Instituto Federal de Educação, Ciência e Tecnologia de São Paulo – Câmpus Cubatão

Reconhecimento Facial e suas aplicações

Alunos:

- Allanis dos Santos Cordeiro – CTII 448
- Beatriz Miranda Bezerra – CTII 417
- Bruno de Oliveira Santos – CTII 448
- Evelyn Santos de Santana – CTII 448
- Gustavo Murilo Cavalcante Carvalho – CTII 448
- Leonardo Yukio Aoki – CTII 448
- Lívia Martins Alves – CTII 448
- Luan Natan de Sousa – CTII 448
- Luiz Felipe Ciantela Machado – CTII 448
- Pedro Henrique Pupo Marques – CTII 417

Disciplina: PJS - Projeto de Sistemas

Discente: Maurício Neves Asenjo

Sumário

1. INTRODUÇÃO

A tecnologia de reconhecimento facial tornou-se parte de nossas vidas diárias, nos proporcionando praticidade e rapidez nas coisas básicas do dia a dia, como desbloquear seu celular e autorizar transações. O reconhecimento facial utiliza uma tecnologia avançada para mapear suas características faciais e criar um modelo quase 3D do seu rosto. Esses dados são usados para garantir que você é quem diz ser, combinando seu rosto com uma imagem capturada anteriormente. Isso só é possível porque, embora existam múltiplas variações entre os rostos, há uma composição básica que não muda. Os aplicativos veem isso como algo em comum que pode variar com base em fatores como a complexidade do sistema. Na verdade, não importa quem seja o desenvolvedor, os sistemas de reconhecimento facial têm o mesmo objetivo: detectar rostos com formas geométricas e algorítmicas. Assim "montando" como um quebra-cabeça.

Muitos dispositivos coletam informações biométricas, incluindo dados faciais, para facilitar logins e controle de acesso. Por exemplo, algumas escolas usam software de reconhecimento facial durante o horário escolar para ajudar a acompanhar os alunos. Este sistema pode ser configurado para que apenas funcionários autorizados da escola possam entrar no prédio, evitando que indivíduos não autorizados acessem áreas sensíveis como salas de aula ou computadores.

O projeto tem como iniciativa implementar essa tecnologia, assim facilitando o cotidiano de alunos e funcionários das escolas, alguns exemplos da efetividade desse sistema são: utilizar como registro de presença, via reconhecimento facial dando aos pais uma maior sensação de segurança sabendo que seu filho está realmente na escola e que a equipe da escola está a cargo desses alunos durante o horário escolar. O sistema ajuda a economizar tempo nas chamadas em sala de aula e permite um melhor gerenciamento da merenda escolar. O sistema informará aos funcionários quantos alunos estão na escola naquele dia e quantas refeições precisam ser fornecidas, para evitar o desperdício.

Além do controle de assiduidade e fluxo, o sistema permite a gestão do comportamento institucional e administrativo de cada escola, trazendo mais agilidade e agilidade ao processo escolar como um todo. Trata-se de um investimento significativo, mas que traz muitas vantagens e controle inteligente de atendimento, permitindo definir estratégias de melhoria com base em dados confiáveis e atualizados diariamente. O sistema gerencia todo o trabalho da secretaria escolar, do qual terá o controle de todos os alunos, desde os presentes no dia, matrícula escolar de cada aluno, infraestrutura, merendas, transporte, bibliotecas, tornando enfim um diagnóstico preciso da situação de todas as instituições.

Com isso, criamos esse projeto desenvolvendo um software de reconhecimento facial que com dados fornecidos na matrícula, esse trabalho explica os passos do desenvolvimento do projeto, desde como funciona o reconhecimento facial até a codificação da interface. O grupo tem o objetivo de facilitar não só a vida dos usuários do campus Cubatão, mas sim de todos os institutos. O projeto visa trazer tecnologia, praticidade, rapidez e organização da vida escolar, modernizando a gestão dos institutos.

2. IAs

2.1. O QUE É INTELIGÊNCIA ARTIFICIAL

A inteligência artificial (IA) refere-se amplamente a qualquer comportamento humano exibido por uma máquina ou sistema. Na forma mais básica da IA, os computadores são programados para "imitar" o comportamento humano usando dados extensivos de exemplos passados de comportamento semelhante. Isso pode variar desde reconhecer diferenças entre um gato e um pássaro até a realização de atividades complexas em uma fábrica.

Se você está falando sobre Deep Learning, pensamento estratégico ou outra espécie de IA, a base de seu uso é em situações que requerem respostas rápidas. Com a IA, as máquinas podem trabalhar de forma eficiente e analisar grandes quantidades de dados em um piscar de olhos, resolvendo problemas através de aprendizado supervisionado, não supervisionado ou reforçado.

- Os chatbots usam a IA para entender os problemas dos clientes mais rapidamente e fornecer respostas mais eficientes

- Os assistentes inteligentes usam a IA para analisar informações críticas de grandes conjuntos de dados de texto livre para melhorar a programação

- Os mecanismos de recomendação podem fornecer recomendações automatizadas para programas de TV com base nos hábitos de visualização dos usuários.

2.2. TIPOS DE INTELIGÊNCIA ARTIFICIAL

A inteligência artificial é classificada em duas categorias principais: IA baseada em funcionalidade e IA baseada em recursos.

- **Com base na funcionalidade**

Máquina Reativa – Esta IA não tem poder de memória e não tem a capacidade de aprender com ações passadas. O Deep Blue da IBM está nessa categoria.

Teoria Limitada – Com a adição de memória, esta IA usa informações passadas para tomar melhores decisões. Aplicativos comuns como aplicativos de localização GPS se enquadram nessa categoria.

Teoria da Mente – Esta IA ainda está sendo desenvolvida, com o objetivo de ter uma compreensão muito profunda das mentes humanas.

IA autoconsciente – Essa IA, que poderia compreender e evocar emoções humanas, bem como ter as suas próprias, ainda é apenas hipotética.

- **Com base em recursos**

Inteligência Estreita Artificial (ANI) – Um sistema que executa tarefas programadas estritamente definidas. Esta IA tem uma combinação de memória reativa e limitada. A maioria das aplicações de IA de hoje estão nessa categoria.

Inteligência Geral Artificial (AGI) – Esta IA é capaz de treinar, aprender, entender e atuar como um ser humano.

Super Inteligência Artificial (ASI) – Esta IA executa tarefas melhores que os humanos devido às suas habilidades superiores de processamento de dados, memória e tomada de decisão. Não existem exemplos do mundo real hoje.

2.3. A RELAÇÃO ENTRE IA, MACHINE LEARNING E DEEP LEARNING

A inteligência artificial é um ramo da ciência da computação que busca simular inteligência humana em uma máquina. Os sistemas de IA são alimentados por algoritmos, usando técnicas como aprendizado de máquina e aprendizado profundo para demonstrar comportamento "inteligente".

Existem algumas tecnologias que despontam dentro do campo da Inteligência Artificial ou que contribuem para que ela evolua. Veja algumas das principais adiante.

- **Machine Learning**

Um computador "aprende" quando seu software é capaz de prever e reagir com sucesso a diferentes cenários com base em resultados anteriores. O aprendizado de máquina refere-se ao processo pelo qual os computadores desenvolvem o reconhecimento de padrões, ou a capacidade de aprender continuamente e fazer previsões com base em dados, e pode fazer ajustes sem serem especificamente programados para fazê-lo. Uma forma de inteligência artificial, o aprendizado de máquina automatiza efetivamente o processo de construção de modelos analíticos e permite que as máquinas se adaptem a novos cenários de forma independente.

- **Deep Learning**

O Deep Learning, ou aprendizagem profunda, é um tipo especial de aprendizado de máquina. Envolve redes neurais artificiais com várias camadas de abstração, sendo aplicado para reconhecimento de padrões e aplicativos de classificação amparados por conjuntos de dados. O processo de aprendizado ocorre entre suas camadas de neurônios matemáticos, em que a informação é transmitida através de cada camada. Nesse esquema, a saída da camada anterior é a entrada da posterior.

O Deep Learning “treina” as máquinas para executarem atividades como se fossem humanos. Por exemplo, identificação de imagens e reconhecimento de fala. Também processa dados.

- **Processamento De Linguagem Natural (PLN)**

Processamento de Linguagem Natural visa ao estudo e à tentativa de se reproduzir processos de desenvolvimento ligados ao funcionamento da linguagem humana. Para isso, emprega softwares, programação e outras soluções. Por meio do PLN, as máquinas podem compreender melhor os textos — o que envolve reconhecimento de contexto, extração de informações, desenvolvimento de resumos etc. Também é possível compor textos partindo de dados obtidos por computadores. O PLN pode ser usado em áreas como atendimento ao consumidor e na produção de relatórios corporativos.

3. COMO FUNCIONA A TECNOLOGIA DE RECONHECIMENTO FACIAL

A maneira como a tecnologia de reconhecimento facial está diretamente ligada as IA (Inteligência Artificial) e funciona por meio de um sistema que utiliza algoritmos de machine learning ou ML (que é um processo de programação que os dispositivos desenvolvem para o reconhecimento de padrões, sem que sejam programados para tal) e softwares que mapeiam o rosto de determinada pessoa. A tecnologia de reconhecimento facial escaneia os certos padrões de rostos de maneira geométrica e logarítmica e com isso determinam as características únicas do rosto e as compara com os que já estão em seu banco de dados. Essas características únicas que são usadas como pontos do rosto humano são, por exemplo:

- A distância apresentada entre os olhos.
- A distância entre os olhos e o nariz.
- A distância entre o nariz e o queixo.
- O formato que o rosto apresenta.
- A estrutura do nariz.
- A estrutura do queixo.
- Marcas e cicatrizes no rosto.

Como dito anteriormente a tecnologia escaneia o rosto na procura dos pontos que serão utilizados, logo depois disso a tecnologia busca validar a imagem capturada para que se possa utilizar na procura pelo banco de dados, quando o dado, é achado no banco ocorre, o reconhecimento da biometria facial e por último ocorre a ação que reconhece o rosto, independentemente da aplicação. Logo após esse ato de reconhecimento alguns programas já armazenam em seus bancos de dados as novas imagens para que exista o aprimoramento da detecção e para que não apresente falhas ao decorrer do tempo, já que os rostos envelhecem.

Há também três tipos principais de variações da tecnologia de reconhecimento facial, eles são:

1. Classificação 1:1: Essa variação é conhecida como variação e a mais popular, onde só utiliza uma única imagem para a comparação do reconhecimento da face.

Um exemplo é o reconhecimento facial presente nos smartphones, que faz uso da verificação 1:1, nesse processo o seu rosto é gravado, e das próximas vez que você entrar, é verificado se o rosto é igual ao que foi salvo anteriormente, dessa forma, é proporcionado ao usuário mais praticidade e segurança.

2. Classificação 1:N: Essa variação é conhecida como identificação, ela utiliza uma imagem a fim de comparação com várias faces e espera determinar um maior grau similaridade com alguma das faces.

Um exemplo da "identificação" 1:N, é o uso da polícia nas perícias de reconhecimento facial, onde rosto dos criminosos são gravados na base de dados, e toda vez que chega um novo criminoso, eles fazem o reconhecimento facial nele para ver seus antecedentes criminais e seus dados, ao invés de ficar horas em um arquivo procurando.

3. Classificação N:N: Essa variação procura através de uma única imagem identificar várias faces de uma única vez e utiliza um algoritmo parecido com a variação 1:1.

Esse tipo de verificação (N:N) é usada por exemplo em câmeras de vigilância pela cidade, capazes de reconhecer simultaneamente todas as pessoas na imagem, sendo assim um uso avançado da tecnologia se comparado aos demais.

4. APIs

4.1 O QUE É UMA API

Application Programing Interface(Interface de Programação de Aplicação), mais conhecida pela sigla API, se trata de uma série de rotinas e padrões de programação que definem a forma de executar uma determinada tarefa. Melhor dizendo, a API é um conjunto de regras para realizar uma tarefa. Ela é o contrato.

A interface pode ser vista como um contrato entre os dois sistemas, onde são definidos campos de entrada e saída que são respeitados pela parte que consome e pela que fornece a informação. Caso haja alguma alteração interna em qualquer um dos lados, o contrato continua a ser respeitado, não obrigando o outro lado a fazer nenhuma alteração no software.

Mesmo que o contrato seja mutável e possa receber novas funcionalidades ao longo do tempo, é vantajoso que no desenvolvimento das APIs, antes mesmo de começar a escrever as linhas de código, seja idealizado primeiro o contrato (contract first), trazendo assim maior maturidade a solução como um todo, pela capacidade de agilizar a geração do código fonte, mocks, documentação etc..

Utilizando APIs é possível realizar a comunicação entre dois sistemas. Elas podem ser usadas para adicionar/integrar ao programa funcionalidades de outras aplicações, um exemplo são os diversos sites que facilitam o cadastro utilizando APIs feitas por terceiros (Google, Facebook, Twitter etc.) que fornecem ao site em questão, os dados presentes na outra plataforma para que sejam utilizados no cadastro.

- **Api, Web Services, Library E Framework**

Quando falamos de API muita gente confunde o termo com outras coisas relacionadas como web services, framework e library. Por serem termos correlatos, às vezes, essa confusão não acarreta em erro, mas isso não significa que sejam coisa iguais. Por isso, é válido fazer algumas distinções.

API são confundidas com web services, pela questão do provimento de serviços, que para acontecer necessita de uma API, mas não é a API em si.

Na computação, Library (biblioteca), também chamada de lib, nada mais é que uma coleção de conjuntos não voláteis que podem ser usados por programas ou para o desenvolvimento de softwares. Elas podem conter dados de configuração, documentação, subrotinas, classes, especificações de valores, tipos de dados etc. Enquanto a API é a especificação, a biblioteca é a implementação das regras.

Um framework normalmente é um conjunto de bibliotecas para conseguir executar uma operação maior. É comum um framework encapsular os comportamentos da API em implementações mais complexas, permitindo o seu uso de forma mais flexível, frequentemente através de extensões, configurações e inversões de controle.

4.2. FACE-API.JS

Face-api.js é uma API JavaScript para detecção e reconhecimento facial no browser implementado sobre a API principal do tensorflow.js. O Face-api.js implementa uma série de redes neurais convolucionais (CNNs), otimizada para a web e dispositivos móveis.

Para detecção facial, o face-api.js implementa os modelos SSD Mobilenet V1, Tiny Face Detector, e o experimental MTCNN.

O SSD (Single Shot Multibox Detector) MobileNet V1 é um modelo baseado no MobileNet V1 que visa obter alta precisão na detecção facial. Esse modelo basicamente calcula a localização de cada face em uma imagem e retorna as caixas delimitadoras junto com sua probabilidade para cada face detectada.

O Tiny Face Detector é um modelo para a detecção facial em tempo real, que quando comparado com o SSD Mobilenet V1, é menor, mais rápido, e consome menos recursos. Esse modelo foi treinado em um conjunto de dados de 14 mil imagens rotuladas com caixas delimitadoras. De acordo com Mühler, clientes com recursos limitados não devem ter problemas ao usar este modelo.

O MTCNN (redes neurais convolucionais em cascata multitarefa) é um modelo experimental que representa uma alternativa de detecção facial para SSD MobileNet V1 e Tiny Yolo V2, que oferece muito mais possibilidades de configurações.

Para detecção de pontos de referência de 68 pontos, existem dois modelos leves e rápidos, o face_landmark_68_model que requer somente 350kb, e o face_landmark_68_tiny_model que requer 80kb. Os modelos foram treinados em um conjunto de dados de aproximadamente 35 mil imagens faciais rotuladas com 68 pontos de referência de face.

Para o reconhecimento facial, um modelo baseado em uma arquitetura do tipo ResNet-34 é fornecido no face-api.js para calcular um descritor de face a partir de qualquer imagem de face. Esse modelo não está limitado ao conjunto de faces usadas para o treinamento, o que significa que os desenvolvedores podem usá-la para o reconhecimento facial de qualquer pessoa. É possível determinar a similaridade de duas faces arbitrárias comparando seus descritores de face.

- **Carregando os modelos**

Todas as instâncias de rede neural global são exportadas via faceapi.nets:

```
console.log(faceapi.nets)
// ageGenderNet detecta o gênero e idade
// faceExpressionNet detecta as expressões
// faceLandmark68Net desenha os traços no rosto (boca, nariz, olhos)
// faceRecognitionNet reconhece de quem é o rosto
// tinyFaceDetector detecta os rostos no vídeo
```

Para carregar um modelo é necessário fornecer os arquivos de modelo, e após isso, assumindo que os modelos estão em public/models:

```
await faceapi.loadSsdMobilenetv1Model('/models') //
accordingly for the other models:
// await faceapi.nets.faceLandmark68Net.loadFromUri('/models')
// await faceapi.nets.faceRecognitionNet.loadFromUri('/models')
// …
```

4.3. IBM WATSON - VISUAL RECOGNITION

O IBM Watson é uma das mais fortes tendências quando o assunto são ferramentas que oferecem suporte para a aplicação de programação cognitiva em grandes quantidades de dados. Ele é, na verdade, um conjunto de APIs: cada API que constitui a ferramenta é responsável por uma especialidade do serviço e pode ser utilizada para criar seus próprios sistemas cognitivos. Os serviços podem ser

subdivididos em 6 grupos de acordo com a área de abrangência: Conversação (Conversation), Conhecimento (WKS, NLU e Discovery), Visão (Visual Recognition), Voz (Speech to Text), Linguagem (Language Translator e NLC), Empatia (Personality Insights e Tone Analyzer).

O Visual Recognition é um serviço que utiliza técnicas de deep learning para analisar o conteúdo de imagens. O serviço possui um conjunto de "classes padrão" que torna-o capaz de analisar o conteúdo de imagens, identificando palavras-chave que as descrevam.

Além de identificar o conteúdo de imagens utilizando as "classes padrão", é possível treinar o Visual Recognition com amostras de imagens que você coletou, ajustadas para um domínio específico de aplicação, criando classificadores personalizados.

4.4. OPENCV

O OpenCV (Open Source Computer Vision Library), se trata de uma biblioteca sobre visão computacional open-source (um software com código aberto), multiplataforma. Foi desenvolvida próximo ao início do século 21 pela empresa Intel. O OpenCV, reúne todos os recursos necessários, para as mais diversas aplicações na área de visão computacional que possam ser imaginadas, ele nos permite: obter imagens de câmeras digitais, fazer o processamento de imagens estáticas ou de vídeos, além de possuir algoritmos de Inteligência Artificial.

Por ser multiplataforma possui compatibilidade com o Windows, Linux, MacOS, IOS e Android, além de ter a possibilidade para utilizar as interfaces C++, C, Java, Python e uma infinidade de outras linguagens. Existem um ponto a ser observado quando se tem a vontade de utilizar o OpenCV, deve se ter noção dos recursos de hardware disponíveis, por lidar com um algoritmo de processamento de imagens, existe um alto consumo de memória e uma carga alta de estresse para o processador, então é aconselhável que possua um cooler de boa qualidade e tenha uma quantidade razoável de memória RAM.

De forma básica esse é o funcionamento do OpenCV:

1 - Captura

2 - Detecção

3 - Pré-processamento / filtro

4 - Inspeção / análise

5 - Resultado

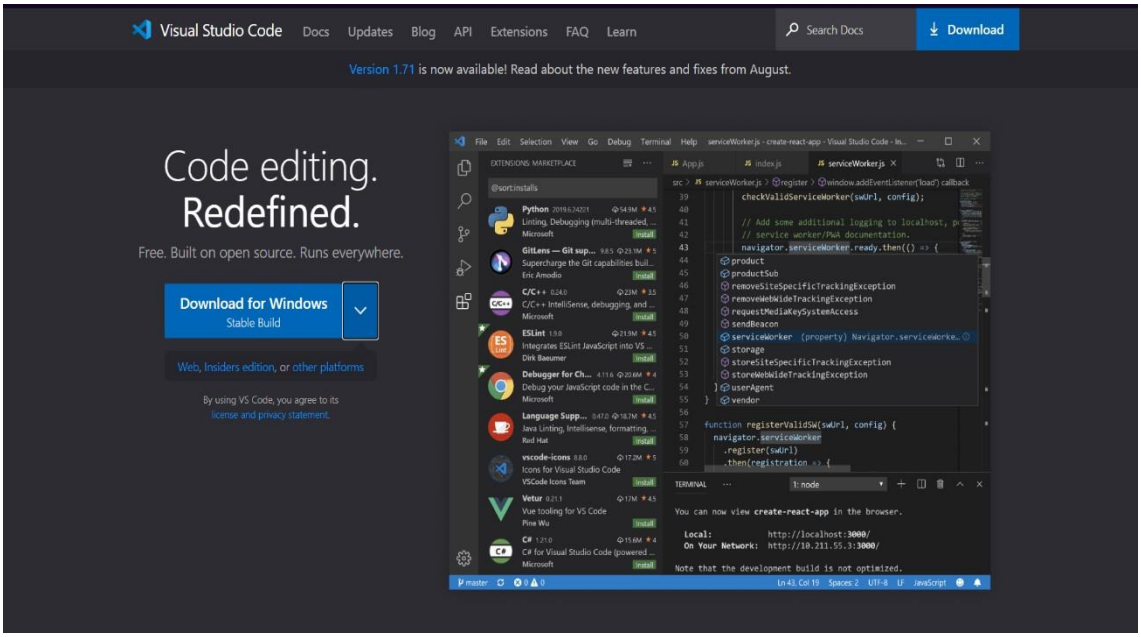
Para o funcionamento precisaremos de um hardware para o processamento e armazenamento das imagens, uma câmera (quanto melhor a qualidade, maior será a utilização do hardware), visibilidade do objeto ou pessoa que será analisado (a) e a programação do OpenCV.

5. DOWNLOAD E INSTALAÇÃO DAS APLICAÇÕES NECESSÁRIAS

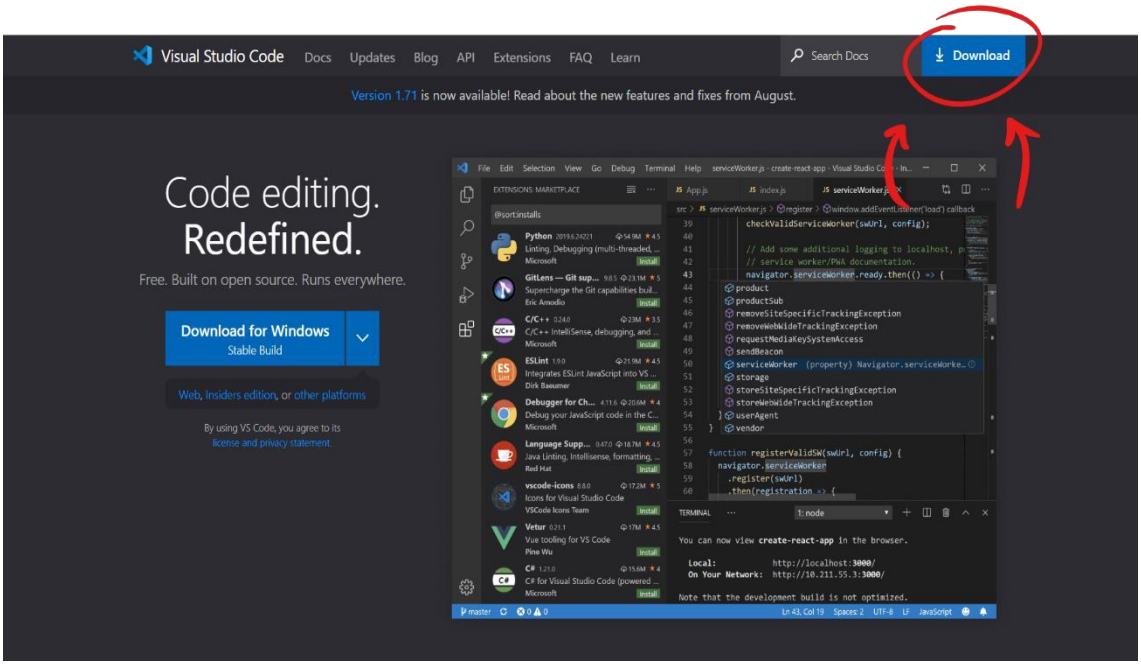
Durante todo o desenvolvimento do projeto, a única aplicação utilizada foi o Visual Studio Code e algumas de suas extensões que podem ser baixadas no próprio aplicativo.

5.1. VISUAL STUDIO CODE

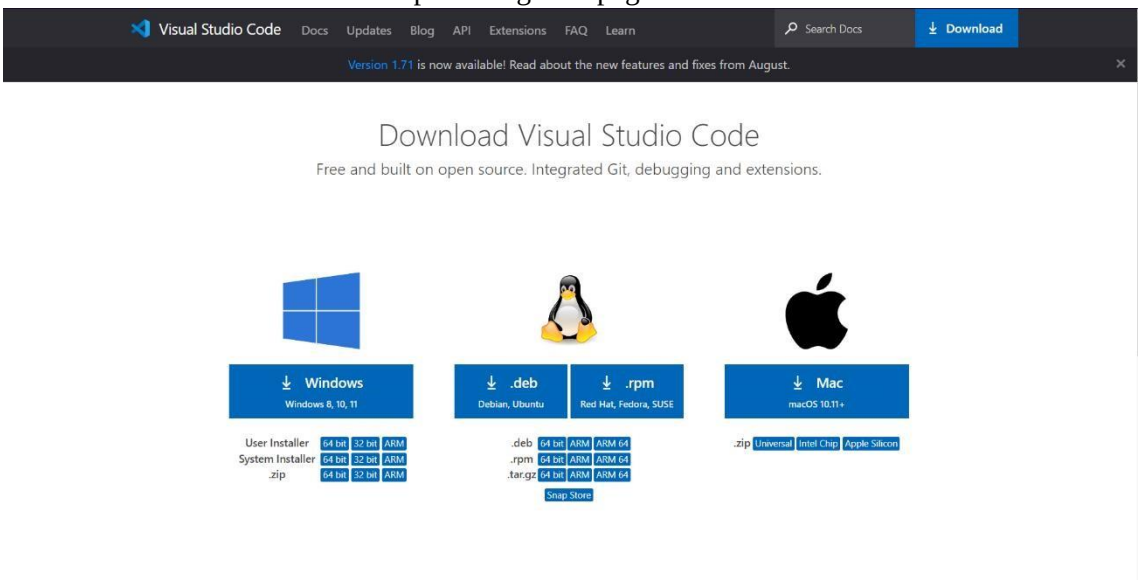
A primeira coisa que deve ser feita para baixar esse aplicativo é acessar o seguinte site: https://code.visualstudio.com.



Para ter acesso a todas as versões do aplicativo você deve clicar no item abaixo.

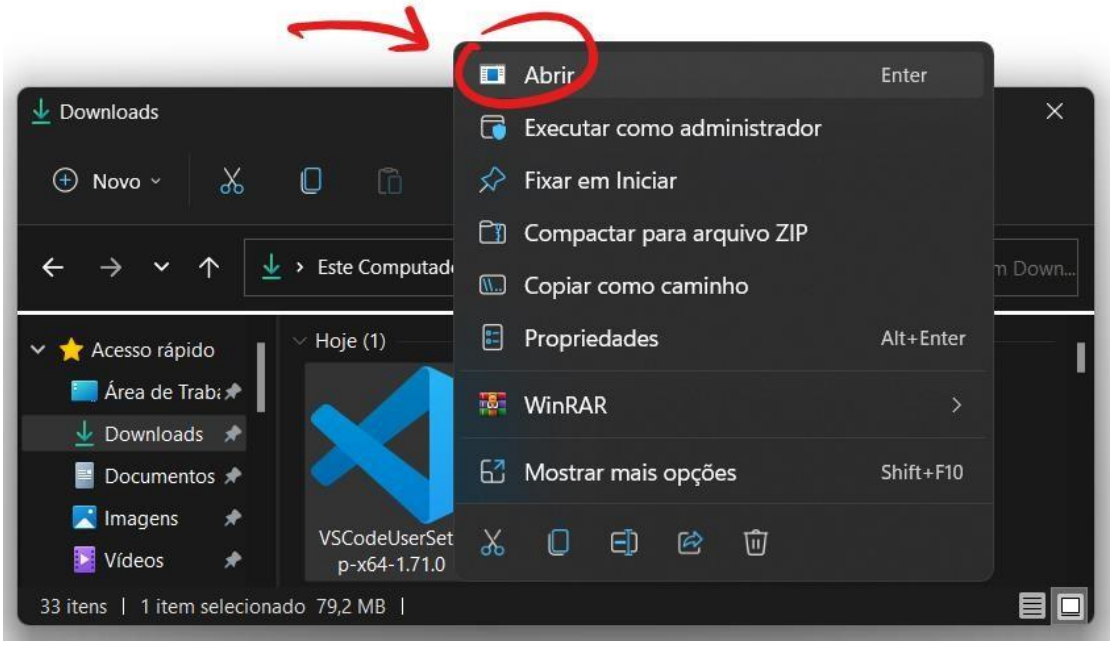


Então você será redirecionado para a seguinte página:

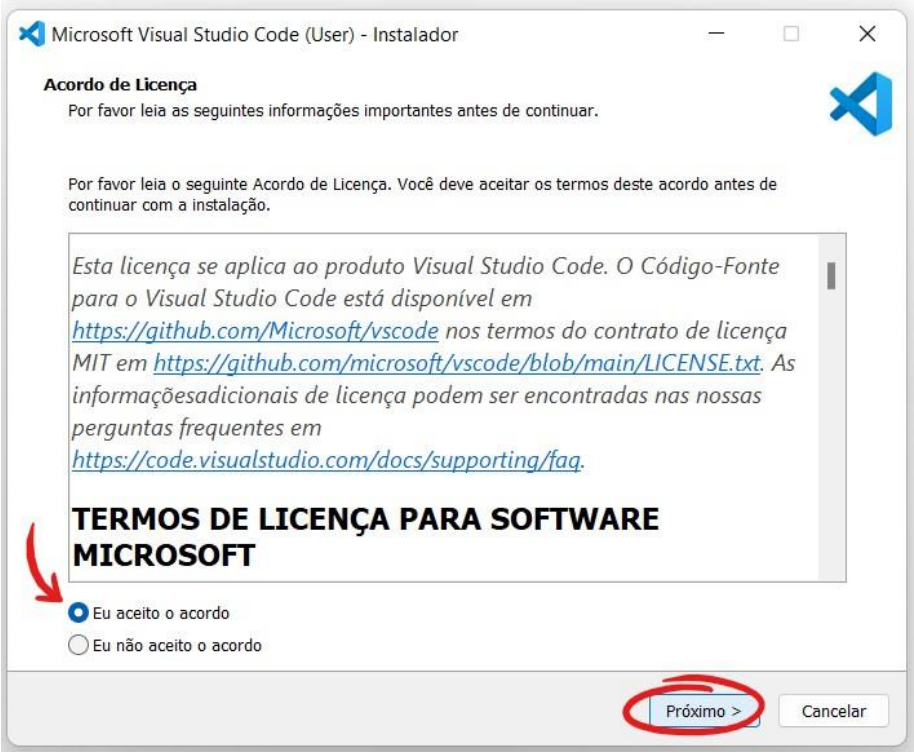


Agora basta selecionar a versão correspondente ao seu sistema operacional e o download do instalador será iniciado.

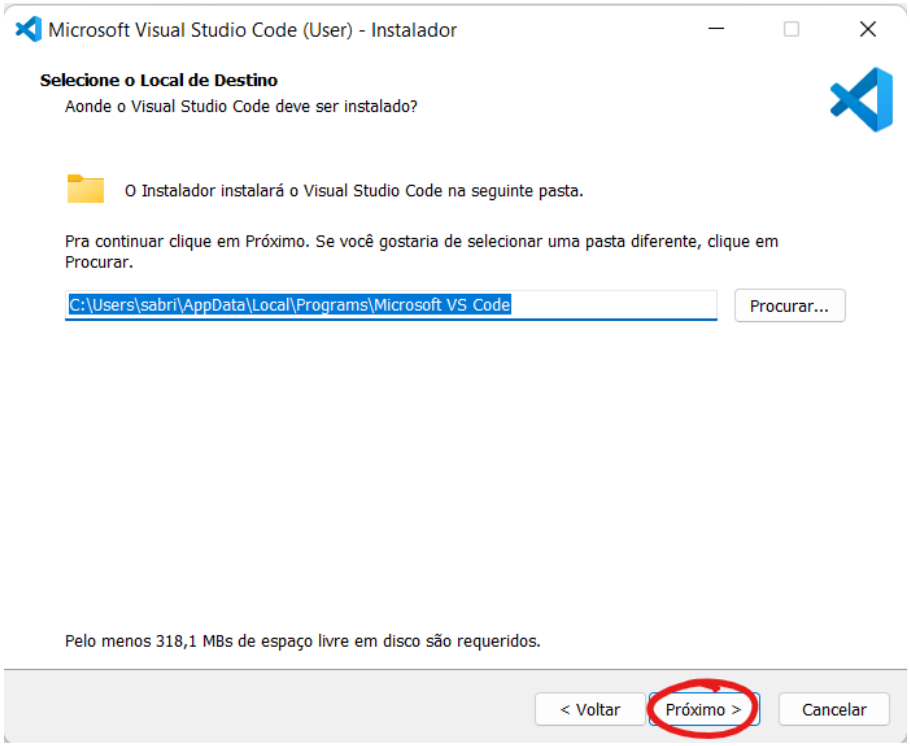
Feito o download, é necessário executar o arquivo. Para isso basta clicar com o botão direito do mouse no arquivo do instalador e então clicar em “Abrir”



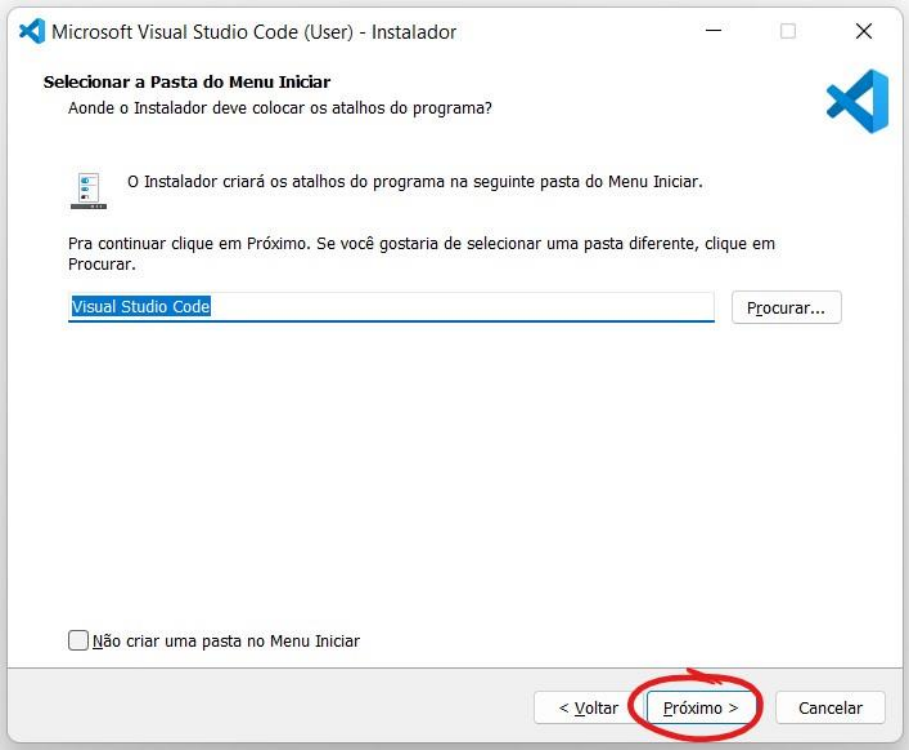
Com o instalador aberto, a primeira coisa a ser feita é ler e então aceitar os termos de licença e uso do aplicativo. Feito isso, basta clicar em próximo.



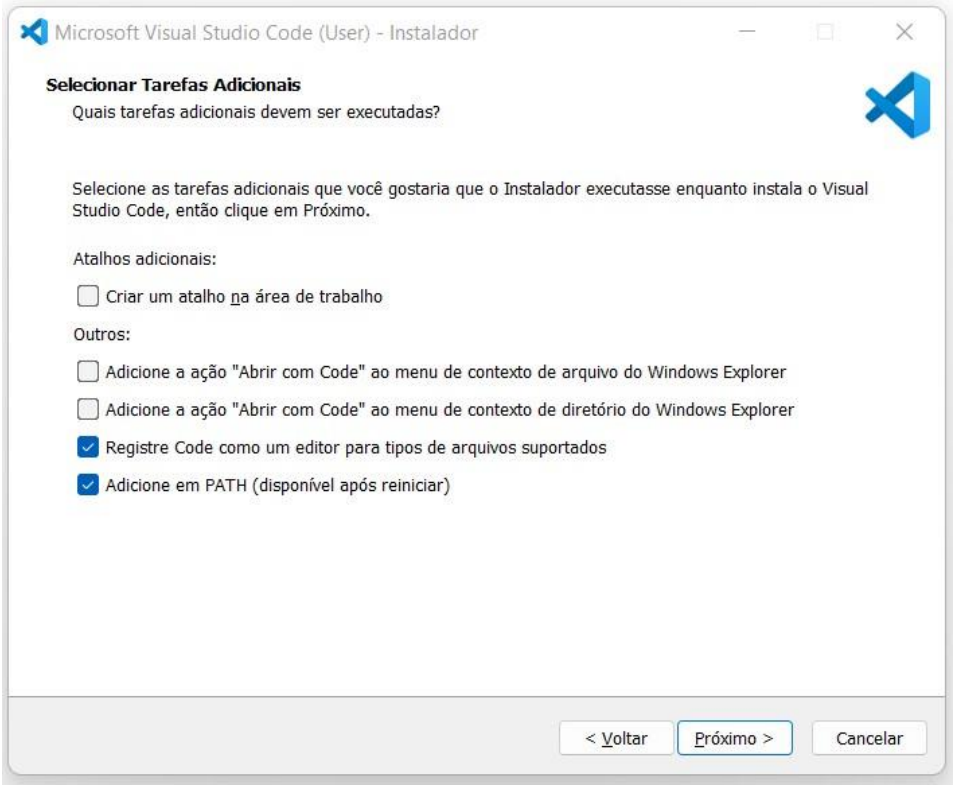
A próxima tela permite que você escolha manualmente o local de instalação. No entanto, podemos utilizar o que foi definido automaticamente e apenas seguir para a próxima página.



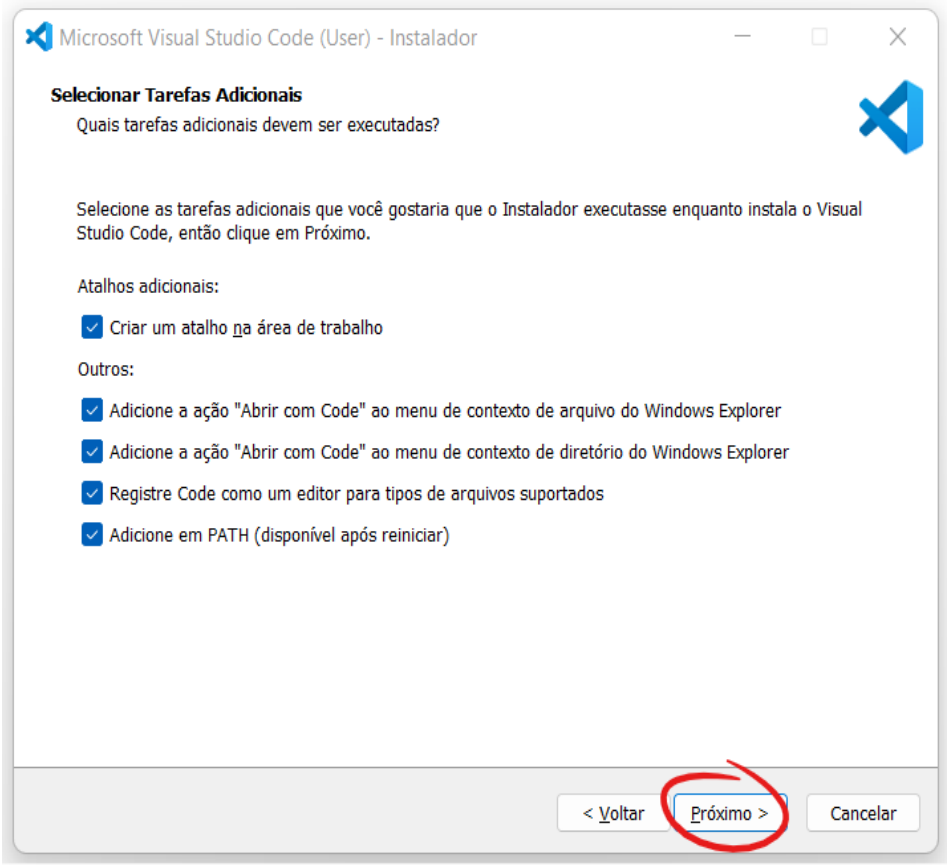
Nessa tela você seleciona se quer ou não adicionar a aplicação ao menu iniciar do windows. Caso não tenha problema com isso, basta ir ao próximo.



A próxima tela aparece da seguinte maneira:

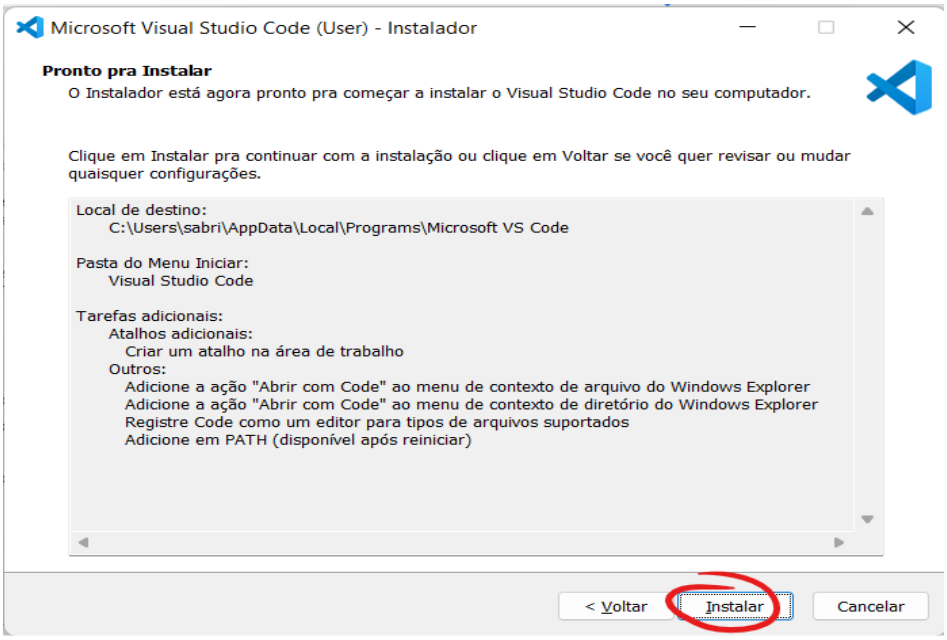


Uma opção, a qual eu prefiro, é marcar todas essas caixas. Após isso, clique em próximo.

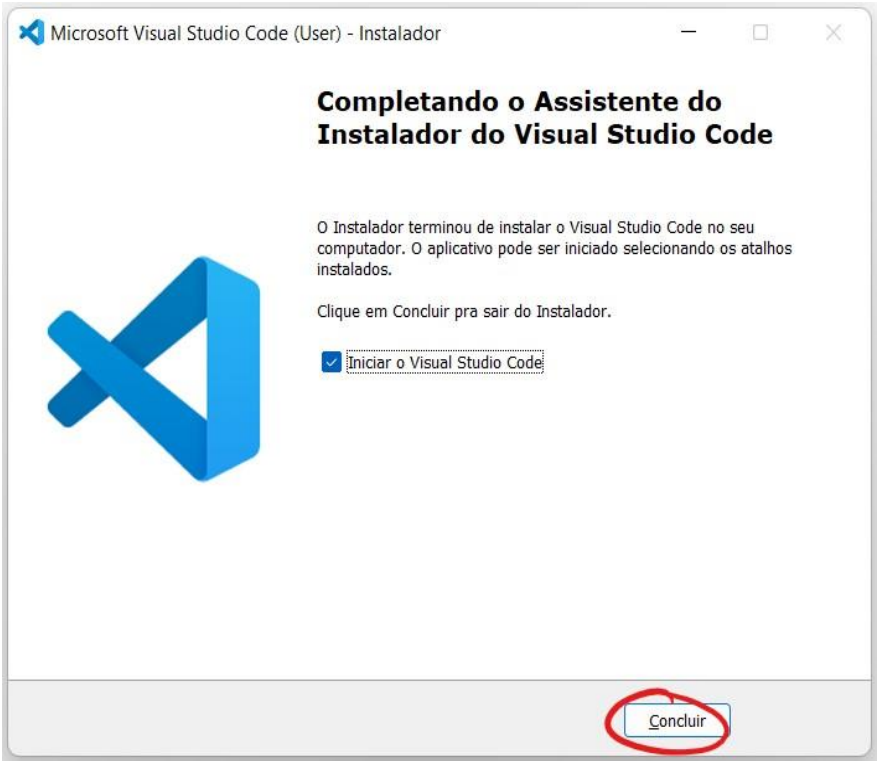


A próxima tela consiste na confirmação da instalação. Basta clicar em instalar e esperar que a instalação seja finalizada.

Observação: o tempo de instalação pode demorar um pouco a depender das especificações da sua máquina.

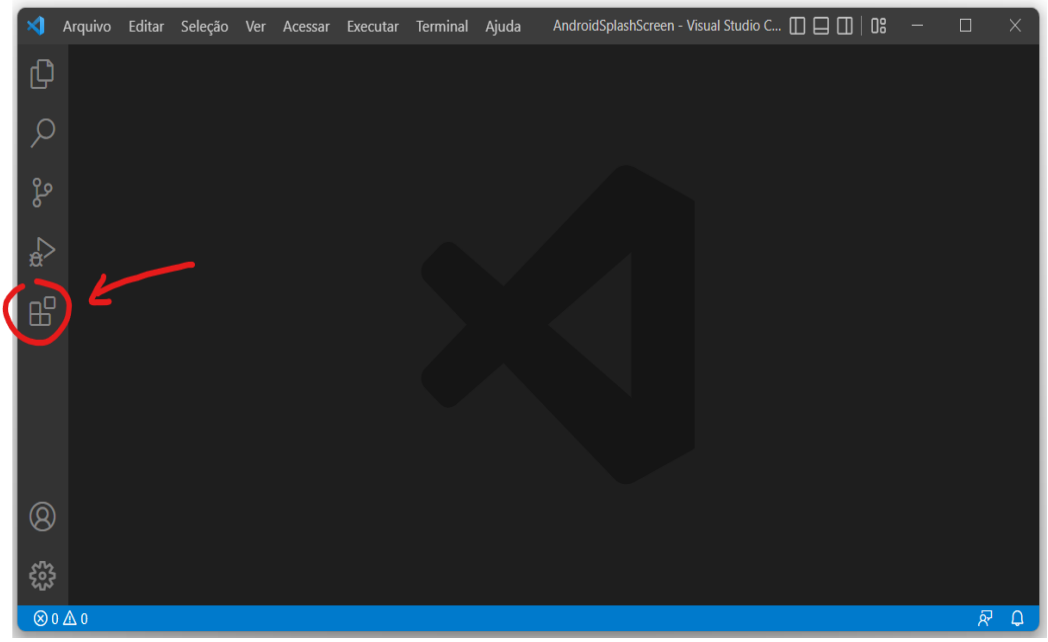


Pronto! A instalação chegou ao fim. Clique em concluir e o aplicativo será aberto.

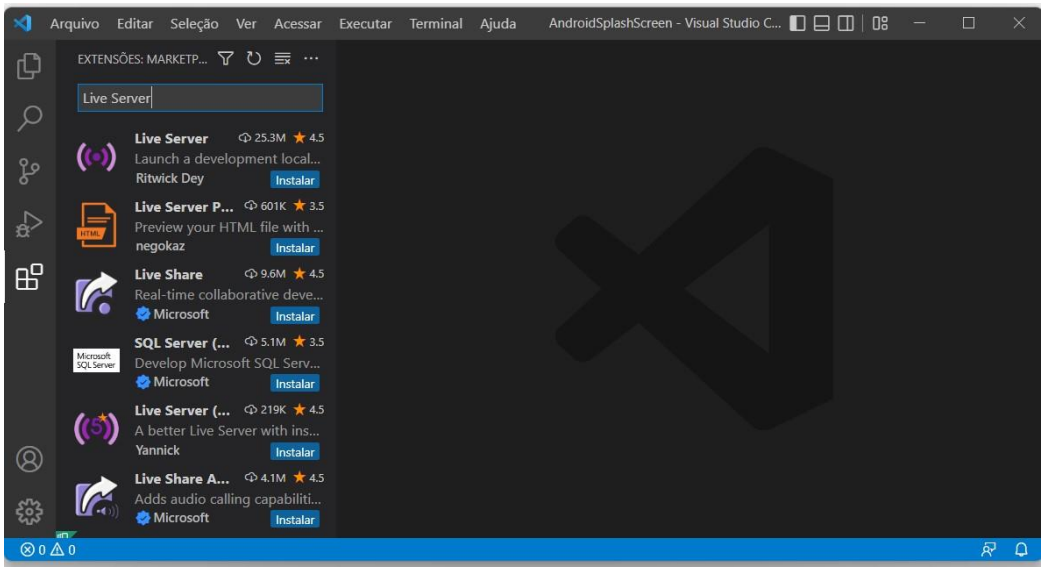


5.2. EXTENSÃO DO VS CODE - LIVE SERVER

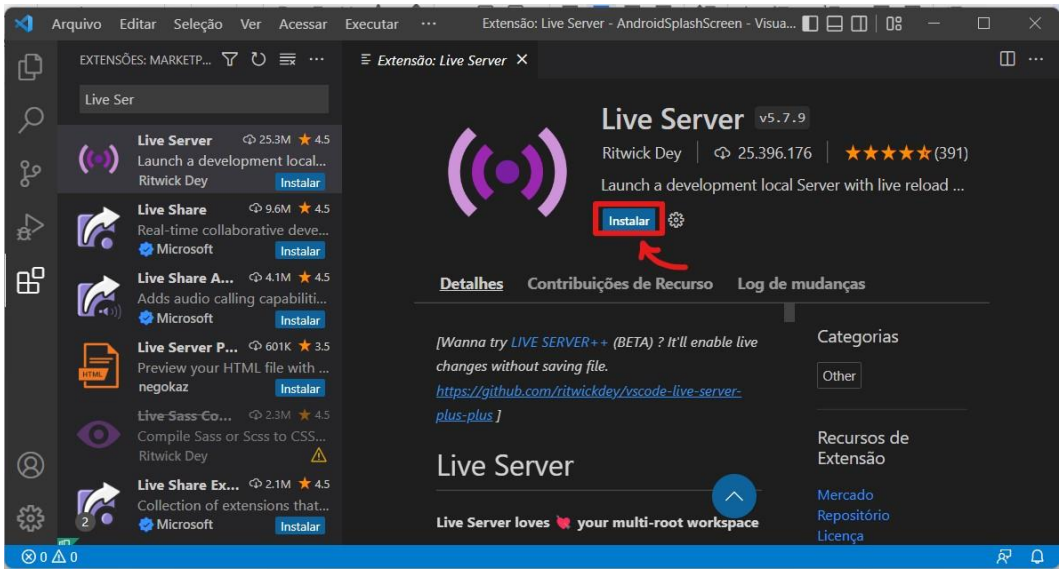
Com o aplicativo aberto clique no item “Extensões” no menu lateral. Também pode ser acessado ao pressionar as teclas: Ctrl+Shif+x



Após isso, vá na barra de pesquisas e procure por “Live Server”, assim como na imagem.



Selecione o primeiro item e clique no botão “Instalar”. Então é só aguardar que ela será instalada automaticamente.



6. CONSTRUÇÃO DO CÓDIGO DA APLICAÇÃO

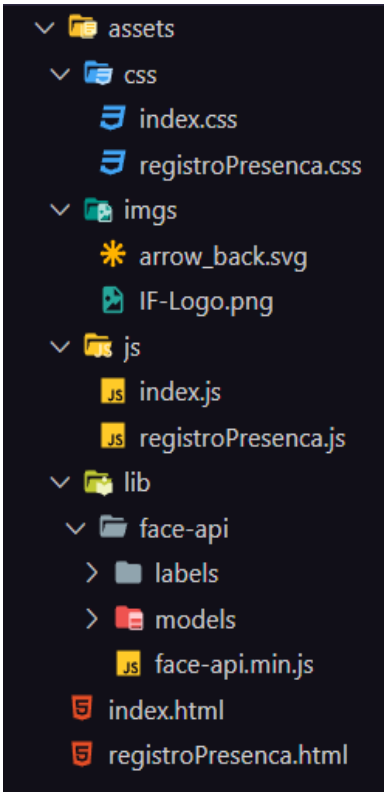
6.1. ESTRUTURA E ORGANIZAÇÃO DOS ARQUIVOS DA APLICAÇÃO

As funcionalidades desta aplicação são:

- Reconhecimento de rosto
- Reconhecimento de olhos, nariz e boca
- Reconhecimento de expressão fácil
- Reconhecimento de gênero
- Reconhecimento de idade
- Todas essas com o objetivo de registrar a presença dos alunos através do reconhecimento facial

As pastas:

- Dentro do 'lib' se encontram o arquivo principal para o uso da face-api (face-api.min.js).
- A pasta 'models' contendo as redes neurais.
- A pasta 'labels' contendo as imagens de cada aluno cadastrado para o uso do reconhecimento facial.
- A pasta ‘css’ terá os comandos de estilos da página
- E a pasta ‘js’ onde estará os arquivos ‘index’ e ‘registroPresenca’



6.2. ARQUIVO HTML PRINCIPAL (INDEX.HTML)

Cabeçalho chamando o css e adicionando o nome da página:

```html
<!DOCTYPE html>
    <html lang="pt-BR">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Lista de Presença</title>
        <link rel="stylesheet" href="./assets/css/index.css">
    </head>
```

Cria-se o vídeo da câmera, define-se tamanho (width e height) além de multá-la para não captar o áudio também (muted).

```html
<video
    autoplay
    id="cam"
    width="594"
    height="462"
    muted
></video>
```

A tabela para marcar quem apareceu na câmera.

```
<div id="listaPresenca">
    <table id="tabela">
        <tr>
            <th>Aluno(a)</th>
            <th>Data / Hora</th>
        </tr>
    </table>
</div>
```

O botão para chamar o HTML que usamos para ver o registro de presença.

```
<a class="buttonPresencas" href="./registroPresenca.html">Consultar histórico de presenças</a>
```

Além do mais, faz a chamada de dois scripts, um que chama a face-api (face-api.min.js) e outro que chama o JavaScript do código (index.js).

```
<script src="./assets/lib/face-api/face-api.min.js"></script>
<script src="./assets/js/index.js"></script>
```

6.3. NO ARQUIVO JAVASCRIPT PRINCIPAL (INDEX.JS)

Está sendo chamado a face-api, com suas redes neurais e seu funcionamento principal.

Além disso, nesse mesmo arquivo está sendo feito o GET para os alunos cadastrados afim de comparar os rostos da tela, e o POST para cadastrar as presenças dos alunos que apareceram na câmera.

A primeira parte do código vai criar a Const ‘cam’ nomeada de ‘cam’ e as variáveis ‘listaPresenca’.

```
const cam = document.getElementById('cam');
var listaPresenca = [];
var alunos = [];
```

Agora, é criada a função 'fazGet(url)’, ela responsável por pegar determinados dados, e retornar uma resposta em forma de texto.

E a função ‘fazPost(url)’, que é responsável por posta esses dados.

```
function fazPost(url) {
    let request = new XMLHttpRequest()
    request.open("POST", url, true)
    request.setRequestHeader("Content-type", "text/plain")
    request.send()
}
```

```
function fazGet(url) {
    let request = new XMLHttpRequest()
    request.open("GET", url, false)
    request.send()
    return request.responseText
}
```

Aqui é criada a função responsável por registrar a presença, através de uma const criada chamada 'usuário' que vai localizá-lo para que possa ser efetuada a presença

```
function registroPresenca(label) {
    const usuario = alunos.find( aluno => aluno.nome == label );
    let url = `https://localhost:5000/PostRegistroPresenca?matricula=${usuario.matricula}`
    fazPost(url)
}
```

A função 'main' é a principal, que acaba iniciando todo o programa, nela foi criada para chamar ao função 'fazGet()', assim, pegando todas os alunos cadastrados no banco e coloca em formato 'JSON' na variável 'alunos'.

```
function main() {
    let data = fazGet("https://localhost:5000/api/Usuario/GetAllUsuarios");
    alunos = JSON.parse(data);
    console.log(alunos);
}
```

Nessa parte do código começa chamando a função 'main()'.

Depois, faz com que consiga escolher a câmera ser utilizada, isso é feito através do 'navigator.mediaDevices.enumerateDevices()', seguido para verificar se algum desses devices tem array, o que significa que tem dispositivo.

Seguindo, usa-se o 'devices.forEach' para encontrar um dispositivo de câmera, para isso foi feito o if (device.kind ='videoinput'), pois significa que é uma câmera

E por fim, faz um if para escolher a câmera desejada (device.label.includes('*nome da câmera*')), com uma função caso o programa tenha acesso a câmera (stream) e uma função caso não consiga acesso a câmera (error).

```
main()

const startVideo = () => {
    navigator.mediaDevices.enumerateDevices()
    .then(devices => {
        if (Array.isArray(devices)) {
            devices.forEach(device => {
                if (device.kind === 'videoinput') {
                    // Usar câmera especifica pelo 'label':
                    // console.log(device);

                    // if (device.label.includes('REDRAGON')) {
                    if (device.label.includes('')) {
                        navigator.getUserMedia(
                            { video: {
                                deviceId: device.deviceId
                            }},
                            stream => cam.srcObject = stream,
                            error => console.error(error)
                        )
                    }

                }
            })
        }
    })
}
```

Essa parte do código vai ser a responsável por reconhecer a pessoa eu aparecer na câmera, isso é feito através da const 'loadLabels = ()'. Dentro dela, é feito a "adição" dos alunos, para que a faceapi aprenda a reconhecer aquele aluno.

```
const loadLabels = () => {
    const labels = [];
    console.log(alunos.length);
    for (let i = 0; i < alunos.length; i++) {
        labels[i] = alunos[i];
    }
```

Essa parte do código é responsável por importar essas labels (imagens) "adicionadas" dos alunos, isso e feito através da const 'img = await faceapi.fetchImage(*caminho de onde encontrar as imagens*)'.

Depois, vamos criar uma const chamada 'detections' para detectar o Rosto (.detectSingleFace(img)), Traços (.withFaceLandmarks()) e a Descrição do rosto (.withFaceDescriptor()).

Por fim, vai ser retornado uma nova descrição para a label reconhecida.

```
    // const labels = ['Aluno1', 'Aluno2', 'Aluno3']
    return Promise.all(labels.map(async label => {
        const descriptions = []
        for (let i = 1; i <= 3; i++) {
            const img = await faceapi.fetchImage(`/assets/lib/face-api/labels/${label.matri
cula}/img${label.matricula}_${i}.jpg`)
            const detections = await faceapi
                .detectSingleFace(img)
                .withFaceLandmarks()
                .withFaceDescriptor()
            descriptions.push(detections.descriptor)
        }
        return new faceapi.LabeledFaceDescriptors(label.nome, descriptions)
    }))
}
```

Aqui foram importadas algumas redes neurais do faceapi, onde no fim chama a função 'startVideo', que é a câmera.

```
Promise.all([
    // Redes neurais:
    // Detectar rostos no vídeo
    faceapi.nets.tinyFaceDetector.loadFromUri('/assets/lib/face-api/models'),
    // Desenhar traços no rosto
    faceapi.nets.faceLandmark68Net.loadFromUri('/assets/lib/face-api/models'),
    // Reconhecimento facial
    faceapi.nets.faceRecognitionNet.loadFromUri('/assets/lib/face-api/models'),
    // Desenhar ao redor do rosto
    faceapi.nets.ssdMobilenetv1.loadFromUri('/assets/lib/face-api/models')
]).then(startVideo)
```

Começa-se adicionando um 'addEventListener('play')', que significa que quando a câmera iniciar, irá fazer algo.

Assim, cria-se um canvas para receber todas as informações que vai se iniciar. Seguindo, coloca-se o tamanho do canvas o mesmo da 'cam' e por fim, adiciona o canvas na página.

```
cam.addEventListener('play', async () => {
    const canvas = faceapi.createCanvasFromMedia(cam)
    const canvasSize = {
        width: cam.width,
        height: cam.height
    }
    const labels = await loadLabels()
    faceapi.matchDimensions(canvas, canvasSize)
    document.body.appendChild(canvas)
    setInterval(async () => {
```

Seguindo, vai faz com que o programa carregue as labels (imagens dos alunos), através da const 'labels = await loadLabels()'.

Agora, faz a faceapi dectar o que precisa-se através do 'faceapi.detectAllFaces()' com os parâmetros 'cam' e 'new faceapi.TinyFaceDetectorOptions'.

Em seguida, faz a faceapi detectar os traços, através da função '.withFaceLandmarks()'. E detectar as expressões através da função '.withFaceDescriptors()'.

Além disso, faz o faceapi adaptar essas detecções para o tamanho do vídeo, através da const ‘resizedDetections’, com os parâmetros ‘detections’ e ‘canvasSize’.

Na const ‘faceMatcher()’ adiciona a taxa de acerto que desejar, nesse caso 0.6 para essas labels.

Na const ‘results()’ é feita para procurar o melhor resultado para essas labels.

```
const detections = await faceapi
    .detectAllFaces(
        cam,
        new faceapi.TinyFaceDetectorOptions()
    )
    .withFaceLandmarks()
    .withFaceDescriptors()
const resizedDetections = faceapi.resizeResults(detections, canvasSize)
const faceMatcher = new faceapi.FaceMatcher(labels, 0.6)
const results = resizedDetections.map(d ⇒
    faceMatcher.findBestMatch(d.descriptor)
)
```

Para limpar a tela do desenho que a faceapi vai fazer a seguir, utiliza-se o ‘canvas.getContext(‘2d’).clearRect’ com os parâmetros do tamanho do ‘canvas’.

Em seguida, faz a faceapi desenhar o que ela conseguiu detectar através da função ‘faceapi.draw.drawDetections’, com os parâmetros ‘canvas’ e ‘resizedDetections’.

Agora, vai fazer com que a faceapi desenhe os traços através da função ‘faceapi.draw.drawFaceLandmarks’ com os parâmetros ‘canvas’ e ‘resizedDetections’.

Agora, vai fazer com que a faceapi desenhe as expressões através da função ‘faceapi.draw.drawFaceExpressions’ com os parâmetros ‘canvas’ e ‘resizedDetections’.

E para a idade e gênero, foi utilizado a função ‘new faceapi.draw.DrawTextField()’ descrita abaixo.

Para que apareça na tela novamente a descrição dessas labels, usa-se a função ‘results.forEach()’, que vai percorrer todos os resultados e a cada um deles, será desenhado uma nova caixa de texto na tela

E para obter as informações que quer mostrar os resultados, que no caso a label (nome) e a distance (o quanto acha que esse nome está certo), isso é feito através da const ‘{label, distance} = result’

Por fim, para desenhar esses resultados na tela, vai usar a função ‘new.faceapi.drwa. DrawTextField()’.

```
canvas.getContext('2d').clearRect(0, 0, canvas.width, canvas.height)

faceapi.draw.drawDetections(canvas, resizedDetections)
faceapi.draw.drawFaceLandmarks(canvas, resizedDetections)
results.forEach((result, index) => {
    const box = resizedDetections[index].detection.box
    const { label, distance } = result
    new faceapi.draw.DrawTextField([
        `${label} (${parseInt(distance * 100, 10)})`
    ], box.bottomLeft).draw(canvas)
```

Cria-se 6 variáveis:

- 'data'.
- 'dia'.
- 'mes'.
- 'ano'.
- 'hora'.
-'min'.

Seguindo, criamos 4 ifs para deixar a data e a hora na forma correta.

Por fim, criamos a const 'dataFormatada' que mostra a data e hora na forma correta.

```
var data = new Date();
var dia = data.getDate();
var mes = (data.getMonth() + 1);
var ano = data.getFullYear();
var hora = data.getHours();
var min = data.getMinutes();

if (dia <= 9)
    dia = `0${dia}`
if (mes <= 9)
    mes = `0${mes}`
if (hora <= 9)
    hora = `0${hora}`
if (min <= 9)
    min = `0${min}`

const dataFormatada = `${dia}/${mes}/${ano} ${hora}:${min}`
```

Esse primeiro if é responsável por escrever "unknown" em vez de salvar todas as pessoas (label) desconhecidas no sistema que aparecem na câmera.

Já o segundo if é responsável por não salva o mesmo aluno toda vez que ele passar, se já tem uma vez, não irá salvar de novo até reiniciar o programa

```
            if (label ≠ "unknown") {
                if (!listaPresenca.includes(label)) {
                    let linha = criaLinha(label, dataFormatada);
                    tabela.appendChild(linha);
                    registroPresenca(label)
                }
            }
        })
    }, 100)
})
```

Aqui, a função 'criaLinha()' vai trazer as seguintes variáveis, e depois retorna 'linha':

- 'linha': que recebe "tr".
- 'tdNome: que recebe "td".
- 'tdDate': que recebe "td".
- 'tdNome.innerHTML': que recebe label.
- 'tdDate.innerHTML': que recebe date.

```
function criaLinha(label, date) {
    linha = document.createElement("tr");
    tdNome = document.createElement("td");
    tdDate = document.createElement("td");
    tdNome.innerHTML = label
    tdDate.innerHTML = date

    linha.appendChild(tdNome);
    linha.appendChild(tdDate);

    let i = listaPresenca.length
    listaPresenca[i] = label

    return linha;
}
```

6.4. ARQUIVO DE ESTILO CSS (INDEX.CSS) PARA O HTML PRINCIPAL (INDEX.HTML)

No body coloca-se os estilos para a imagem que irá parecer, no caso:

- Coloca-se sem margem (margin) e sem preenchimento (padding).
- Altura (height) e largura (width).
- Coloca-se um fundo com a imagem do logo do IF (background).
- Coloca-se cor (color).
- A exibição deixa-se flexível (display: flex).
- O alinha-se no centro (align-items: center).
- Especifica-se o tamanho da fonte (font-sizr: 18px).
- Especifica-se as fontes aceitas (font-family).

```css
body {
    margin: 0;
    padding: 0;
    width: 100vw;
    height: 100vh;
    background: #FFF url("../imgs/IF-Logo.png")
 no-repeat left top fixed;
    color: #333333;
    display: flex;
    align-items: center;
    font-size: 18px;
    font-family: 'Trebuchet MS', 'Lucida Sans Un
icode', 'Lucida Grande', 'Lucida Sans', Arial, s
ans-serif;
}
```

No vídeo e no Canvas coloca-se os estilos:

- (Positivo: absoluta) para deixar a posição fixa.
- (Left: 50px) para ficar a 50 pixels da esquerda.

```css
video {
    position: absolute;
    left: 50px;
}

canvas {
    position: absolute;
    left: 50px;
}
```

Na #ListaPresenca coloca-se os estilos:

- (Positivo: absoluta) para deixar a posição fixa.
- (Right: 100px) para ficar a 100 pixels da direita.

No tr, th e td coloca-se apenas o preenchimento de 10px e 20px.

```css
#listaPresenca {
    position: absolute;
    right: 100px;
}

tr, th, td {
    padding: 10px 20px;
}
```

Especificadamente, no th coloca-se também os seguintes estilos:

- (Border) de 2px, sólida e coloca-se uma cor.
- (Font-sizr) definindo o tamanho da fonte de 20 pixels.
- (Color) para colocar uma cor.
- (Background-color) definindo uma cor Mara o fundo.

Já no td coloca-se somente o seguinte estilo:

- (Background-color) definindo uma cor para o fundo.

```css
th {
    border: 2px solid #1B420C;
    font-size: 20px;
    color: #FFF;
    background-color: #377724;
}

td {
    background-color: #E8E7E7;
}
```

No .buttonPresencas os estilos adicionados foram:

- (Positivo: absoluta) para deixar a posição fixa.

- (Right: 10px) para ficar a 10 pixels da direita.

- (Bottom: 10px) para ter 10 pixels de fundo.

E para finalizar o arquivo de estilo css 'index.css', o a:link, a:visited e a:hover, ganharam os seguintes estilos:

- (Color) para colocar uma cor.

- (Text-decoration), no caso, underLine (sob a linha).

```css
.buttonPresencas {
    position: absolute;
    right: 10px;
    bottom: 10px;
}

a:link {
    color: #377724;
    text-decoration: underline;
}

a:visited {
    color: #377724;
    text-decoration: underline;
}

a:hover {
    color: #1B420C;
    text-decoration: underline;
}
```

.

6.5. ARQUIVO HTML (REGISTROPRESENCA.HTML)

Através dele irá consultar o registro de presença.

Nele contém:

- um filtro de prontuário.

- data/hora inicial e data/hora final.

Cabeçalho chamando o css e adicionando o nome da página:

```html
<!DOCTYPE html>
    <html lang="pt-BR">
    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Registro de Presença</title>
        <link rel="stylesheet" href="./assets/css/registroPresenca.css">
    </head>
```

Aqui, primeiramente chama-se o script, que chama o JavaScript do código (registroPresenca.js).

Após, é criado um link em botão (buttonVoltar) para voltar para página ‘index.js’.

```
    <body>
        <script src="./assets/js/registroPresenca.js"></script>

        <a class="buttonVoltar" href="./index.html"><img class="iconVoltar" src="./assets/imgs/arrow_back.s
vg" alt="Voltar"/></a>

        <h1>Consultar histórico de presenças</h1>
```

Aqui é criado um form responsável por filtrar os prontuários, chama a função ‘filtroPresenca()’.

São criadas três labels diferentes:

- ‘matricula’: que irá receber o prontuário.
- ‘dtInicio’: que irá receber a data início.
- ‘dtFim’: que por sua vez, irá receber a data fim.

Por fim, nessa parte, também é criado o botão (button) para consulta chamado ‘consulta’.

```
        <form id="filtro" style='display: block;' onsubmit="filtroPresenca()">
            <label for="matricula">Prontuário</label>
            <input name="matricula" id="matricula" style="text-transform: uppercase" type="text">
            <br><br>
            <label for="dtInicio">Data Inicio</label>
            <input name="dtInicio" id="dtInicio" type="datetime-local">
            <label for="dtFim">Data Fim</label>
            <input name="dtFim" id="dtFim" type="datetime-local">
            <br><br>
            <button id="consulta" class="buttonConsulta" style='display: block; margin: 0 0 0 auto;' type=
"submit">Consultar</button>
        </form>
```

Nessa parte do código, é criado outro botão para realizar uma nova consulta chamado ‘novaConsulta’ já aplicando seu estilo, que chama a função ‘novaConsulta()’

Após isso, é criado uma tabela com os parâmetros:

- Matrícula.
- Data/Hora.

```
        <div id="consulta">
            <button id="novaConsulta" class="buttonConsulta" style='display: none; margin: 0 0 10px auto;'
onclick="novaConsulta()">Nova Consulta</button>
            <table id="tabela" style='display: none;'>
                <tr>
                    <th>Matrícula</th>
                    <th>Data / Hora</th>
                </tr>
            </table>
        </div>
```

Para finalizar o arquivo HTML (registroPresenca.html), e colocado uma imagem (img) do logo do IF.

```
        <img class="logo" src="./assets/imgs/IF-Logo.png" alt="IF">
    </body>
</html>
```

6.6. ARQUIVO JAVASCRIPT (REGISTROPRESENCA.JS)

Feito para fazer o POST do filtro e ter o retorno dos dados específicos.

Além de fazer o controle da tabela dinâmica de acordo com as informações do filtro.

Começa-se criando a função ‘filtroPresenca()’, nela, chama-se a função ‘visualizacao()’.

Cria-se 4 variaveis let:

- ‘url’: que recebe o link para o ‘GetByFilter’

- ‘matricula’: que receber o documento do getElementById ‘matricula’ transformado em um valor.

- ‘dtInicio’: que receber o documento do getElementById ‘dtInicio’ transformado em um valor.

- ‘dtFim’: que receber o documento do getElementById ‘dtFim’ transformado em um valor.

Seguindo, foram criados dois ifs, para caso ‘dtInicio’ e ‘dtFinal’ forem vazios, receberem um valor.

Finalizando com a criação do parâmetro que faz a variável ‘matricula’ sempre ser maiúscula (toUpperCase()). Para assim, chamar a função ‘fazPost’ com os parâmetros ‘url’ e ‘body’.

```
function filtroPresenca() {

    visualizacao()

    let url = "https://localhost:5000/GetByFilter"

    let matricula = document.getElementById("matricula").value
    let dtInicio = document.getElementById("dtInicio").value
    let dtFim = document.getElementById("dtFim").value

    if (dtInicio == "")
        dtInicio = "0001-01-01"
    if (dtFim == "")
        dtFim = "0001-01-01"

    body = {
        "matricula": matricula.toUpperCase(),
        "dtInicio": dtInicio,
        "dtFim": dtFim,
    }

    event.preventDefault()

    fazPost(url, body)
}
```

Cria-se a função ‘fazPost()’ com os parâmetros de url e body. nela e criada as variáveis:

- ‘request’.

- ‘presencas’ transformando-a em formato ‘JSON’.

- ‘tabela’ que recebe o valor ‘tabela’.

- ‘linha’.

```
function fazPost(url, body) {
    let request = new XMLHttpRequest()
    request.open("POST", url, true)
    request.setRequestHeader("Content-type", "application/json")
    request.send(JSON.stringify(body))

    request.onload = function() {
        let presencas = JSON.parse(this.responseText)
        let tabela = document.getElementById("tabela");
        presencas.forEach(element => {
            let linha = criaLinha(element);
            tabela.appendChild(linha);
        });
    }
}
```

Função ‘criaLinha(presencas)’, nela coloca-se os valores para as seguintes variáveis:

- ‘linha’: recebe “tr”.
- ‘tdMatricula’: “td”.
- ‘tdDate’: “td”.

Cria-se 6 variáveis:

- ‘data’;
- ‘dia’.
- ‘mes’.
- ‘anos’.
- ‘hora’.
- ‘min’.

Seguindo, cria-se 4 ifs para deixar a data e a hora na forma correta.

```
function criaLinha(presencas) {
    linha = document.createElement("tr");
    tdMatricula = document.createElement("td");
    tdData = document.createElement("td");

    let data = new Date(presencas.dataPresenca)
    var dia = data.getDate();
    var mes = (data.getMonth() + 1);
    var ano = data.getFullYear();
    var hora = data.getHours();
    var min = data.getMinutes();

    if (dia <= 9)
        dia = `0${dia}`
    if (mes <= 9)
        mes = `0${mes}`
    if (hora <= 9)
        hora = `0${hora}`
    if (min <= 9)
        min = `0${min}`
```

Cria-se a const ‘dataFormatada’ que mostra a data e hora na forma correta.

E depois retorna o valor de ‘linha’.

```
    const dataFormatada = `${dia}/${mes}/${ano} ${hora}:${min}`

    tdMatricula.innerHTML = presencas.matricula
    tdData.innerHTML = dataFormatada

    linha.appendChild(tdMatricula);
    linha.appendChild(tdData);

    return linha;
}
```

Cria-se a função ‘visualizacao()’, nela é criada 4 variáveis:

- ‘consulta’ que é setada como “none”.
- ‘novaConsulta’ que é setada como “block”.
- ‘form’ que é setada como “none”.
- ‘tabela’ que é setada como “block”.

```
function visualizacao() {
    var consulta = document.getElementById("consulta");
    var novaConsulta = document.getElementById("novaConsulta");
    var form = document.getElementById("filtro");
    var tabela = document.getElementById("tabela");
    consulta.style.display = "none";
    novaConsulta.style.display = "block";
    form.style.display = "none";
    tabela.style.display = "block";
}
```

Cria-se a função ‘novaConsulta()’.

```
function novaConsulta() {
    location.reload()
}
```

6.7. ARQUIVO DE ESTILO CSS (REGISTROPRESENCA.CSS) PARA O HTML DO REGISTRO DE PRESENÇA

No body coloca-se os estilos para a imagem que irá parecer, no caso:

- Coloca-se sem margem (margin) e sem preenchimento (padding).
- Coloca-se um fundo com a imagem do logo do IF (background).
- Coloca-se cor (color).
- A exibição deixa-se flexível (display: flex).
- A direção flexível coloca-se em coluna (flex-direction: column).
- O alinha-se no centro (align-items: center).
- Justifica-se o conteúdo no centro (justify-content: center).
- Especifica-se o tamanho da fonte (font-sizr: 18px).
- Especifica-se as fontes aceitas (font-family).

```
body {
    margin: 0;
    padding: 0;
    background: #FFF url("../imgs/IF-Logo.png") no-repeat left top f
ixed;
    color: #333333;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    font-size: 18px;
    font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Gran
de', 'Lucida Sans', Arial, sans-serif;
}
```

No .buttonVoltar, os estilos adicionados foram:

- (Position: fixed) que deixa a posição fixa.
- (Height: 50px) que deixa 50 pixels de altura.
- (Left: 10px) que deixa o botão 10 pixels da esquerda.
- (Top: 10px) que deixa o botão 10 pixels do topo.

```
.buttonVoltar {
    position: fixed;
    height: 50px;
    left: 10px;
    top: 10px;
}
```

No .buttonConsulta, os estilos adicionados foram.

- (Background-color) que escolhe a cor do fundo.
- (Border:nome) que deixa sem borda.
- (Padding) que deixa com preenchimento 5 e 10 pixels.
- (Border-radius) que deixa o raio da marmita em 5 pixel.
- (Cursor) que deixa o curso em forma de ponteiro.

```
.buttonConsulta {
    background-color: #81e264;
    border: none;
    padding: 5px 10px;
    border-radius: 5px;
    cursor: pointer;
}
```

No .buttonConsulta:hover foi adicionando o estilo:

- (Background-color) para colocar cor no fundo.

```
.buttonConsulta:hover {
    background-color: #9ee28a;
}
```

No .logo, foram adicionados os seguintes estilos:

- (Position: fixed) que deixa a posição fixa.
- (Height: 50px) que deixa 50 pixels de altura.
- (Right: 10px) que deixa o botão 10 pixels da direita.
- (Bottom: 10px) que deixa o botão 10 pixels de fundo.

```
.logo {
    position: fixed;
    height: 50px;
    right: 10px;
    bottom: 10px;
}
```

No #registroPresenca, os estilos que foram adicionados:

- (Position:static) que deixa a posição parada.

- (Margin) que deixa a margem com 20 pixels de um lado e 0 do outro.

Além disso, tanto tr, com o th e o td tiveram o seguinte estilo adicionado igualmente:

- (Padding)que deixa o preenchimento de 10 pixels de um lado e 20 pixels do outro.

```
#registroPresenca {
    position: static;
    margin: 20px 0;
}

tr, th, td {
    padding: 10px 20px;
}
```

Especificadamente, no th coloca-se também os seguintes estilos:

- (Border) de 2px, sólida e coloca-se uma cor.

- (Font-sizr) definindo o tamanho da fonte de 20 pixels.

- (Color) para colocar uma cor.

- (Background-color) definindo uma cor mara o fundo.

Já no td coloca-se somente o seguinte estilo:

- (Background-color) definindo uma cor mara o fundo.

```
th {
    border: 2px solid #1B420C;
    font-size: 20px;
    color: #FFF;
    background-color: #377724;
}

td {
    background-color: #E8E7E7;
}
```

6.8. FUNCIONAMENTO DO RECONHECIMENTO NO CÓDIGO DA APLICAÇÃO

Const ‘img = await faceapi.fetchImage está buscando as imagens, que podem estar em um banco de dados externo ou interno, para que seja realizada a análise das faces, que é feita por meio da comparação dos pontos dos olhos, boca e contorno do rosto. Após detectar as descrições do rosto, retornamos essas informações para uma nova instância do faceapi.

```
    // const labels = ['Aluno1', 'Aluno2', 'Aluno3']
    return Promise.all(labels.map(async label => {
        const descriptions = []
        for (let i = 1; i <= 3; i++) {
            const img = await faceapi.fetchImage(`/assets/lib/face-api/labels/${label.matri
cula}/img${label.matricula}_${i}.jpg`)
            const detections = await faceapi
                .detectSingleFace(img)
                .withFaceLandmarks()
                .withFaceDescriptor()
            descriptions.push(detections.descriptor)
        }
        return new faceapi.LabeledFaceDescriptors(label.nome, descriptions)
    }))
}
```

Com o reconhecimento já processado, continua-se apresentando a imagem na tela e faz o desenho dos traços analisados na etapa anterior.

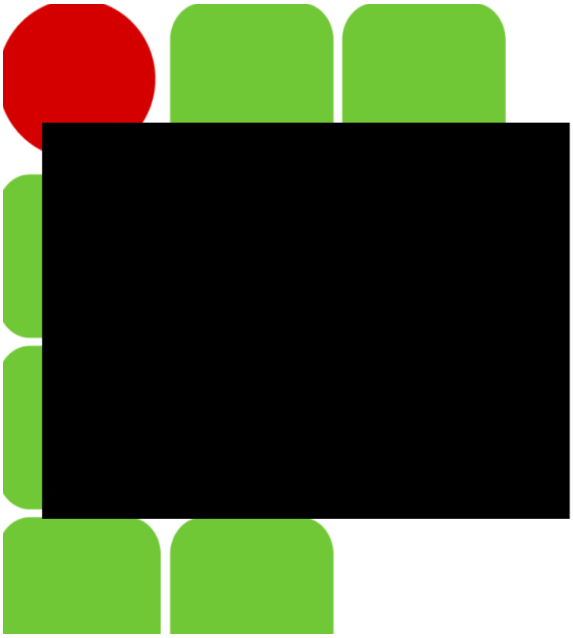
7. EXPLICANDO O FUNCINAMENTO DE CADA ETAPA DA APLICAÇÃO

7.1. PÁGINA INICIAL

Para abrir o programa, dê dois cliques no ícone indicado.

Abaixo apresenta-se a tela inicial do programa. Devemos esperar a API carregar para fazermos o reconhecimento facial.

A imagem aparecerá nesta seção indicada na cor preta.



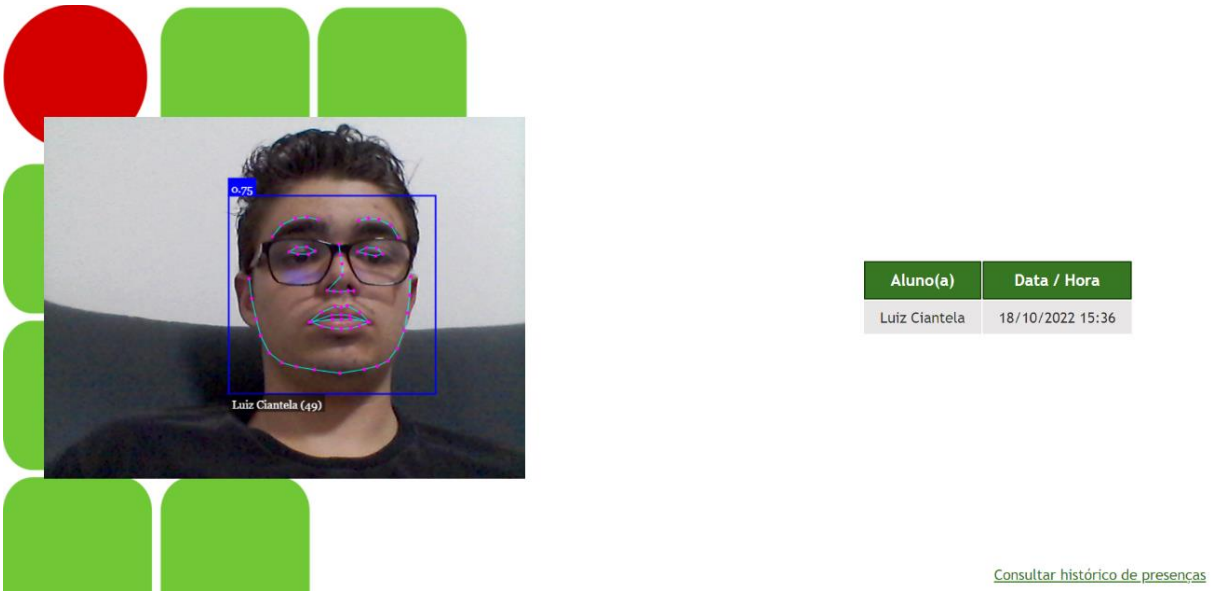
A tabela abaixo é criada automaticamente quando o reconhecimento facial é feito.

Aluno(a)	Data / Hora
Luiz Ciantela	18/10/2022 15:36

O link “Consultar histórico de presenças” leva à tela do registro das presenças (8.2).

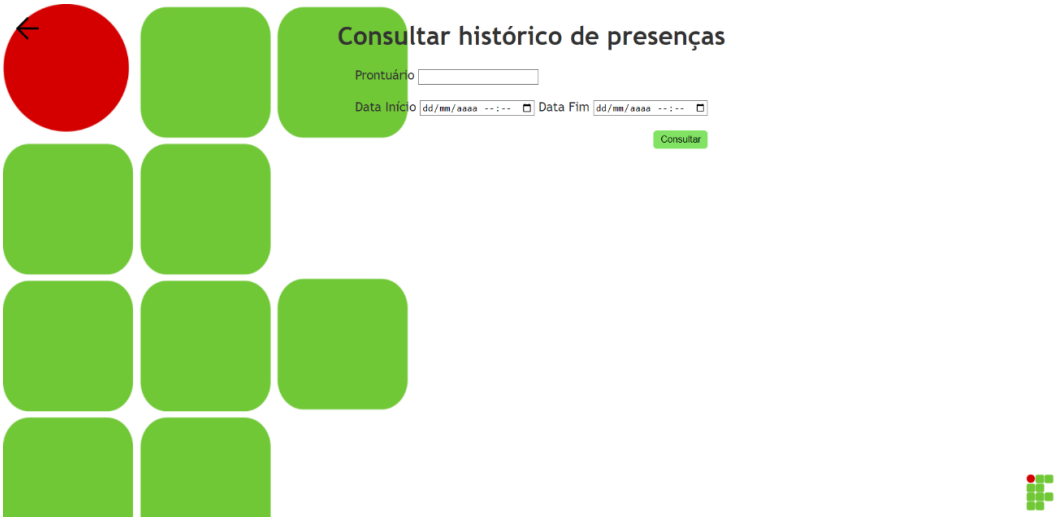
Tela carregada e pronta para realizar o reconhecimento.

Exemplo de um rosto sendo reconhecido pela API, identificado pelo nome. Há também a porcentagem de quão precisa é a captura do rosto com a rede neural em uma escala de zero a cem por cento.



7.2 PÁGINA DE CONSULTA DO HISTÓRICO DE PRESENÇAS

Tela de consulta do histórico de presenças.



No botão abaixo, o usuário retorna à tela inicial.

Abaixo é indicado o local para digitar o prontuário ao qual irá ser consultado o histórico, junto com a data de início e fim para verificação do tempo desse histórico.

IMPORTANTE - Caso não seja especificado nenhum prontuário ou data, será apresentado todo o histórico cadastrado no banco.

Caso seja inserido apenas o prontuário e não as datas, será trago todo o histórico do prontuário inserido. Da mesma forma ocorre com as datas quando o prontuário não é inserido. Abaixo um exemplo de consulta feita ao prontuário escolhido.

O botão ‘Nova consulta’ retorna ao layout padrão (7.2).

REFERÊNCIAS

MENEZES, Karina. O que é reconhecimento facial. **IdBlog**, 2020. Disponível em: <https://blog.idwall.co/o-que-e-reconhecimento-facial/>. Acesso em: 01 de maio de 2022.

KLEINA, Nilton. Como funciona o reconhecimento facial. **TecMundo**, 2021. Disponível em: <https://www.tecmundo.com.br/camera-digital/10347-como-funcionam-os-sistemas-dereconhec imento-facial.htm>. Acesso em: 01 de maio de 2022.

TUDO que você precisa saber sobre reconhecimento facial. **Rtm**, 2020. Disponível em: <https://www.rtm.net.br/tudo-que-voce-precisa-saber-sobre-reconhecimento-facial/>. Acesso em: 01 de maio de 2022.

GOMES, Helton Simões. Como funciona o reconhecimento facial? Entenda a tecnologia que lê o rosto. **UOL**, 2018. Disponível em: <https://www.uol.com.br/tilt/ noticias/redacao /2018/10/11/entenda-atecnologia-por-tras-do-reconhecimento-facial.htm>. Acesso em: 01 de maio de 2022.

TAVARES, Henrique. IBM Watson: aprenda tudo sobre a ferramenta de inteligência de dados. **Blog da Oncase**, 2021. Disponível em: <https://www.oncase.com.br/blog/ferram entas-analiticas/tudosobre-ibm-watson/>. Acesso em: 01 de maio de 2022.

CARLETO, Diogo. Face-api.js: reconhecimento facial em JavaScript. **InfoQ**, 2018. Disponível em: <https://www.infoq.com/br/news/2018/11/faces-api-js/>. Acesso em: 01 de maio de 2022.

JAVASCRIPT API for face detection and face recognition in the browser implemented on top of the tensorflow.js core API. **GitHub**, 2020. Disponível em: <https://justadudewh ohacks.github.io/faceapi.js/docs/index.html>. Acesso em: 01 de maio de 2022.

FACE-api,js. **GitHub**, 2020. Disponível em: <https://github.com/justadudewhohacks/fa ce-api.js>. Acesso em: 01 de maio de 2022.

PASSARELLI, Leandro. Aplicação de visão computacional com OpenCV. **EMBARCADOS**, 2016. Disponível em: <https://www.embarcados.com.br/aplicacao-de-visao-computacional-com-opencv/>. Acesso em: 01 de maio de 2022.

DIAS, Tiago. Análise de Imagens com OpenCV. **Dados ao Cubo**, 2022. Disponível em: <https://dadosaocubo.com/analise-de-imagens-com-opencv/>. Acesso em: 01 de maio de 2022.

BERTOLETI, Pedro. OpenCV - o que é, onde usar e como instalar na Raspberry Pi (MIC412). **Instituto NBC**, 2020. Disponível em: <https://www.newtoncbraga.com.br/index.php/microcontroladores/143tecnologia/17799-opencv-o-que-e-onde-usar-e-como-instalar-na-raspberry-pi-mic412.html>. Acesso em: 01 de maio de 2022.

OPEN CV. **Open CV**, 2022. Disponível em: <https://opencv.org>. Acesso em: 01 de maio de 2022.

QUAL é a diferença de API, biblioteca e framework. **Stack Overflow**, 2014. Disponível em: <https://pt.stackoverflow.com/questions/17501/qual-%C3%A9-a-diferen%C3%A7a-de-apibibliotecaeframework>. Acesso em: 01 de maio de 2022.

O que é uma API. **AWS**, 2022. Disponível em: <https://aws.amazon.com/pt/what-is/api/>. Acesso em: 01 de maio de 2022.