificação utilizando câmera com resolução de 720p e condições favoráveis

Figura 6.10: alguns erros observados no cenário 3

6.2.4 Cenário de Teste 4

No último teste, foi aplicado condições desfavoráveis ao reconhecimento facial para analisar como a aplicação se comportaria diante de iluminação precária e resolução menor na câmera.

Como esperado, foram apresentados os piores resultados, onde mesmo realizando intervenções no ambiente, a aplicação não conseguiu identificar de forma correta o indivíduo presente, conforme exposto na figura 6.13, entretanto ainda pode-se considerar um resultado positivo, já que dos 4 erros observados,

em dois casos o sistema conseguiu ajustar a resposta correta.

Figura 6.11: Alguns resultados da classificação utilizando câmera com resolução de 720p e luz artificial

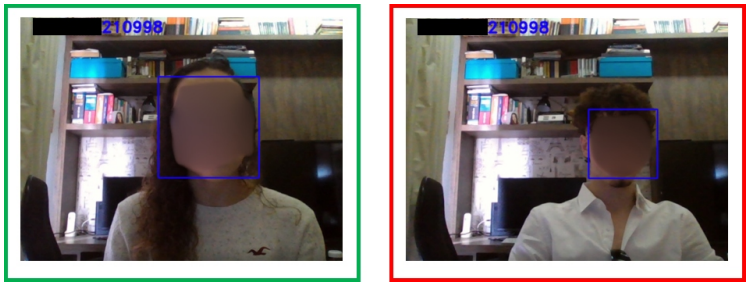

Figura 6.12: Alguns erros observados no cenário 4

6.2.5 Resultados

Conforme exposto, os resultados se mostraram favoráveis a aplicabilidade do sistema de reconhecimento, tendo em vista que mesmo com a variação de cenários, obteve-se a identificação dos indivíduos cadastrados. Conclui-se que condições de luminosidade e resolução das câmeras interferem diretamente na capacidade de identificação, tanto nos cenários de cadastro quanto nos cenários de reconhecimento.

Foi contabilizado, também, a quantidade de vezes que o algoritmo conseguiu chegar ao resultado correto após o erro, onde houve ajustes na posição do indivíduo reconhecido ou no ambiente em que foi realizado o teste. O resultados encontra-se na quinta coluna da tabela 6.1.

Entretanto em alguns cenários, onde não foi possível, mesmo após vários ajustes de luminosidade, chegar ao resultado correto. Vale destacar uma nova variável a ser considerada como fator limitante do sistema, que é o grau de semelhança entre indivíduos, tendo em vista que o resultado apresentado para o indivíduo 1, remete ao indivíduo 2.

Tabela 6.1: Resultados dos testes realizados

Cenários	Total de testes	Acertos imediatos	Erros	Acertos após ajustes
1	10	8	2	2
2	10	6	4	3
3	10	7	3	2
4	10	6	4	2

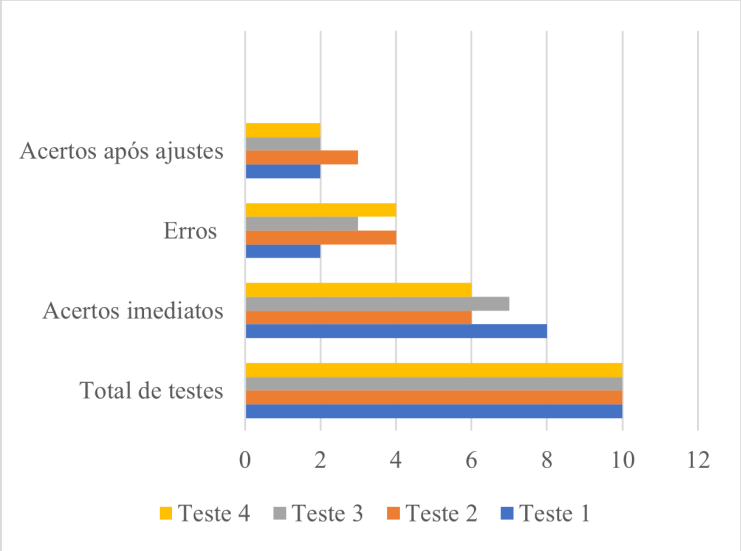

Figura 6.13: Resultados observados

Diante do apuramento dos testes, foi possível quantificar e visualizar a acurácia da efetividade do sistema, considerando resultados expostos em 6.1.

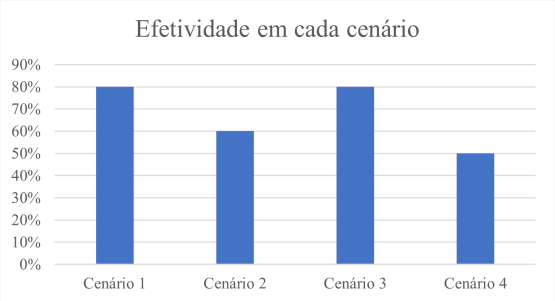

Figura 6.14: Cenário 1: 80% - Cenário 2: 60% - Cenário 3: 80% - Cenário 4: 50%

Figura 6.15: Efetividade considerando a soma dos testes

Por fim, percebe-se que o sistema tem uma operação satisfatória para os testes a qual foi proposto, onde mesmo com os indicativos de erro, foi possível atender aos objetivos anteriormente estabelecidos.

7 CONCLUSÃO

Nesse trabalho foi desenvolvido uma solução leve, com pouco gasto de recursos e consumo de energia, onde o objetivo foi realizar o reconhecimento facial, com cadastro de entrada de indivíduos, em uma arquitetura onde foi otimizado custo e processamento. Ao longo do trabalho foram apresentados conceitos teóricos acerca da arquitetura de *fog computing*, bem como a sua aplicabilidade para atingir o propósito desse projeto.

A aplicação foi hospedada no sistema operacional Windows e desenvolvida com auxílio de bibliotecas como OpenCV, Flask e Scikit-learn, onde foi estruturada para funcionar como um servidor *web*. A partir da solicitação dessa aplicação, os usuários podem realizar o cadastro e realizar o registro do horário dentro do sistema, onde posteriormente os dados poderão ser enviados para a camada de nuvem, para centralização dos registros.

Objetivou-se analisar o comportamento e efetividade da aplicação diante de diferentes cenários, variando luminosidade na realização do cadastro da pessoa, e no registro, variando a qualidade da câmera e a luminosidade do ambiente. Foram 5 pessoas cadastradas, 3 em ambientes favoráveis à iluminação e 2 em ambientes internos, onde a iluminação pode ser prejudicada.

Para validar a arquitetura e a avaliar os resultados da aplicação foram analisados quatro cenários, onde variava-se iluminação do ambiente e a resolução da câmera. A primeira variável, se tornou mais expressiva e determinante na variação do resultado da aplicação do que a resolução das câmeras utilizadas, independente do contexto do cadastro. Percebeu-se, também, certas instabilidades em pessoas que possuem mais semelhanças físicas, mas foi um comportamento esperado, diante das condições impostas nos testes onde alguns integrantes fazem parte do mesmo grupo familiar.

Na implementação da arquitetura, no que diz respeito a *fog computing*, foi possível observar uma resposta satisfatória para o experimento proposto, porém com algumas ressalvas, como recursos de hardware, que produziram um fator limitante para o desempenho da aplicação tanto no âmbito do cadastro como no reconhecimento. Foi necessário ajustar versionamento das bibliotecas e linguagem e adaptar o código de forma que o desempenho fosse mais fiel possível ao resultado esperado. Sendo assim, a implementação da comunicação com a nuvem se tornou um importante aliado para otimizar o armazenamento e atingir os objetivos propostos.

Por fim, foi possível completar todos os objetivos específicos estabelecidos na seção de Introdução, além de avaliar e estabelecer bons requisitos de reconhecimento facial com foco na arquitetura em *fog computing*.

7.1 TRABALHOS FUTUROS

Sugere-se como trabalhos futuros:

- Teste da aplicação em diferentes circunstâncias, tais como:
  - Sistemas Operacionais de código aberto;
  - Sistemas Operacionais de código privado, incluindo o apresentado neste experimento, desde que apresente alguma alteração seja de código, de testes e afins;
  - Provedores de nuvem pública ou privada;
  - Ambientes de *multicloud*;
  - Novos dispositivos na camada de *fog computing*.
- Interface de autenticação para que apenas pessoas autorizadas a realizarem cadastro os façam;
- Adição de protocolos de criptografia e segurança;
- Testes de análise de desempenho operacional da ferramenta;
- Testes em outras linguagens de programação mais ou menos otimizadas;
- Testes em ambiente que não envolvam arquitetura de *fog computing* para analisar a responsividade do código.
- Uma validação sistêmica junto ao usuário com uma submissão ao comitê de ética.

Há outras possibilidades de análises e testes que podem ser sugeridas, sempre explorando a criatividade e curiosidade de quem deseja explorar a técnica de reconhecimento facial. Os itens apresentados anteriormente não são limitantes e tem o propósito de fomentar as mais diversas possibilidades de analise e recriação do experimento.

REFERÊNCIAS BIBLIOGRÁFICAS

1 APLICATIVOS de ponto eletrônico ajudam empresas a controlar a jornada durante o home office - Época Negócios | Carreira. <https://epocanegocios.globo.com/Carreira/noticia/2020/07/aplicativos-de-ponto-eletronico-ajudam-empresas-controlar-jornada-durante-o-home-office.html>. (Accessed on 11/22/2022).

2 IORGA, M.; FELDMAN, L.; BARTON, R.; MARTIN, M.; GOREN, N.; MAHMOUDI, C. *The nist definition of fog computing*. [S.l.], 2017.

3 BARELLI, F. *Introdução à visão computacional: Uma abordagem prática com Python e OpenCV*. [S.l.]: Editora Casa do Código, 2018.

4 OVERVIEW of Distributed Web Server. | Download Scientific Diagram. <https://www.researchgate.net/figure/Overview-of-Distributed-Web-Server_fig1_220900351>. (Accessed on 02/17/2023).

5 OPENCV: classificador em cascata. <https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html>. (Accessed on 02/28/2023).

6 CASCATAS de Haar, explicadas. Uma breve introdução em Haar... | por Aditya Mittal | Analytics Vidhya | Médio. <https://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d>. (Accessed on 02/28/2023).

7 NEAREST Neighbors Classification — scikit-learn 1.2.1 documentation. <https://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html#sphx-glr-auto-examples-neighbors-plot-classification-py>. (Accessed on 02/28/2023).

8 RASPBERRY Pi 4 Computer Model B 4 Gb RAM : Amazon.com.br: Computadores e Informática. <https://www.amazon.com.br/Raspberry-Pi-Computer-Model-RAM/dp/B07TC2BK1X>. (Accessed on 02/17/2023).

9 MICROSOFT é reconhecida como líder no Quadrante Mágico™ da Gartner® de 2021 para Plataformas de Serviços de Conteúdo. <https://www.microsoft.com/pt-br/microsoft-365/blog/2022/03/10/microsoft-recognized-as-a-leader-in-the-2021-gartner-magic-quadrant-for-content-services-platforms/>. (Accessed on 02/17/2023).

10 2022 Gartner Cloud Infrastructure and Platform Services (CIPS) MQ | Google Cloud. <https://cloud.google.com/resources/gartner-cloud-infrastructure-and-platform-services-2022>. (Accessed on 02/17/2023).

11 LIMA, A. R.; NUNES, C. Avm–faculdade integrada pós-graduação lato sensu.

12 COLOMBO, F. J.; NETO, B. B.; BARROS, L. d. J. R. de. Um estudo sobre a biometria a study on biometrics. *Revista Interface Tecnológica*, v. 10, n. 1, p. 37–44, 2013.

13 MARCONDES, J. S. Biometria, sistema biométrico: O que é, como funciona?

14 MILANO, D. de; HONORATO, L. B. Visao computacional. *Faculdade de Tecnologia, Universidade Estadual de Campinas*, 2010.

15 PRACIANO, B. J.; FILHO, F. L. de C.; LUCAS, M.; MARTINS, D. F.; SILVA, D. A. da; JÚNIOR, R. T. de S. Segurança do ambiente usando dispositivo iot com processamento distribuído. In: *Atas das Conferências Ibero-Americanas WWW/Internet 2019 e Computação Aplicada 2019*. [S.l.: s.n.], 2019. p. 163–170.

16 FISCHER, I. A. et al. Gerenciamento proativo baseado na análise computacional dos nodos fog que integram smart classroom. Universidade Federal de Santa Maria, 2020.

17 MELL, P.; GRANCE, T. et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National . . . , 2011.

18 DASTJERDI, A. V.; BUYYA, R. Fog computing: Helping the internet of things realize its potential. *Computer*, IEEE, v. 49, n. 8, p. 112–116, 2016.

19 BRASIL. Decreto-lei nº 5.452, de 1º de maio de 1943: Aprova a consolidação das leis do trabalho. *Diário Oficial [da] República Federativa do Brasil*, Brasília, DF, 1943. ISSN 2596-0253. Disponível em: <https://www.planalto.gov.br/ccivil_03/decreto-lei/del5452.htm>.

20 ESCOLAS adotam chamadas digitais e aumentam frequência dos alunos. <https://www.unimestre.com/escolas-adotam-chamadas-digitais-e-aumentam-frequencia-dos-alunos/>. (Accessed on 11/22/2022).

21 TAVARES, H. L. Identificação de pessoas baseada em características antropométricas e de marcha extraídas de poses 2d. Universidade Estadual Paulista (UNESP), 2021.

22 CAO, Z.; SIMON, T.; WEI, S.-E.; SHEIKH, Y. Realtime multi-person 2d pose estimation using part affinity fields. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 7291–7299.

23 KREISS, S.; BERTONI, L.; ALAHI, A. Pifpaf: Composite fields for human pose estimation. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. [S.l.: s.n.], 2019. p. 11977–11986.

24 ABADI ASHISH AGARWAL, P. B. E. B. Z. C. C. C. G. S. C. A. D. J. D. M. D. S. G. I. G. A. H. G. I. M. I. Y. J. R. J. L. K. M. K. J. L. D. M. R. M. S. M. D. M. C. O. M. S. J. S. B. S. I. S. K. T. P. T. V. V. V. V. F. V. O. V. P. W. M. W. M. W. Y. Y. M.; ZHENG, X. *tensorflow-whitepaper2015.pdf*. <https://www.tensorflow.org/extras/tensorflow-whitepaper2015.pdf>. (Accessed on 08/17/2022).

25 TAN, D.; HUANG, K.; YU, S.; TAN, T. Efficient night gait recognition based on template matching. In: IEEE. *18th International Conference on Pattern Recognition (ICPR'06)*. [S.l.], 2006. v. 3, p. 1000–1003.

26 WANG, Y.; YU, S.; WANG, Y.; TAN, T. Gait recognition based on fusion of multi-view gait sequences. In: SPRINGER. *International Conference on Biometrics*. [S.l.], 2006. p. 605–611.

27 DIAS, B. S. S.; PORTO, I.; FILHO, F. L. de C.; MATA, R. Z. da; ALMEIDA, L. de O.; MENDONÇA, F. L.; JR, R. T. de S. et al. Sistema para a identificação de aglomerações operando em redes iot e fog computing. *Revista Ibérica de Sistemas e Tecnologias de Informação*, Associação Ibérica de Sistemas e Tecnologias de Informacao, n. E47, p. 300–311, 2022.

28 BIRLA, D. *GitHub - deepak112/Social-Distancing-AI: Tool to moniter social distancing using CCTV feeds, videos. Can be used at public places and workplace.* 2020. <https://github.com/deepak112/Social-Distancing-AI>. (Accessed on 08/13/2022).

29 HOW to check if a given point lies inside or outside a polygon? - GeeksforGeeks. <https://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-inside-a-polygon/#:~:text=Draw%20a%20horizontal%20line%20to,true%2C%20then%20point%20lies%20outside.> (Accessed on 08/17/2022).

30 UNIVERSITY of Reading (2009). PETS 2009 Benchmark Data. University of Reading. <http://cs.binghamton.edu/~mrldata/pets2009>. (Accessed on 08/17/2022).

31 DUTRA, B. V.; MARTINS, L. M. C. E. Hids by signature for embedded devices in iot networks. In: SERVICIO DE PUBLICACIONES. *Actas de las V Jornadas Nacionales de Ciberseguridad: junio 5-7, 2019. Cáceres*. [S.l.], 2019. p. 53–61.

32 GEITGEY A. NAZARIO, J. *GitHub - ageitgey/face_recognition: The world's simplest facial recognition api for Python and the command line*. 2017. <https://github.com/ageitgey/face_recognition/>. (Accessed on 08/21/2022).

33 AMOS, B.; LUDWICZUK, B.; SATYANARAYANAN, M. et al. Openface: A general-purpose face recognition library with mobile applications. *CMU School of Computer Science*, CMU School of Computer Science Pittsburgh, PA, v. 6, n. 2, p. 20, 2016.

34 KING, D. E. Dlib-ml: A machine learning toolkit. *The Journal of Machine Learning Research*, JMLR. org, v. 10, p. 1755–1758, 2009.

35 DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. In: IEEE. *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. [S.l.], 2005. v. 1, p. 886–893.

36 KAZEMI, V.; SULLIVAN, J. One millisecond face alignment with an ensemble of regression trees. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2014. p. 1867–1874.

37 SCHROFF, F.; KALENICHENKO, D.; PHILBIN, J. Facenet: A unified embedding for face recognition and clustering. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2015. p. 815–823.

38 RÜPING, S. *SVM kernels for time series analysis*. [S.l.], 2001.

39 KORGUT, D.; BERNARDO, W. *Sistema de monitoramento seguro por reconhecimento facial baseado em Internet das Coisas*. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2019.

40 HOME - OpenCV. 2019. <https://opencv.org/>. (Accessed on 08/21/2022).

41 VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: IEEE. *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*. [S.l.], 2001. v. 1, p. I–I.

42 ANDRADE, M. B.; SILVA, F. F. da; MARTINS, L. M. e; OLIVEIRA, H. W. d. S.; MENDONÇA, F. L. de; SOUSA, R. T. de. Iot fog-based image matching monitoring system for physical access control through ip camera devices. In: IEEE. *2020 Workshop on Communication Networks and Power Systems (WCNPS)*. [S.l.], 2020. p. 1–6.

43 FACE-RECOGNITION · PyPI. 2020. <https://pypi.org/project/face-recognition/>. (Accessed on 08/21/2022).

44 BALLA, P. B.; JADHAO, K. Iot based facial recognition security system. In: IEEE. *2018 international conference on smart city and emerging technology (ICSCET)*. [S.l.], 2018. p. 1–4.

45 OTHMAN, N. A.; AYDIN, I. A face recognition method in the internet of things for security applications in smart homes and cities. In: IEEE. *2018 6th International Istanbul Smart Grids and Cities Congress and Fair (ICSG)*. [S.l.], 2018. p. 20–24.

46 LEINHART, R.; MAYDT, J. An extended set of haar-like features. In: *Proc. Int. Conf. on Image Processing*. [S.l.: s.n.], 2002. p. 900–903.

47 AHONEN, T.; HADID, A.; PIETIKAINEN, M. Face description with local binary patterns: Application to face recognition. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 28, n. 12, p. 2037–2041, 2006.

48 WANG, P.; LIN, W.-H.; WU, B.-H.; CHAO, K.-M.; LO, C.-C. A cross-age face recognition approach using fog computing architecture for user authentication on mobile devices. In: IEEE. *2018 IEEE 15th International Conference on e-Business Engineering (ICEBE)*. [S.l.], 2018. p. 86–93.

49 MODELS/RESEARCH/SLIM/NETS/MOBILENET at master · tensorflow/models. <https://github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet>. (Accessed on 08/21/2022).

50 CHEN, B.-C.; CHEN, C.-S.; HSU, W. H. Face recognition and retrieval using cross-age reference coding with cross-age celebrity dataset. *IEEE Transactions on Multimedia*, IEEE, v. 17, n. 6, p. 804–815, 2015.

51 FILHO, F. L. de C.; ROCHA, R. L.; ABBAS, C. J.; MARTINS, L. M. E.; CANEDO, E. D.; SOUSA, R. T. de. Qos scheduling algorithm for a fog iot gateway. In: IEEE. *2019 Workshop on Communication Networks and Power Systems (WCNPS)*. [S.l.], 2019. p. 1–6.

52 PRADO, D. S. d.; ALMEIDA, L. C. d.; MARTINS, L.; MENDONÇA, F. L. de; SOUSA, R. T. d. et al. Design of a fog controller to provide an iot middleware with hierarchical interaction capability. In: SPRINGER. *International Conference on Information Technology & Systems*. [S.l.], 2021. p. 280–292.

53 OTHMAN, N. A.; AYDIN, I. A new iot combined body detection of people by using computer vision for security application. In: *2017 9th International Conference on Computational Intelligence and Communication Networks (CICN)*. [S.l.: s.n.], 2017. p. 108–112.

54 MOHAN, H.; ANITHA, S.; CHAI, R.; LING, S. H. Edge artificial intelligence: Real-time noninvasive technique for vital signs of myocardial infarction recognition using jetson nano. *Advances in Human-Computer Interaction*, Hindawi, v. 2021, 2021.

55 AYDIN, I.; OTHMAN, N. A. A new iot combined face detection of people by using computer vision for security application. In: *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*. [S.l.: s.n.], 2017. p. 1–6.

56 FAJARDO, F. T. Desenvolvimento de uma aplicação web com detecção e reconhecimento facial. Universidade Federal de Santa Maria, 2019.

57 MONTAGNER, D. H. Sistema de reconhecimento e identificação facial em dispositivo embarcado. 002, 2020.

58 ROZAR, J. L.; FRANCISCO, A. M. Reconhecimento facial com técnicas de machine learning. *Sistemas de Informação-Pedra Branca*, 2020.

59 RIBEIRO, C. F. C. Gateway redundante iot com uso de fog computing. 2019.

60 PEREIRA, V.; DIAS, S.; QUEIROZ, K. de. Sistema inteligente para controle de acesso e monitoramento de multiplos ambientes (class control). In: SBC. *Anais Estendidos do IX Simpósio Brasileiro de Engenharia de Sistemas Computacionais*. [S.l.], 2019. p. 25–30.

61 SANTOS, B. P.; SILVA, L. A.; CELES, C. S.; NETO, J. B. B.; PERES, B. S.; VIEIRA, M. A. M.; VIEIRA, L. F. M.; GOUSSEVSKAIA, O. N.; LOUREIRO, A. A. Internet das coisas: da teoria à prática. 2016.

62 O Que Falta Para Impulsionar O Mercado De IoT No Brasil? <https://www.connectdata.net/post/o-que-falta-para-impulsionar-o-mercado-de-iot-no-brasil>. (Accessed on 01/12/2023).

63 VOAS, J.; KUHN, R.; LAPLANTE, P.; APPLEBAUM, S. Internet of things (iot) trust concerns. *NIST Tech. Rep*, v. 1, p. 1–50, 2018.

64 LUAN, T. H.; GAO, L.; LI, Z.; XIANG, Y.; WEI, G.; SUN, L. Fog computing: Focusing on mobile users at the edge. *arXiv preprint arXiv:1502.01815*, 2015.

65 ARKIAN, H. R.; DIYANAT, A.; POURKHALILI, A. Mist: Fog-based data analytics scheme with cost-efficient resource provisioning for iot crowdsensing applications. *Journal of Network and Computer Applications*, Elsevier, v. 82, p. 152–165, 2017.

66 AAZAM, M.; HUH, E.-N. Fog computing micro datacenter based dynamic resource estimation and pricing model for iot. In: IEEE. *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*. [S.l.], 2015. p. 687–694.

67 NAHA, R. K.; GARG, S.; GEORGAKOPOULOS, D.; JAYARAMAN, P. P.; GAO, L.; XIANG, Y.; RANJAN, R. Fog computing: Survey of trends, architectures, requirements, and research directions. *IEEE access*, IEEE, v. 6, p. 47980–48009, 2018.

68 HU, P.; DHELIM, S.; NING, H.; QIU, T. Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of network and computer applications*, Elsevier, v. 98, p. 27–42, 2017.

69 BOTTA, A.; DONATO, W. D.; PERSICO, V.; PESCAPÉ, A. Integration of cloud computing and internet of things: a survey. *Future generation computer systems*, Elsevier, v. 56, p. 684–700, 2016.

70 RUDEK, M.; COELHO, L. d. S.; JR, O. C. Visão computacional aplicada a sistemas produtivos: fundamentos e estudo de caso. *XXI Encontro Nacional de Engenharia de Produção-2001, Salvador*, 2001.

71 ZHENG, H.; ZHANG, G.; ZHANG, L.; WANG, Q.; LI, H.; HAN, Y.; XIE, L.; YAN, Z.; LI, Y.; AN, Y. et al. Comprehensive review of web servers and bioinformatics tools for cancer prognosis analysis. *Frontiers in oncology*, Frontiers Media SA, v. 10, p. 68, 2020.

72 DETECçãO e reconhecimento facial usando OpenCV e KNN do zero | por Tanvi Penumudy | Analytics Vidhya | Médio. <https://medium.com/analytics-vidhya/face-detection-and-recognition-using-opencv-and-knn-from-scratch-dcba9b0fd07d>. (Accessed on 02/28/2023).

73 1.6. Nearest Neighbors — scikit-learn 1.2.1 documentation. <https://scikit-learn.org/stable/modules/neighbors.html#classification>. (Accessed on 02/28/2023).

74 ABOUT - OpenCV. <https://opencv.org/about/>. (Accessed on 01/29/2023).

75 AHMED, A. S.; NOORI, Z. H.; ATIYAHC, J. M. Dynamic object detection and evaluation patterns using opencv.

76 WHAT is Flask Python - Python Tutorial. <https://pythonbasics.org/what-is-flask-python/>. (Accessed on 01/29/2023).

77 WELCOME to Flask — Flask Documentation (2.2.x). <https://flask.palletsprojects.com/en/2.2.x/>. (Accessed on 01/29/2023).

78 GHIMIRE, D. Comparative study on python web frameworks: Flask and django. 2020.

79 COPPERWAITE, M.; LEIFER, C. *Learning flask framework*. [S.l.]: Packt Publishing Ltd, 2015.

80 STACK Overflow Developer Survey 2022. <https://survey.stackoverflow.co/2022/#most-loved-dreaded-and-wanted-webframe-want>. (Accessed on 01/29/2023).

81 BISONG, E. et al. *Building machine learning and deep learning models on Google cloud platform*. [S.l.]: Springer, 2019.

82 STANČIN, I.; JOVIĆ, A. An overview and comparison of free python libraries for data mining and big data analysis. In: IEEE. *2019 42nd International convention on information and communication technology, electronics and microelectronics (MIPRO)*. [S.l.], 2019. p. 977–982.

83 GHOLIZADEH, S. Top popular python libraries in research. *Authorea Preprints*, Authorea, 2022.

84 MCKINNEY, W. et al. pandas: a foundational python library for data analysis and statistics. *Python for high performance and scientific computing*, Seattle, v. 14, n. 9, p. 1–9, 2011.

85 HAO, J.; HO, T. K. Machine learning made easy: a review of scikit-learn package in python programming language. *Journal of Educational and Behavioral Statistics*, SAGE Publications Sage CA: Los Angeles, CA, v. 44, n. 3, p. 348–361, 2019.

86 PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V. et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, JMLR. org, v. 12, p. 2825–2830, 2011.

87 BALON, B.; SIMIć, M. Using raspberry pi computers in education. In: *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. [S.l.: s.n.], 2019. p. 671–676.

88 SILVEIRA, R. B. *História do Microsoft® Windows®*.

89 QUAL o sistema operacional de PC mais usado do mundo? - Canaltech. <https://canaltech.com.br/software/qual-o-sistema-operacional-de-pc-mais-usado-do-mundo-224432/>. (Accessed on 02/17/2023).

90 GUPTA, B.; MITTAL, P.; MUFTI, T. A review on amazon web service (aws), microsoft azure & google cloud platform (gcp) services. In: *Proceedings of the 2nd International Conference on ICT for Digital, Smart, and Sustainable Development, ICIDSSD 2020, 27-28 February 2020, Jamia Hamdard, New Delhi, India*. [S.l.: s.n.], 2021.

91 BISONG, E.; BISONG, E. An overview of google cloud platform services. *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*, Springer, p. 7–10, 2019.

92 CLOUD Storage | Google Cloud. <https://cloud.google.com/storage#:~:text=Cloud%20Storage%20offers%20secure%2Dby,policy%20locks%2C%20and%20signed%20URLs>. (Accessed on 02/14/2023).

93 HOW to Install Windows 10 on Raspberry Pi 4 [Full Guide]. <https://www.partitionwizard.com/clone-disk/windows-10-on-raspberry-pi-4.html>. (Accessed on 02/28/2023).

94 OBJECT Detection Using Haar Cascade: OpenCV. <https://www.analyticsvidhya.com/blog/2022/04/object-detection-using-haar-cascade-opencv/#:~:text=What%20are%20Haar%20Cascades%3F,%2C%20buildings%2C%20fruits%2C%20etc>. (Accessed on 02/27/2023).

95 K-VIZINHOS Mais Próximos (KNN) - Aprender Ciência de Dados. <https://aprenderdatascience.com/k-vizinhos-mais-proximos-knn/#:~:text=K%2DVizinhos%20Mais%20Pr%C3%B3ximos%20(KNN)%20%C3%A9%20um%20dos%20algoritmos,lo%20para%20tarefas%20de%20classifica%C3%A7%C3%A3o.> (Accessed on 02/27/2023).

96 OPENCV: Cascade Classifier. <https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html>. (Accessed on 02/18/2023).

97 WELCOME to Flask — Flask Documentation (2.2.x). <https://flask.palletsprojects.com/en/2.2.x/#user-s-guide>. (Accessed on 02/18/2023).

98 BáSICO do OpenCV com Python – mostrar vídeo e webcam – Professor Benjamin. <https://www.galirows.com.br/meublog/blog/basico-opencv-python-video-webcam/>. (Accessed on 02/18/2023).

ANEXOS

I.1 INSTALAÇÃO DO WINDOWS 10 NA RASPBERRY PI

Para realizar o download da imagem do Windows 10 para a Raspberry 4, basta seguir o tutorial no seguinte link: https://www.partitionwizard.com/clone-disk/windows-10-on-raspberry-pi-4.html. Nesse tutorial mostra em alguns passos como instalar uma imagem do Windows em um cartão micro SD.

1. Baixar ferramentas e arquivos necessários: no próprio computador pessoal foi instalado duas coisas importantes: a ferramenta WoR e o arquivo UUP. WoR é uma ferramenta prática desenvolvida para fazer o flash de uma imagem ARM do Windows 10 Raspberry Pi para um cartão microSD e o arquivo UUP pode ser usado para criar o arquivo Windows 10 Raspberry Pi ISO no seu computador.

2. Criação do arquivo ISO Raspberry Pi do Windows 10: através do arquivo UUP é realizado o download do copilador ISO. Após o download, a ferramenta começará a criar o arquivo Windows 10 Raspberry Pi ISO.

3. Formatação do cartão SD para FAT32: com a criação do arquivo ISO realizado com sucesso, é possível fazer o flash do arquivo no cartão SD usando a ferramenta WoR. Nesse caso, é necessário utilzarum cartão SD de 16 GB ou mais e certificar-se de que está formatado para FAT32.

4. Flash do arquivo ISO para o cartão SD: com o cartão SD conectado ao computador, utiliza-se o arquivo WoR.exe para transferência da ISO para o cartão.

5. Instalação do Windows 10 no Raspberry Pi 4: com a Raspberry Pi desligada, foi inserido o cartão microSD, onde segue-se o processo de instalação padrão do Windows, seguindo instruções da tela.

I.2 PACOTES INSTALADOS PARA EXECUÇÃO DA APLICAÇÃO

Antes da execução do arquivo app.py, responsável por executar toda a lógica e funções do sistema, é necessário instalar alguns pacotes. O primeiro é o pacote do Python, seguindo o tutorial presente em https://python.org.br/instalacao-windows/, com alguns passos é possível configurar e baixar o pacote python e pip.

Posteriormente, com uso do pip, que é um sistema de gerenciamento de pacotes padrão usado para instalar e gerenciar pacotes de software escritos em Python, é executado o seguinte comando:

```
pip install -r requirements.txt
```

Sendo que nesse arquivo de texto encontra-se os pacotes necessários para a implementação das bibliotecas citadas no capítulo 4.

Tabela 1: Lista de pacotes baixados

Pacote	Versão	Link da definição (explicação)
cachetools	5.3.0	https://pypi.org/project/cachetools/
certifi	2022.12.7	https://pypi.org/project/certifi/
charset-normalizer	3.0.1	https://pypi.org/project/charset-normalizer/
click	8.1.3	https://pypi.org/project/click/
colorama	0.4.6	https://pypi.org/project/colorama/
Flask	2.2.2	https://pypi.org/project/Flask/
google-api-core	2.11.0	https://pypi.org/project/google-api-core/
google-auth	2.16.0	https://pypi.org/project/google-auth/
google-cloud-core	2.3.2	https://pypi.org/project/google-cloud-core/
google-cloud-storage	2.7.0	https://pypi.org/project/google-cloud-storage/
google-crc32c	1.5.0	https://pypi.org/project/google-crc32c/
google-resumable-media	2.4.1	https://pypi.org/project/google-resumable-media/
googleapis-common-protos	1.58.0	https://pypi.org/project/googleapis-common-protos/
idna	3.4	https://pypi.org/project/idna/
importlib-metadata	6.0.0	https://pypi.org/project/importlib-metadata/
itsdangerous	2.1.2	https://pypi.org/project/itsdangerous/
Jinja2	3.1.2	https://pypi.org/project/Jinja2/
joblib	1.2.0	https://pypi.org/project/joblib/
MarkupSafe	2.1.1	https://pypi.org/project/MarkupSafe/
numpy	1.24.1	https://pypi.org/project/numpy/
opencv-python	4.7.0.68	https://pypi.org/project/opencv-python/
pandas	1.5.2	https://pypi.org/project/pandas/
protobuf	4.21.12	https://pypi.org/project/protobuf/
pyasn1	0.4.8	https://pypi.org/project/pyasn1/
pyasn1-modules	0.2.8	https://pypi.org/project/pyasn1-modules/
python-dateutil	2.8.2	https://pypi.org/project/python-dateutil/
pytz	2022.7	https://pypi.org/project/pytz/
requests	2.28.2	https://pypi.org/project/requests/
rsa	4.9	https://pypi.org/project/rsa/
scikit-learn	1.2.0	https://pypi.org/project/scikit-learn/
scipy	1.10.0	https://pypi.org/project/scipy/
six	1.16.0	https://pypi.org/project/six/
threadpoolctl	3.1.0	https://pypi.org/project/threadpoolctl/
urllib3	1.26.14	https://pypi.org/project/urllib3/
Werkzeug	2.2.2	https://pypi.org/project/Werkzeug/
zipp	3.11.0	https://pypi.org/project/zipp/

I. DETALHAMENTO DO CÓDIGO

I.1 MÉTODOS PARA INTERAÇÃO COM O GOOGLE CLOUD STORAGE

A implementação começa com a criação de uma conta no Google Cloud e registro das credenciais para a utilização e manipulação do storage por meio de aplicações.

O Google Cloud disponibiliza um arquivo JSON com as credenciais da chave gerada. Com este arquivo adicionado no projeto como "service-key-cloud.json"e referenciado como variável de ambiente com o identificador "GOOGLE_APPLICATION_CREDENTIALS"para que o cliente utilize os recursos atrelados a essa chave através da biblioteca por meio da biblioteca "google-cloud-storage".

A implementação do serviço de nuvem começa com a definição da classe "GCStorage"que define métodos para a manipulação dos dados dentro do *bucket* na nuvem, seguindo a recomendação apresentada na ferramenta do Google Cloud.

A seguir está os códigos responsáveis pela interação com o Google Cloud Storage.

Para a declaração da variável de ambiente que possui as credenciais de acesso a API do Cloud Storage, e a implementação da classe que possui os métodos de interação com o *bucket*, deve-se adicionar o seguinte bloco de código:

app.py home.html

app.py > ...

```python
#### Cloud

os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = 'service-key-cloud.json'

STORAGE_CLASSES = ('STANDARD', 'NEARLINE', 'COLDLINE', 'ARCHIVE')


class GCStorage:
    def __init__(self, storage_client):
        self.client = storage_client

    def create_bucket(self, bucket_name, storage_class, bucket_location='US'):
        bucket = self.client.bucket(bucket_name)
        bucket.storage_class = storage_class
        return self.client.create_bucket(bucket, bucket_location)

    def get_bucket(self, bucket_name):
        return self.client.get_bucket(bucket_name)

    def list_buckets(self):
        buckets = self.client.list_buckets()
        return [bucket.name for bucket in buckets]

    def upload_file(self, bucket, blob_destination, file_path):
        file_type = file_path.split('.')[-1]
        if file_type == 'csv':
            content_type = 'text/csv'
        elif file_type == 'psd':
            content_type = 'image/vnd.adobe.photoshop'
        else:
            content_type = mimetypes.guess_type(file_path)[0]
        blob = bucket.blob(blob_destination)
        blob.upload_from_filename(file_path, content_type=content_type)
        return blob

    def list_blobs(self, bucket_name):
        return self.client.list_blobs(bucket_name)
```

Figura I.1: Variável de ambiente e Métodos

O método "create_bucket()"que recebe como parâmetro o nome do *bucket*, a classe que esse *bucket* pertence, nesse projeto apenas a classe STANDARD é utilizada. O último parâmetro é a localização da nuvem em que será criado definido por "bucket_location"e para esse trabalho foi definido como "US".

Uma vez definido os parâmetros o método cria o *bucket* através do método da API da biblioteca "google-cloud-storagecreate_bucket()".

O método "get_bucket()"recupera um *bucket* específico e recebe como parâmetro a própria instância e o nome do *bucket*.

Ja o método "list_buckets()"recebe a própria instância como parâmetro e chama o método da API JSON da biblioteca google-cloud-storage "list_buckets()"que retorna uma lista de *buckets* associado ao cliente.

O método "upload_file"é utilizado para fazer upload dos arquivos ".csv"contidos na pasta "Attendance", ele recebe como parâmetro o *bucket* selecionado, o local que será atribuído dentro do *bucket* no Google Cloud Storage denominado "blob_destination"e o caminho do arquivo local que será enviado pro *bucket*.

Tendo isso em vista, o método separa o arquivo da sua extensão por meio do método "split()"do python,

e checa por meio de condicionais qual a tipagem do arquivo para ser informado no header do upload através do content_type, nessa checagem é analisado se a extensão é ".csv"atribuindo "text/csv"ao content_type, se a extensão é ".psd"é atribuindo "image/vnd.adobe.photoshop"ao content_type, e caso não seja nenhuma dessas duas o método "guess_type"da biblioteca "mimetypes"é utilizado para fazer essa conversão automaticamente.

Após esse processo cria-se a variável *blob* que é um acrônimo para "Binaty Large Object"e funciona com o mesmo conceito de objeto dentro do Google Cloud Storage passando como parâmetro a localização do objeto que nesse caso é o "blob_destination"e em seguida utiliza o método "upload_from_filename"da biblioteca "google-cloud-storage"que faz upload de um *blob* para o *bucket* recebendo como parâmetro o nome do arquivo, nesse caso sendo o "file_path", e o content_type definido previamente.

E como último método utilizado na manipulação de dados na nuvem é o método "list_blobs()"que retorna uma lista de *blobs*s dentro do *bucket*, ele recebe como parâmetro a própria instância e o nome do *bucket* referenciado como "bucket_name".

I.2 VARIAVEIS RELACIONADAS A INTERAÇÃO DO GOOGLE CLOUD STORAGE

```
53  #### 1. Prepare Variables
54
55  working_dir = pathlib.Path.cwd()
56  files_folder = working_dir.joinpath('Attendance')
57  downloads_folder = working_dir.joinpath('Downloaded')
58  bucket_name = 'tcc_bucket_heloisa_pedro'
59
60  #### 2. Construct GCSorage Instance
61
62  storage_client = storage.Client()
63  gcs = GCStorage(storage_client)
64
65  #### 3. Create tcc_bucket Cloud Storage Bucket
66
67  if not bucket_name in gcs.list_buckets():
68      gcs.create_bucket('tcc_bucket_heloisa_pedro', STORAGE_CLASSES[0])
69  else:
70      bucket_gcs = gcs.get_bucket(bucket_name)
71
```

Figura I.2: Variáveis Auxiliares de Preparação

Foi definida algumas variáveis para integração dos ambientes:

- "working_dir": recebe o caminho do diretório que o programa é executado através do método "pathlib.Path.cwd()";
- "files_folder”: recebe o caminho da pasta "Attendance"que contém todos os arquivos das frequências armazenados localmente.

- "downloads_folder”: indica o caminho para a pasta "Downloaded"que armazena os arquivos de frequência baixados do *bucket*;

- "bucket_name": recebe o nome do *bucket* que será criado, no caso desse projeto recebeu o nome de "tcc_bucket_heloisa_pedro".

- "storage_client": é utilizado para referenciar um objeto da classe "Client"do Google Cloud, que quando criado utiliza as credenciais fornecidas pra definir o cliente que possui permissão para manipular os *bucket*s .

Tendo isso em vista, o código começa a ser executado utilizando o primeiro método da classe definida "GCStorage"chamado "__init__()"que é o construtor de objetos da própria classe, ele recebe "self"que é a instanciação da própria classe e recebe uma variável "storage_client", que nesse caso, é utilizado para referenciar um objeto da classe "Client"do Google Cloud, que quando criado utiliza as credenciais fornecidas pra definir o cliente que possui permissão para manipular os *bucket*s.

Uma vez definido esse método, ele é utilizado no código em que a instância do objeto da classe "Client"é feita através de "storage_client = storage.Client()", e utilizando esse objeto instancia-se um outro objeto da classe definida "GCStorage"feito com "gcs = GCStorage(storage_client)"e armazenado na variável "gcs".

Em seguida, verifica-se se existe um *bucket* no Storage com o nome definido no "bucket_name"através do método "list_buckets()", e então é verificado o nome de cada *bucket* por um laço de repetição "for". Caso não exista nenhum *bucket* com o nome definido no "bucket_name"um novo *bucket* é criado com esse nome utilizando o método "create_bucket()", e se ja existir o *bucket* é recuperado pelo método "get_bucket()".

I.3 DESENVOLVIMENTO DO SERVIDOR WEB

As figuras a seguir são funções implementadas para o funcionamento das ferramentas dentro da aplicação que serão explicadas no momento em que elas são utilizadas para ficar mais claro sua responsabilidade dentro do código.

```python
#### Defining Flask App
app = Flask(__name__)


#### Saving Date today in 2 different formats
datetoday = date.today().strftime("%d_%m_%y")
datetoday2 = date.today().strftime("%d-%B-%Y")


#### Initializing VideoCapture object to access WebCam
face_detector = cv2.CascadeClassifier('static/haarcascade_frontalface_default.xml')
cap = cv2.VideoCapture(0)


#### If these directories don't exist, create them
if not os.path.isdir('Attendance'):
    os.makedirs('Attendance')
if not os.path.isdir('static/faces'):
    os.makedirs('static/faces')
if f'Attendance-{datetoday}.csv' not in os.listdir('Attendance'):
    with open(f'Attendance/Attendance-{datetoday}.csv','w') as f:
        f.write('Name,Roll,Time')


#### get a number of total registered users
def totalreg():
    return len(os.listdir('static/faces'))


#### extract the face from an image
def extract_faces(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    face_points = face_detector.detectMultiScale(gray, 1.3, 6)
    return face_points


```

Figura I.3: Funções Implementadas na Aplicação pt.1

```python
#### Identify face using ML model
def identify_face(facearray):
    model = joblib.load('static/face_recognition_model.pkl')
    return model.predict(facearray)


#### A function which trains the model on all the faces available in faces folder
def train_model():
    faces = []
    labels = []
    userlist = os.listdir('static/faces')
    for user in userlist:
        for imgname in os.listdir(f'static/faces/{user}'):
            img = cv2.imread(f'static/faces/{user}/{imgname}')
            resized_face = cv2.resize(img, (50, 50))
            faces.append(resized_face.ravel())
            labels.append(user)
    faces = np.array(faces)
    knn = KNeighborsClassifier(n_neighbors=6)
    knn.fit(faces,labels)
    joblib.dump(knn,'static/face_recognition_model.pkl')


#### Extract info from today's attendance file in attendance folder
def extract_attendance():
    df = pd.read_csv(f'Attendance/Attendance-{datetoday}.csv')
    names = df['Name']
    rolls = df['Roll']
    times = df['Time']
    l = len(df)
    return names,rolls,times,l


```

Figura I.4: Funções Implementadas na Aplicação pt.2

```
130
131  #### Extract info from today's attendance file in attendance folder
132  def extract_attendance():
133      df = pd.read_csv(f'Attendance/Attendance-{datetoday}.csv')
134      names = df['Name']
135      rolls = df['Roll']
136      times = df['Time']
137      l = len(df)
138      return names,rolls,times,l
139
140
141  #### Add Attendance of a specific user
142  def add_attendance(name):
143      username = name.split('_')[0]
144      userid = name.split('_')[1]
145      current_time = datetime.now().strftime("%H:%M:%S")
146
147      df = pd.read_csv(f'Attendance/Attendance-{datetoday}.csv')
148      if int(userid) not in list(df['Roll']):
149          with open(f'Attendance/Attendance-{datetoday}.csv','a') as f:
150              f.write(f'\n{username},{userid},{current_time}')
151
```

Figura I.5: Funções Implementadas na Aplicação pt.3

A primeira é a rota "/", figura I.6, que é rota executada no momento em que o site é carregado, nela é verificado se já existe alguma frequência registrada através da função "extract_attendance()", que checa na pasta "Attendance"se existe algum arquivo ".csv"com registros de frequência do dia atual. Caso exista alguma frequência dentro do arquivo do dia, o conteúdo do arquivo é extraído e armazenado em variáveis, juntamente com a quantidade de registros da planilha.

Após esse processo, é retornada a função "render_template()"que tem como função carregar a interface do web site, e recebe como parâmetro os parâmetros extraídos da função anterior, o arquivo html com a interface do sistema "home.html", a data e a hora em tempo real, e o resultado da função "totalreg()", sendo que a função "totalreg()"checa quantos usuários existem cadastrados dentro da pasta "static/faces"e retorna esse valor.

```
153  #### main page
154  @app.route('/')
155  def home():
156      names,rolls,times,l = extract_attendance()
157      return render_template('home.html',names=names,rolls=rolls,times=times,l=l,totalreg=totalreg(),datetoday2=datetoday2)
158
159
```

Figura I.6: Rota raiz "/"do Servidor Web

A segunda rota é a rota "/add", figura I.7, que é responsável pelo registro de usuários dentro da plataforma. Caso o usuário já exista o código irá sobrescrever os dados cadastrados anteriormente, caso não exista uma nova pasta é criada a partir do comando "os.makedirs()"recebendo os dados do cadastro como parâmetro.

Em seguida a captura de vídeo pela câmera é iniciada a partir da função da biblioteca do OpenCV "cv2.VideoCapture(0)"e então o código entra em um loop para a captura do rosto do usuário até que tenha sido registrado um total de 50 fotos ou se a tecla "esc"for apertada.

Dentro do loop a câmera captura um frame de imagem por cada instante de tempo e utiliza a função "extract_faces()"que converte o espaço de cores do frame para preto, cinza e branco, esse passo é necessário para que o algoritmo de detecção de faces "haar cascade"funcione com maior exatidão, e em seguida aplicase o método "face_detector.detectMultiScale()"que faz a detecção do rosto utilizando o modelo trei-

nado "haarcascade_frontalface_default.xml", recebe como parâmetro o frame em preto, cinza e branco, quanto a escala da imagem deve ser reduzida, o número de vizinhos utilizados para cada classificação e por fim retorna a identificação do rosto do usuário como "face_points".

Agora com apenas o rosto do usuário para ser manipulado, o código utiliza duas funções do OpenCV "cv2.rectangle()"e "cv2.putText()"que respectivamente plotam para o usuário o retângulo e o texto "Images Captured: i/50"para cada frame que um rosto for detectado, em que i é o número de capturas.

Em seguida checa a condição que para cada captura, o frame seja registrado na pasta do usuário através da função "cv2.imwrite()". Depois de registrar todas as fotos, o loop é quebrado e a captura de frames pela câmera é cessado.

Em seguida executa-se a função "train_model()"que extrai todos os usuários e fotos armazenados na pasta "faces"e entra num loop que carrega, redimensiona cada imagem com as respetcivas funções do OpenCV "cv2.imread()cv2.resize()"que recebe como parâmetro a imagem do rosto, e em seguida grava essas imagens no array "faces"e guarda o usuário de cada imagem no array "labels", e treina o modulo com esses dois arrays, utilizando o método "KNeighborsClassifier.fit()"da biblioteca scikit-learn passando os arrays "faces"e "labels"respectivamente e salva esse modelo com o nome "face_recognition_model.pkl". Por fim, após o treinamento do modelo, a página inicial é carregada novamente utilizado o mesmo método utilizado na rota anterior.

```
223  #### Add a new user button function
224  @app.route('/add',methods=['GET','POST'])
225  def add():
226      newusername = request.form['newusername']
227      newuserid = request.form['newuserid']
228      userimagefolder = 'static/faces/'+newusername+'_'+str(newuserid)
229      if not os.path.isdir(userimagefolder):
230          os.makedirs(userimagefolder)
231      cap = cv2.VideoCapture(0)
232      i,j = 0,0
233      while 1:
234          _,frame = cap.read()
235          faces = extract_faces(frame)
236          for (x,y,w,h) in faces:
237              cv2.rectangle(frame,(x, y), (x+w, y+h), (255, 0, 20), 2)
238              cv2.putText(frame,f'Images Captured: {i}/50',(30,30),cv2.FONT_HERSHEY_DUPLEX,1,(255, 0, 20),2,cv2.LINE_AA)
239              if j%10==0:
240                  name = newusername+'_'+str(i)+'.jpg'
241                  cv2.imwrite(userimagefolder+'/'+name,frame[y:y+h,x:x+w])
242                  i+=1
243              j+=1
244          if j==500:
245              break
246          cv2.imshow('Adding new User',frame)
247          if cv2.waitKey(1)==27:
248              break
249      cap.release()
250      cv2.destroyAllWindows()
251      print('Training Model')
252      train_model()
253      names,rolls,times,l = extract_attendance()
254      return render_template('home.html',names=names,rolls=rolls,times=times,l=l,totalreg=totalreg(),datetoday2=datetoday2)
255
256
```

Figura I.7: Rota "/add"do Servidor Web

A terceira rota "/start", figura I.8, utiliza bastante das funções e métodos da segunda rota, sendo utilizada para fazer o reconhecimento facial do usuário a partir do modelo treinado, e então registrar a frequência no arquivo ".csv".

Caso tenha pessoas no banco de dados, a captura de vídeo é iniciada através do método "cv2.VideoCapture(0)"e entra num loop infinito em que repete os mesmos passos para a captura de frames, sendo que a identificação do rosto e amostragem dos retângulos da segunda rota são feitos novamente.

Ao passo que a identificação de rosto seja feita, o código chama a função "identify_face()"que recebe uma imagem como parâmetro, e carrega o modelo treinado "face_recognition_model.pkl"e em sequência retorna o resultado da classificação utilizando o método "predict()"com a imagem como parâmetro.

Após feita a classificação e identificação do usuário, o código adiciona a caixinha de identificação de rosto e o nome do usuário que foi identificado através do método "cv2.putText()"e então entra na função "add_attendance()"q, ue recebe o nome e o identificador do usuário como parâmetro, e usa eles em conjunto com o método "datetime.now()"que determina data e hora atual para fazer o registro da frequência em um arquivo ".csv".

Para isso checa se o identificador da pessoa está dentro do arquivo ".csv"do dia atual com o método "pd.read_csv()"e caso não esteja o nome, o identificador e a hora atual são registrados no arquivo utilizando o método "f.write()".

Por fim, depois que encerra-se o loop, finaliza todo tipo de captura feito pela câmera e recarrega novamente a página inicial com o método "render_template()"porém atualizando a lista de frequência que é mostrada na interface de usuário.

```
160  #### Take Attendance Button funciton
161  @app.route('/start',methods=['GET'])
162  def start():
163      if 'face_recognition_model.pkl' not in os.listdir('static'):
164          return render_template('home.html',totalreg=totalreg(),datetoday2=datetoday2,mess='There is no trained model in the static folder. Please add a new face to contin
165
166      cap = cv2.VideoCapture(0)
167      ret = True
168      while ret:
169          ret,frame = cap.read()
170          if extract_faces(frame)!=():
171              (x,y,w,h) = extract_faces(frame)[0]
172              cv2.rectangle(frame,(x, y), (x+w, y+h), (255, 0, 20), 2)
173              face = cv2.resize(frame[y:y+h,x:x+w], (50, 50))
174              identified_person = identify_face(face.reshape(1,-1))[0]
175              add_attendance(identified_person)
176              cv2.putText(frame,f'{identified_person}',(30,30),cv2.FONT_HERSHEY_DUPLEX,1,(255, 0, 20),2,cv2.LINE_AA)
177          cv2.imshow('Attendance',frame)
178          if cv2.waitKey(1)==27:
179              break
180      cap.release()
181      cv2.destroyAllWindows()
182      names,rolls,times,l = extract_attendance()
183      return render_template('home.html',names=names,rolls=rolls,times=times,l=l,totalreg=totalreg(),datetoday2=datetoday2)
184
```

Figura I.8: Rota "/start"do Servidor Web

A quarta rota "/store", figura I.9, tem como objetivo registrar os arquivos das frequências ".csv"localizados na pasta "Attendance". Quando a rota é chamada entra-se em um laço de repetição selecionando cada um dos arquivos presentes na pasta.

Dentro do laço é o utilizado a função "upload_file()"descrito anteriormente para fazer o upload de cada arquivo para o *bucket* dentro do storage do Google Cloud, que nesse caso é o *bucket* "tcc_bucket_heloisa_pedro", além disso o programa imprime no terminal todos os arquivos localizados no storage utilizando a função "list_*blobs*s()". Após esse processo a página inicial é recarregada utilizando a função "render_template()".

```python
#### This function will run when we click on Send to Cloud Storage Button
@app.route('/store',methods=['GET'])
def store():
    for file_path in files_folder.glob('*.*'):
        print(file_path)

        # use full file name
        gcs.upload_file(bucket_gcs, file_path.name, str(file_path))

        for blob in gcs.list_blobs('tcc_bucket_heloisa_pedro'):
            print(blob.name)
    names, rolls, times, l = extract_attendance()
    return render_template('home.html',names=names,rolls=rolls,times=times,l=l,totalreg=totalreg(),datetoday2=datetoday2)

```

Figura I.9: Rota "/store"do Servidor Web

A penúltima rota "/download", figura I.10, tem a função de fazer o download de todos os arquivos presentes dentro do *bucket* no Google Cloud Storage. Uma vez que essa rota é chamada é feita a listagem de todos os *blobs* do *bucket* através do método "list_blobs()"passando como parâmetro o nome do *bucket* "tcc_bucket_heloisa_pedro".

Em seguida é feito um laço de repetição selecionando cada arquivo contido no storage, e então checa-se se a pasta "Downloaded"existe e caso não exista ela é criada utilizando a função "mkdir()". Dessa forma, para cada arquivo é utilizado o método "download_to_filename()"da biblioteca "google-cloud-storage"que recebe como parâmetro o local que o *blob* deve ser baixado, referenciado como "path_download".

Por fim, depois de todos os arquivos baixados a página inicial é recarregada através do método "render_template()".

```python
#### This function will run when we click on Download Cloud Storage Button
@app.route('/download',methods=['GET'])
def download():
    gcs_tcc_blobs = gcs.list_blobs('tcc_bucket_heloisa_pedro')
    for blob in gcs_tcc_blobs:
        path_download = downloads_folder.joinpath(blob.name)
        if not path_download.parent.exists():
            path_download.parent.mkdir(parents=True)
        blob.download_to_filename(str(path_download))
        print(blob)
    names, rolls, times, l = extract_attendance()
    return render_template('home.html',names=names,rolls=rolls,times=times,l=l,totalreg=totalreg(),datetoday2=datetoday2)

```

Figura I.10: Rota "/download"do Servidor Web

Finalizado as rotas do servidor *web*, a última rota "/delete", figura I.11, foi feita para apagar todos os arquivos dentro do *bucket* do *storage*. Feita de forma semelhante a rota "/download", nessa rota é feita a listagem de todos os *blobs* presentes no *bucket*, e entra em um laço de repetição em que seleciona-se cada *blob* e utiliza-se o método "delete()"da biblioteca do "google-cloud-storage"que deleta um blob do Cloud Storage. Uma vez que todos os *blobs* foram deletados, novamente a página inicial recarrega utilizando o método "render_template()".

```python
#### This function will run when we click on Delete Cloud Storage Button
@app.route('/delete',methods=['GET'])
def delete():
    gcs_tcc_blobs = gcs.list_blobs('tcc_bucket_heloisa_pedro')
    for blob in gcs_tcc_blobs:
        blob.delete()
        print(blob)
    names, rolls, times, l = extract_attendance()
    return render_template('home.html',names=names,rolls=rolls,times=times,l=l,totalreg=totalreg(),datetoday2=datetoday2)


```

Figura I.11: Rota "/delete"do Servidor Web