

## APRESENTAÇÃO DA ATIVIDADE

---

No ramo automobilístico é perceptível a evolução tecnológica constante não apenas no produto principal o automóvel, mas também internamente nas montadoras, sempre inovando em algum processo tendo o alvo redução de custos, tempo e aumentando cada vez mais a qualidade. Tendo como base esses pilares, para que a mudança seja assertiva é de extrema necessidade a execução de ensaios, simulações e coleta de dados para geração de indicadores diversos, como validação de quando uma mudança/intervenção se faz necessária.

Para atividade que estamos propondo serão criados três grupos, onde os integrantes trabalharão com robótica, sistema de visão, gestão e disponibilização de dados por IOT, coleta e gestão de dados para geração de informação visual usando computador, tablet ou celular e disponibilização de dados por meio de realidade aumentada.

No campo da robótica vamos realizar a digitalização de um processo que conta com 3 robôs colaborativos responsáveis pelo transporte entre células, controle de qualidade utilizando sistema de visão e por fim o armazenamento do produto classificado em condição de aprovado ou reprovado.

No campo da informática teremos duas equipes distintas uma coletando dados do robô e disponibilizando para exibição. Outra equipe será responsável por coletar estes dados e gerir a forma e como eles serão apresentados ao usuário final.

## EQUIPE DE DESENVOLVIMENTO FULL-STACK E REALIDADE AUMENTADA

---

Esta equipe será responsável pela coleta dos dados disponibilizados por APIs e a disponibilização destes dados em formato mais amigável.

Criaremos um pequeno Dashboard usando HTML, CSS, JAVASCRIPT e a biblioteca CHART.JS para exibição dos dados de forma amigável.

Também veremos como criar uma aplicação em Realidade Aumentada usando AFrame e MindAR, bibliotecas Javascript para este propósito.

## API REST

---

Rest significa Representationel State Transfer (Transferência de Estado Representacional) que é um tipo de arquitetura de software. Uma arquitetura REST então se refere a um conjunto de regras que devem ser seguidas no desenvolvimento da aplicação.

Para se enquadrar na arquitetura devem seguir as seguintes regras:

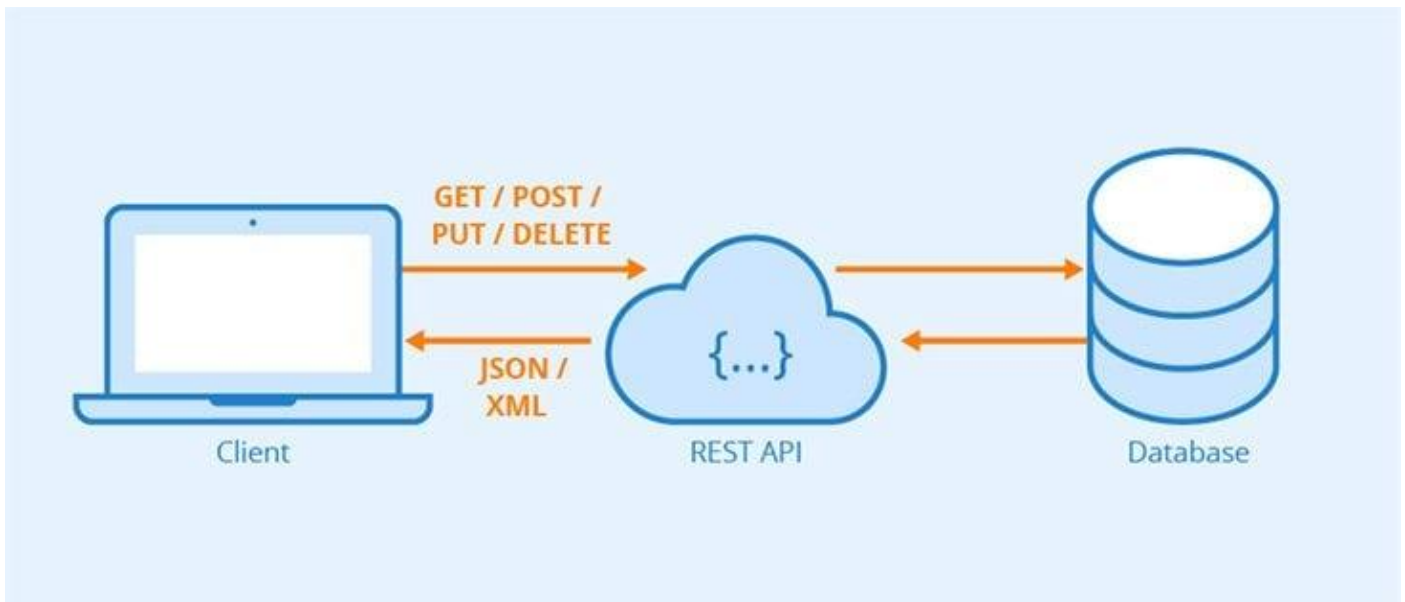
Arquitetura cliente-servidor: Baseada em solicitações HTTP.

Comunicação stateless: Não deve armazenar nenhuma informação.

Cache: Precisa conseguir armazenar dados em Cache.

Interface uniforme: Precisa conter uma interface uniforme pois oferecerá uma comunicação padronizada entre usuário e software, a manipulação de recursos é feita como JSON ou XML.

Sistema de camadas: As camadas devem possuir uma funcionalidade específica como segurança ou carregamento.



Usaremos o Node.js para criar nossas API's com base em Javascript.

Para instalar o Node-RED é necessário ter o Node instalado, pelo link a seguir ( <https://nodejs.org/en/> ) podemos baixar a versão LTS mais recente do Node, sempre baixa e instale a versão LST.

Após a instalação do Node já podemos construir nossas apis.

## *CRIANDO UMA API BÁSICA EM NODE*

O código a seguir representa uma API básica que retorna um JSON contendo informações de produção representando peças boas, peças ruins, peças totais, timestamp, temperatura, no formato de objeto literal Javascript.

```
{
  "pecasboas": "77",
  "pecasruins": "98",
  "pecastotais": "175",
  "timestamp": "1674062682031",
  "temperatura": "40"
}
```

Abra o Visual Studio Code ou o editor de sua preferência e digite o código a seguir, crie um novo arquivo chamado mbi.js e digite o código a seguir.

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'application/json');
  res.setHeader('Access-Control-Allow-Origin', '*');

  let pb = Math.floor(Math.random()*100)
  let pr = Math.floor(Math.random()*100)
  let pt = pb+pr
  let dados = {
    pecasboas:pb.toString(),
    pecasruins:pr.toString(),
    pecastotais:pt.toString(),
    timestamp:(new Date().getTime()).toString(),
    temperatura:(Math.floor(Math.random()*65)+35).toString()
  }
})
```

```
res.end(JSON.stringify(dados))
});

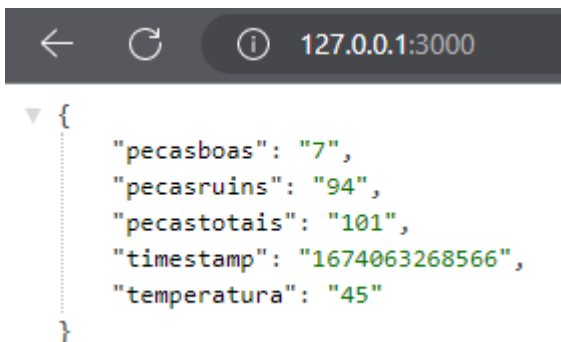
server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

O que este código faz é criar um servidor na máquina local (127.0.0.1 / localhost) escutando na porta 3000, em seguida gera alguns números aleatórios para peças boas “pb” e peças ruins “pr”, calcula peças totais “pt”, cria o objeto dados que conterá todas as informações retornadas pela API e no final, com a função “end” envia os dados para quem fez a requisição.

Para rodar o servidor, abra o prompt, navegue até a pasta e digite o comando “node mbi.js”.

```
C:\Senai\MBI-Automotiva>node mbi.js
Server running at http://127.0.0.1:3000/
```

Quando servidor estiver rodando abra o browser e digite o end-point <http://127.0.0.1:3000/>, sem as aspas, ao pressionar ENTER será retornado o JSON com os dados gerados pela API.



```
{
  "pecasboas": "7",
  "pecasruins": "94",
  "pecastotais": "101",
  "timestamp": "1674063268566",
  "temperatura": "45"
}
```

Como usamos a função “random” a cada vez que for chamada a API irá retornar valores diferentes.

## CONSTRUINDO APIS DE FORMA RÁPIDA E ROBUSTA COM NODE-RED

Node-RED é uma ferramenta de desenvolvimento organizada em fluxo baseada em Javascript para construção de aplicações Back-End com objetivo de conectar dispositivos de hardware, criar APIs e serviços online, atende muito bem aos requisitos de desenvolvimento para Internet das Coisas.

Pelo site oficial do Node-RED ( <https://nodered.org/> ) é possível obter todas as informações necessárias inclusive os guias de iniciação e instalação ( <https://nodered.org/docs/getting-started/> ) para diversos dispositivos e sistemas operacionais como Windows ( <https://nodered.org/docs/getting-started/windows> ).

Após instalado vamos iniciar o Node-RED, pelo prompt vamos digitar o comando “node-red” e pressionar ENTER.

```
Microsoft Windows [vers o 10.0.19045.2480]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\SENAIU.0>node-red
18 Jan 14:23:52 - [info]
Welcome to Node-RED
*****

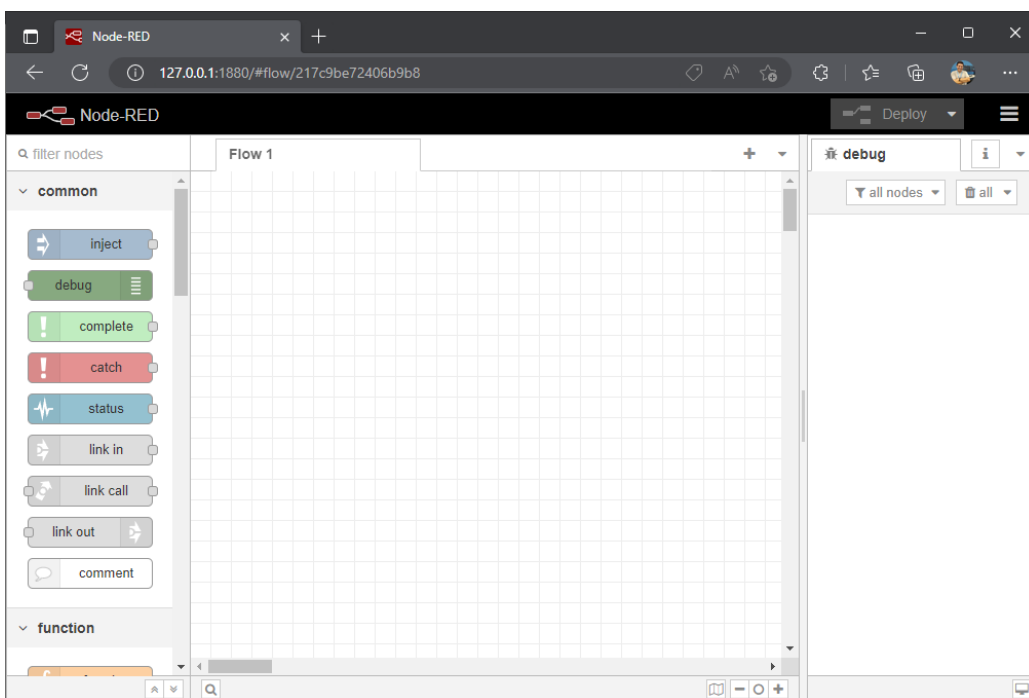
18 Jan 14:23:52 - [info] Node-RED version: v3.0.2
18 Jan 14:23:52 - [info] Node.js version: v18.12.1
18 Jan 14:23:52 - [info] Windows_NT 10.0.19045 x64 LE
18 Jan 14:23:53 - [info] Loading palette nodes
18 Jan 14:23:54 - [info] Settings file : C:\Users\SENAIU.0\.node-red\settings.js
18 Jan 14:23:54 - [info] Config store : 'default' [module=memory]
18 Jan 14:23:54 - [info] User directory : \Users\SENAIU.0\.node-red
18 Jan 14:23:54 - [warn] Projects disabled : editorTheme.projects.enabled=false
18 Jan 14:23:54 - [info] Flows file : \Users\SENAIU.0\.node-red\flows.json
18 Jan 14:23:54 - [warn]

Your flow credentials file is encrypted using a system-generated key.
If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.
You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.

18 Jan 14:23:54 - [warn] Encrypted credentials not found
18 Jan 14:23:54 - [info] Server now running at http://127.0.0.1:1880/
18 Jan 14:23:54 - [info] Starting flows
18 Jan 14:23:54 - [info] Started flows
```

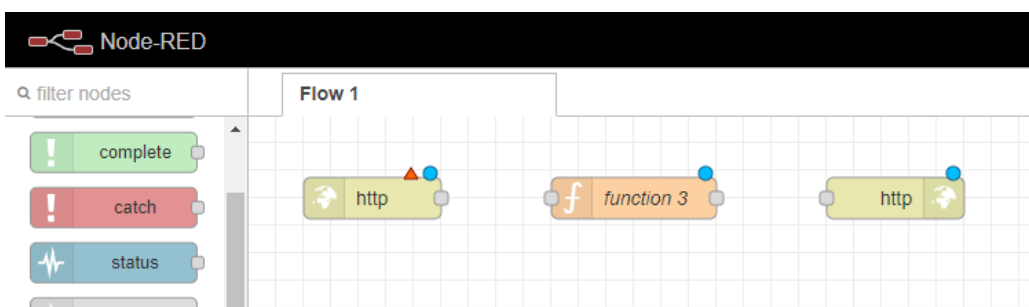
O servidor ser  iniciado e por padr o roda em ( 127.0.0.1 / localhost ) na porta 1880.

Para abrir o ambiente de desenvolvimento abra o browser e digite o endere o “127.0.0.1:1880” sem aspas. Ao pressionar ENTER o ambiente ser  carregado.

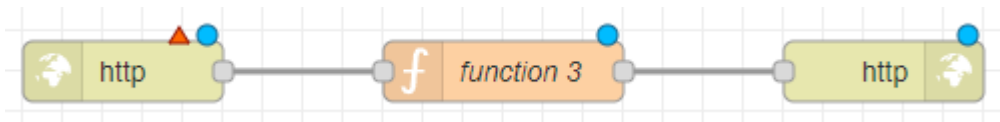


Vamos construir uma API semelhante a que fizemos com Node puro.

O Node-RED   uma ferramenta Low Code, de arrastar e soltar, sendo assim procure os n s http in, http response e function, posicione-os conforme a ilustra  o a seguir.



Como   uma ferramenta baseada em fluxo fa a a conex o dos n s conforme a ilustra  o a seguir.



Aplicando um clique duplo no nó http in podemos editar suas configurações, conforme a ilustração a seguir digite “/mbi” no campo “URL” e clique em “Done”.

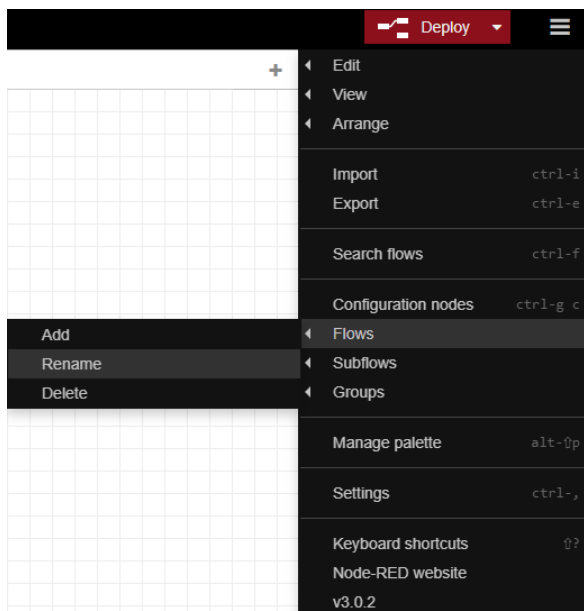
The screenshot shows the 'Edit http in node' dialog box. At the top, there are three buttons: 'Delete', 'Cancel', and 'Done'. Below the buttons is a 'Properties' section with a gear icon, a document icon, and a preview icon. The 'Properties' section contains three fields: 'Method' with a dropdown menu showing 'GET', 'URL' with a text input field containing '/mbi', and 'Name' with a text input field containing 'Name'.

Agora aplique um clique duplo no nó http response e configure conforme a ilustração a seguir, usando o botão “+ add”.

The screenshot shows the 'Edit http response node' dialog box. At the top, there are three buttons: 'Delete', 'Cancel', and 'Done'. Below the buttons is a 'Properties' section with a gear icon, a document icon, and a preview icon. The 'Properties' section contains three fields: 'Name' with a text input field containing 'Name', 'Status code' with a text input field containing 'msg.statusCode', and 'Headers' with a list of headers. The 'Content-Type' header is selected, and its value is 'application/json'.

Clique em “Done” após a configuração.

Para renomear o fluxo de trabalho atual clique no MENU → Flows → Rename e digite o nome “MBI Automotiva 4.0”



Agora vamos ao código principal, aplique um clique duplo no nó “function” e digite o código a seguir, na aba “On Message”.

```
let pb = flow.get("pb") || 0
let pr = flow.get("pr") || 0
if (Math.random() * 100 > 10) {
  pb++
} else {
  pr++
}
let pt = pb + pr
flow.set("pb", pb)
flow.set("pr", pr)
let dados = {
  pecasboas: pb,
  pecasruins: pr,
  pecastotais: pt,
  timestamp: new Date().getTime(),
  temperatura: Math.floor(Math.random() * 65) + 35
}
msg.payload = dados
return msg;
```

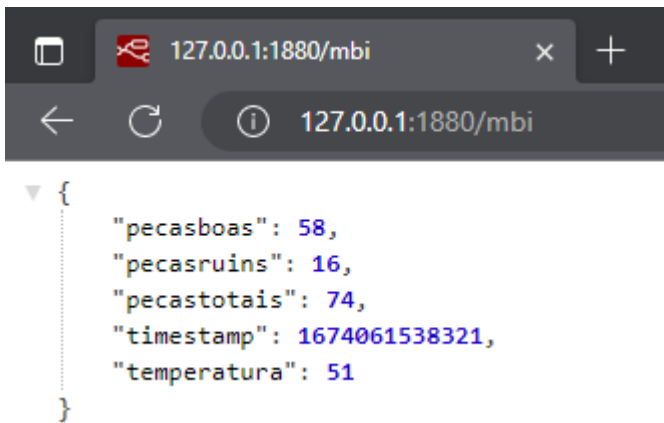
Este código tem um comportamento semelhante ao primeiro, com algumas particularidades do Node-RED.

Usando a função get da classe flow geramos as variáveis globais ao fluxo chamadas “pb” e “pr”, geramos um valor aleatório para decidir se a peça gerada será uma peça boa ou ruim e incrementa as devidas variáveis “pb” para peças boas e “pr” para peças ruins.

O restante é particularidade do Node-RED.

Da mesma forma que a anterior a API irá retornar um JSON contendo as variáveis pecasboas, pecasruins, pecastotais, timestamp e temperatura.

Ao digitar o código para submeter o fluxo ao servidor basta clicar no botão “Deploy”. Após o deploy abra o browser e digite o endereço “127.0.0.1:1880/mbi”



Será retornado o JSON idêntico ao da API anterior, com valores diferentes é claro, a cada chamada.

## CRIANDO O FRONT-END DA APLICAÇÃO

Nosso Front-End será construído usando as tecnologias web mais comuns, HTML, CSS e Javascript, crie um novo arquivo chamado index.html e digite o código a seguir.

```
<!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>MBI Automotiva 4.0</title>
</head>
<body>
  <div style="width:500px;display: flex;flex-direction: row;">
    <canvas id="graficoProducao" width="50" height="50"></canvas>
    <canvas id="graficoTemperatura" width="50" height="50"></canvas>
  </div>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <script src="mbi_javascript.js"></script>
</body>
</html>
```

Este código represente uma página web simples com duas áreas especiais para exibição de gráficos com os respectivos Ids graficoProducao e graficoTemperatura.

Agora vamos ao código Javascript que dará vida à nossa página. Crie um novo arquivo com nome mbi\_javascript.js e digite o código a seguir.

```
const ctx_graficoProducao = document.getElementById('graficoProducao');
const ctx_graficoTemperatura = document.getElementById('graficoTemperatura');

const graficoProducao=new Chart(ctx_graficoProducao, {
  type: 'bar',
  data: {
    labels: ['Peças Boas','Peças Ruins','Total Peças'],
    datasets: [{
      label: 'Produção',
      data: [0, 0, 0],
      backgroundColor: [
        'rgba(64, 64, 255, 0.2)',
        'rgba(255, 64, 64, 0.2)',
        'rgba(255, 180, 64, 0.2)'
      ],
      borderColor: [
```

```

        'rgb(0, 0, 255)',
        'rgb(255, 0, 0)',
        'rgb(255, 128, 0)'
    ],
    borderWidth: 1
  }]
}
});

const graficoTemperatura=new Chart(ctx_graficoTemperatura, {
  type: 'doughnut',
  data: {
    labels: [
      'Temperatura',
      'Range'
    ],
    datasets: [{
      label: 'Temperatura',
      data: [0,100],
      backgroundColor: [
        'rgb(255,64, 64)',
        'rgb(128, 128, 128)'
      ]
    }]
  }
});

const api=async()=>{
  const endpoint="http://127.0.0.1:1880/mbi"
  let res=await fetch(endpoint)
  .then(res=>res.json())
  .then(dados=>{
    graficoProducao.data.datasets[0].data[0]=dados.pecasboas
    graficoProducao.data.datasets[0].data[1]=dados.pecasruins
    graficoProducao.data.datasets[0].data[2]=dados.pecastotais
    graficoProducao.update()

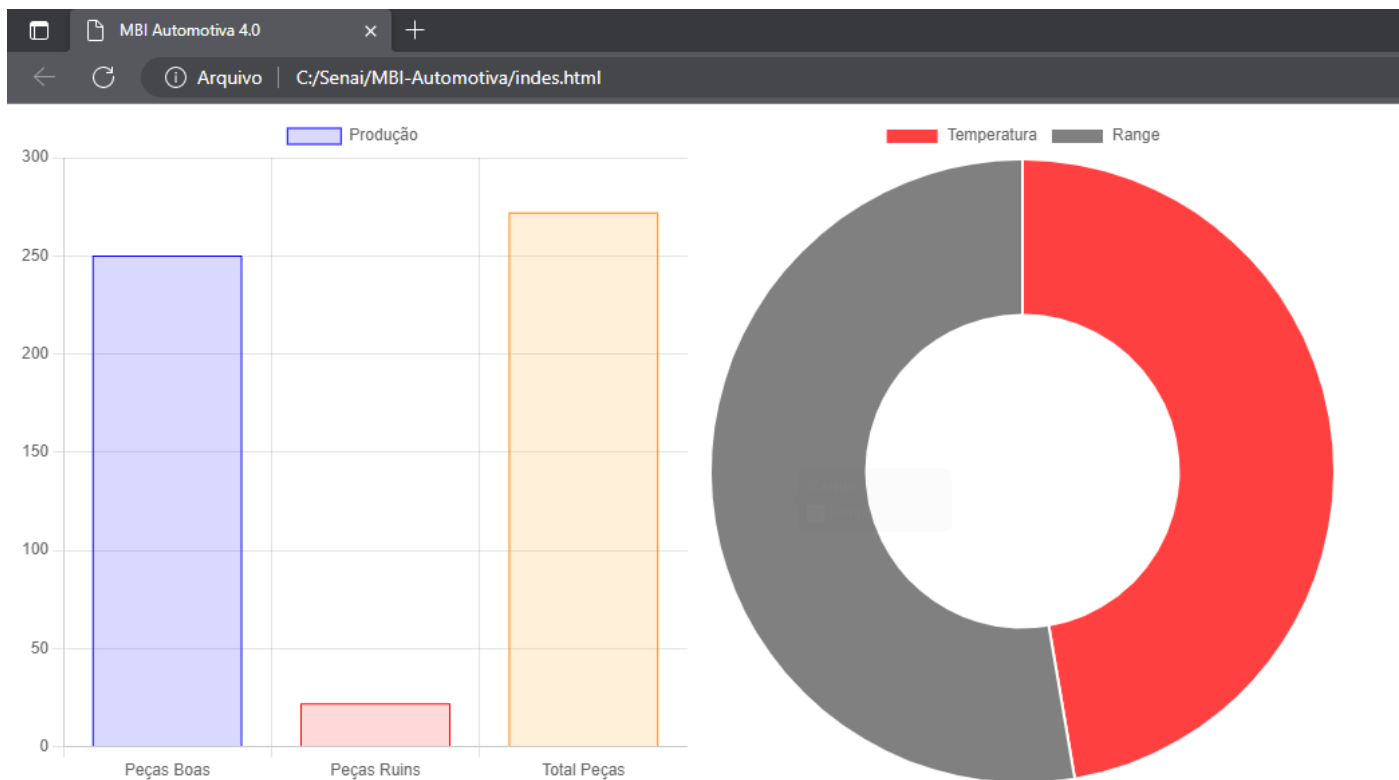
    const maxtemp=100
    graficoTemperatura.data.datasets[0].data[0]=dados.temperatura
    graficoTemperatura.data.datasets[0].data[1]=maxtemp-dados.temperatura
    graficoTemperatura.update()
  })
}

```

```
let intervalo=setInterval(api,5000)
```

Este código irá resultar em uma página semelhante a ilustração a seguir, onde a cada 5 segundos os dados são requisitados à API e atualizados nos gráficos.





Nesta aplicação estamos usando um biblioteca para geração dos gráficos chamada chart.js ( <https://www.chartjs.org/docs/latest/developers/api.html> ).

## EXPLICANDO O CÓDIGO

Nas primeiras linhas simplesmente associamos os elementos da página HTML “graficoProducao” e “graficoTemperatura” às devidas variáveis/constantes para que possamos trabalhar com estes elementos em nosso script.

```
const ctx_graficoProducao = document.getElementById('graficoProducao');  
const ctx_graficoTemperatura = document.getElementById('graficoTemperatura');
```

Em seguida configuramos o gráfico de barras conforme o modelo da biblioteca que estamos usando chart.js

```
const graficoProducao=new Chart(ctx_graficoProducao, {  
  type: 'bar',  
  data: {  
    labels: ['Peças Boas','Peças Ruins','Total Peças'],  
    datasets: [{  
      label: 'Produção',  
      data: [0, 0, 0],  
      backgroundColor: [  
        'rgba(64, 64, 255, 0.2)',  
        'rgba(255, 64, 64, 0.2)',  
        'rgba(255, 180, 64, 0.2)'  
      ],  
      borderColor: [  
        'rgb(0, 0, 255)',  
        'rgb(255, 0, 0)',  
        'rgb(255, 128, 0)'  
      ],  
      borderWidth: 1  
    }]  
  }  
});
```

Na sequência configuramos o gráfico do tipo “doughnut” conforme o modelo da biblioteca chart.js

```
const graficoTemperatura=new Chart(ctx_graficoTemperatura, {
  type: 'doughnut',
  data: {
    labels: [
      'Temperatura',
      'Range'
    ],
    datasets: [{
      label: 'Temperatura',
      data: [0,100],
      backgroundColor: [
        'rgb(255,64, 64)',
        'rgb(128, 128, 128)'
      ]
    }]
  }
});
```

Declaramos a função que iremos rodar para consumir o endpoint da API que criamos e atualizar os gráficos com os valores obtidos.

```
const api=async()=>{
```

Chamamos a função fetch que faz o consumo da API pelo endpoint indicado na linha 5.

```
const endpoint="http://127.0.0.1:1880/mbi"
let res=await fetch(endpoint)
```

Gerenciamos o consumo da API e a obtenção dos dados retornados pela API.

```
.then(res=>res.json())
.then(dados=>{
```

Inserimos os dados coletados pela API nas barras do primeiro gráfico.

```
graficoProducao.data.datasets[0].data[0]=dados.pecasboas
graficoProducao.data.datasets[0].data[1]=dados.pecasruins
graficoProducao.data.datasets[0].data[2]=dados.pecastotais
graficoProducao.update()
```

Definimos o valor máximo para temperatura.

```
const maxtemp=100
```

Inserimos os dados coletados pela API nas barras do segundo gráfico gráfico.

```
graficoTemperatura.data.datasets[0].data[0]=dados.temperatura
graficoTemperatura.data.datasets[0].data[1]=maxtemp-dados.temperatura
graficoTemperatura.update()
```

Por fim criamos o intervalo que será responsável pela chamada da função api, a cada 5 segundos a função é chamada, então obtemos os dados da API criada e atualizamos os gráficos de acordo com os dados obtidos.

```
let intervalo=setInterval(api,5000)
```

## CONSTRUINDO A APLICAÇÃO EM REALIDADE AUMENTADA

O QUE É REALIDADE AUMENTADA?

Realidade Aumentada ou RA é a integração de elementos ou informações virtuais a visualização do mundo real através de uma câmera.

Em síntese é a capacidade computacional de inserir elementos virtuais ao contexto do mundo real visto por uma câmera.

Também são usados outros sensores além da câmera, como giroscópio e acelerômetro. De forma que o software possa calcular as distorções para exibição do conteúdo digital de acordo com o mundo real.

A realidade aumentada pode ser utilizada em design de produtos, ações de marketing, treinamento e suporte em plantas industriais, entre outros.

Os primeiros sistemas de realidade aumentada usando experiências imersivas foram inventadas no começo dos anos 90.

Não confunda Realidade Aumentada com Realidade Virtual. A Realidade Aumentada usa o mundo real e insere informações digitais neste mundo. Já a Realidade Virtual substitui completamente o mundo real de quem está fazendo uso da tecnologia.

## *FERRAMENTA DE DESENVOLVIMENTO*

Hoje em dia existem diversas ferramentas de desenvolvimento de RA, desde ferramentas gratuitas a ferramentas pagas.

Google Arcore (Android / iOS): Plataforma do Google para construir aplicações em RA

Arkit2 (iOS): Plataforma para construção de RA em dispositivos Apple

Vuforia: Framework para Unity e Android Studio que permite a criação de aplicações de RA nestas plataformas, usando seus recursos.

## *COMO FUNCIONA A APLICAÇÃO DE REALIDADE AUMENTADA?*

O funcionamento é bem simples. A aplicação scaneia o ambiente o tempo todo em busca dos alvos que ela está programada para reconhecer.

Ao encontrar um alvo a aplicação pode rodar funções previamente programadas.

Uma das ações mais comuns é carregar um modelo 3D ou carregar um dashboard com informações diversas, todo conteúdo carregado irá ser mostrado no contexto do mundo real.

## *CRIANDO A PRIMEIRA APLICAÇÃO EM RA*

Vamos criar uma aplicação simples usando uma ferramenta gratuita na Internet chamada MindAR.

O MindAR é uma biblioteca gratuita opensource para criação de aplicações em RA.

<https://hiukim.github.io/mind-ar-js-doc/>

MindAR usa recursos de outra ferramenta gratuita chamada AFRAME, focada em Realidade Virtual.

<https://aframe.io/>

MindAR pode rodar diretamente em uma página HTML estática. É bem simples.

Para usar o recurso, basta anexar os scripts mostrados a seguir via CDN (Content Delivery Network).

```
<script src="https://cdn.jsdelivr.net/npm/mind-ar@1.1.5/dist/mindar-image.prod.js"></script>
<script src="https://aframe.io/releases/1.2.0/aframe.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/mind-ar@1.1.5/dist/mindar-image-aframe.prod.js"></script>
```

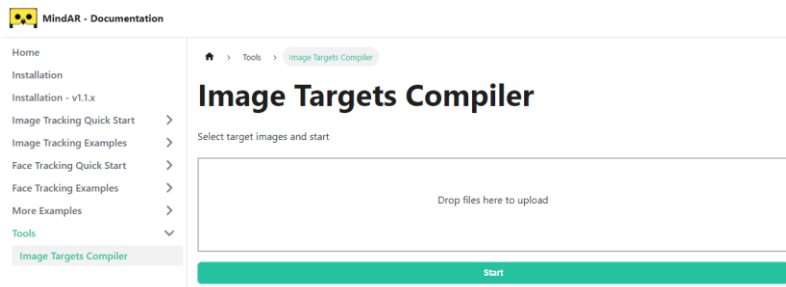
Veja o código HTML básico completo.

```
<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <script src="https://cdn.jsdelivr.net/gh/hiukim/mind-ar-js@1.1.5/dist/mindar-image.prod.js"></script>
    <script src="https://aframe.io/releases/1.2.0/aframe.min.js"></script>
    <script src="https://cdn.jsdelivr.net/gh/hiukim/mind-ar-js@1.1.5/dist/mindar-image-aframe.prod.js"></script>
  </head>
  <body>
    <a-scene mindar-image="imageTargetSrc: SEU_ARQUIVO_ALVO.mind;" color-space="sRGB" renderer="colorManagement: true,
    physicallyCorrectLights" vr-mode-ui="enabled: false" device-orientation-permission-ui="enabled: false">
      <a-camera position="0 0 0" look-controls="enabled: false"></a-camera>
      <a-entity mindar-image-target="targetIndex: 0">
        <a-plane color="blue" opacity="0.5" position="0 0 0" height="0.552" width="1" rotation="0 0 0"></a-plane>
      </a-entity>
    </a-scene>
  </body>
</html>
```

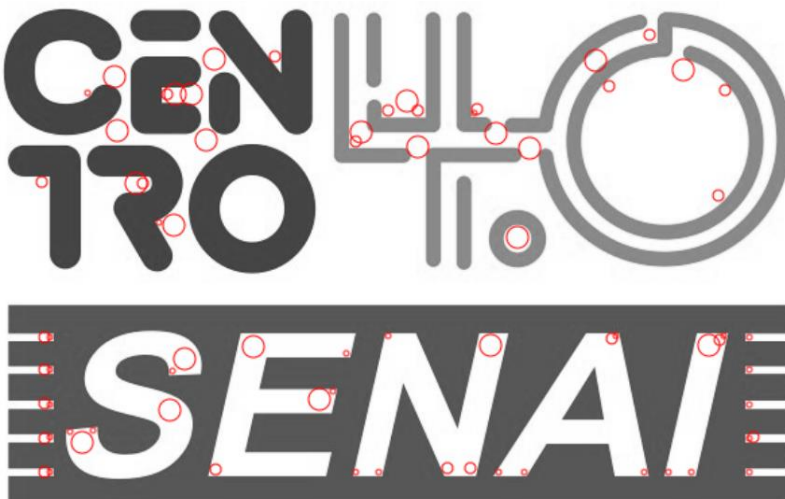
## CRIANDO ALVOS USANDO MINDAR

Para criar o alvo, podemos usar diretamente o site do MindAR na seguinte URL ( <https://hiukim.github.io/mind-ar-js-doc/tools/compile> )

Arraste a imagem para a área indicada (Drop files here to upload) e na sequência clique em “Start”.



A imagem será processada e ao final gerado um alvo onde podemos alterar a escala conforme mostra a imagem a seguir em escala 3.



Após selecionar a escala desejada, no caso usamos a 3, clique no botão “Download compile”. Baixe o arquivo “targets.mind”, você pode mudar o nome do arquivo caso queira.

## DISPONIBILIZANDO SEU ALVO EM CDN

CDN (Content Delivery Network) em resumo é uma rede de fornecimento, entrega e distribuição de conteúdo e podemos usar este recurso para fornecer o alvo que criamos à nossa aplicação de RA.

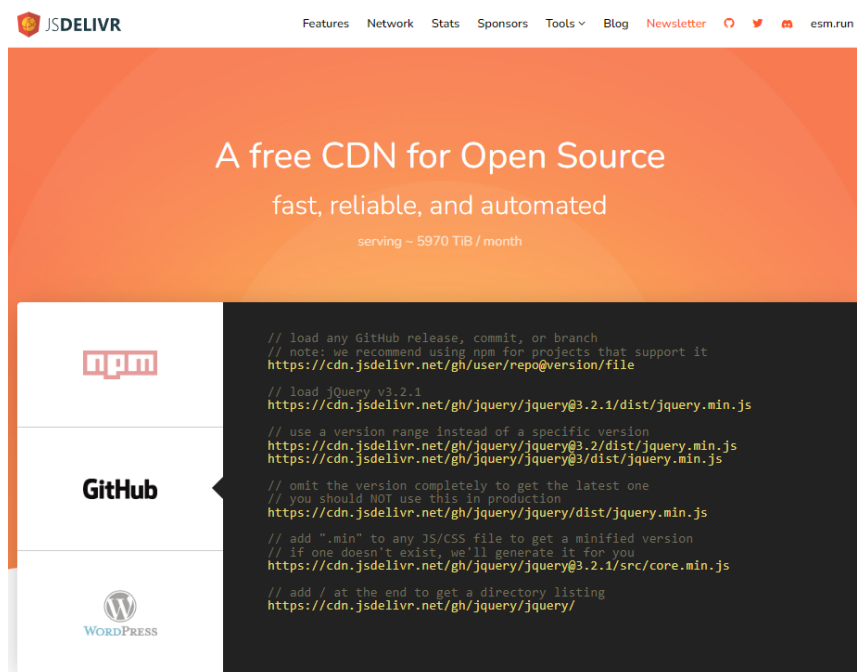
Para que possamos disponibilizar conteúdo via CDN vamos usar um dos serviços de CDN gratuitos que é o jsDelivr ( <https://www.jsdelivr.com> ) em conjunto com o GitHub ( <https://github.com/> ).

O processo é bem simples e você precisa de uma conta no GitHub, portanto se não tem providencie uma.

Suba o arquivo do alvo que criamos para um repositório “público” a sua escolha em sua conta no GitHub, após “comitar” o arquivo clique nele dentro do repositório para ver o link do arquivo, em nosso exemplo o link está mostrado a seguir.

<https://github.com/brunosenai/ra/blob/main/targets.mind>

Agora usando o jsDelivr clique na seção do GitHub para ver os exemplos de links.



Iremos usar o primeiro modelo do GitHub que é este “<https://cdn.jsdelivr.net/gh/user/repo@version/file>”

Sendo assim, basta substituir as partes indicadas no link do jsDelivr pelo link do GitHub

<https://cdn.jsdelivr.net/gh/user/repo@version/file>

<https://github.com/brunosenai/ra/blob/main/targets.mind>

Então o link final ficará assim: <https://cdn.jsdelivr.net/gh/brunosenai/ra/targets.mind>

Este é o link que usaremos em mindar-image imageTargetSrc no código básico HTML, então vamos ao código

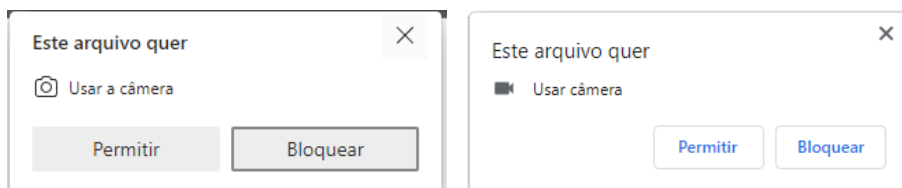
```
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <script src="https://cdn.jsdelivr.net/gh/hiukim/mind-ar-js@1.1.5/dist/mindar-image.prod.js"></script>
  <script src="https://aframe.io/releases/1.2.0/aframe.min.js"></script>
```

```

<script src="https://cdn.jsdelivr.net/gh/hiukim/mind-ar-js@1.1.5/dist/mindar-image-
aframe.prod.js"></script>
</head>
<body>
  <a-scene mindar-image="imageTargetSrc: https://cdn.jsdelivr.net/gh/brunosenai/ra/targets.mind;"
color-space="sRGB" renderer="colorManagement: true, physicallyCorrectLights" vr-mode-ui="enabled: false"
device-orientation-permission-ui="enabled: false">
    <a-camera position="0 0 0" look-controls="enabled: false"></a-camera>
    <a-entity mindar-image-target="targetIndex: 0">
      <a-plane color="blue" opacity="0.5" position="0 0 0" height="0.552" width="1" rotation="0 0
0"></a-plane>
    </a-entity>
  </a-scene>
</body>
</html>

```

Ao roda este HTML no browser ele pedirá permissão para usar a webcam e claro que precisamos clicar em “Permitir”



Aponte o nosso alvo para a webcam e veja que será mostrado um quadrado azul exatamente sobre o alvo.



Este quadrado é exatamente o plano que criamos em nosso código, mostrado a seguir.

```

<a-plane color="blue" opacity="0.5" position="0 0 0" height="0.552" width="1" rotation="0 0 0"></a-plane>

```

## CRIANDO DASHBOARD SIMPLES DE PEÇAS BOAS E RUINS

Vamos criar um Dashboard simples com duas barras para o gráfico de produção, as barras irão representar peças boas e peças ruins.

Crie um novo arquivo chamado index\_ra.html e digite o código a seguir o código.

```

<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <script src="https://cdn.jsdelivr.net/gh/hiukim/mind-ar-js@1.1.5/dist/mindar-
image.prod.js"></script>

```

```

<script src="https://aframe.io/releases/1.2.0/aframe.min.js"></script>
<script src="https://cdn.jsdelivr.net/gh/hiukim/mind-ar-js@1.1.5/dist/mindar-image-
aframe.prod.js"></script>
</head>
<body>

  <a-scene mindar-image="imageTargetSrc: https://cdn.jsdelivr.net/gh/brunosenai/ra/targets.mind;">
    <a-camera look-controls></a-camera>
    <a-entity mindar-image-target="targetIndex: 0" position="0 0 -1">
      <a-plane color="#888" opacity="0.5" position="0 0 0" height="0.5" width="1" rotation="0 0
0">
        <a-text value="SENAI - CENTRO 4.0" position="-0.2 0.2 0" scale="0.2 0.2 0.2"></a-
text>

        <a-box color="rgba(64, 64, 255, 0.2)" depth="0.005" height="0" width="0.05"
position="-0.3 0.2 0" id="id_barra_pecasboas"></a-box>
        <a-text value="PecasBoas" position="-0.20 -0.22 0" scale="0.1 0.1 0.1"></a-text>
        <a-text value="000" position="-0.20 -0.25 " scale="0.1 0.1 0.1"
id="qtde_pecasboas"></a-text>

        <a-box color="rgba(255, 64, 64, 0.2)" depth="0.005" height="0" width="0.05"
position="0 0.2 0" id="id_barra_pecasruins"></a-box>
        <a-text value="PecasRuins" position="0.1 -0.22 0" scale="0.1 0.1 0.1"></a-text>
        <a-text value="000" position="0.1 -0.25 0" scale="0.1 0.1 0.1"
id="qtde_pecasruins"></a-text>

        <a-box color="#aaa" depth="0.005" height="0.01" width="0.8" position="0 -0.2 0.01"
id="linhaBasse"></a-box>
      </a-plane>
    </a-entity>
  </a-scene>

  <script>
    const escalaMax=0.35
    const barra_pecasboas=document.getElementById("id_barra_pecasboas")
    const barra_pecasruins=document.getElementById("id_barra_pecasruins")
    const qtde_pecasboas=document.getElementById("qtde_pecasboas")
    const qtde_pecasruins=document.getElementById("qtde_pecasruins")

    let posYbarra_pecasboas=posYbarra_pecasruins=0
    let valBarra_pecasboas=valBarra_pecasruins=0
    let valPosBarra_pecasboas=valPosBarra_pecasruins="0 0 0"
    let posBase=-0.2
    let pecasboas=0
    let pecasruins=0
    let pecastotais=0

    const controle=()=>{
      const endpoint="http://127.0.0.1:1880/mbi"
      fetch(endpoint)
        .then(request => request.json())
        .then(dados=>{
          pecasboas=(dados.pecasboas)
          pecasruins=(dados.pecasruins)
          pecastotais=pecasboas+pecasruins

          valBarra_pecasboas=(pecasboas-0)*(escalaMax-0)/(pecastotais-0)+0
          valBarra_pecasruins=(pecasruins-0)*(escalaMax-0)/(pecastotais-0)+0

          posYbarra_pecasboas=(valBarra_pecasboas/2)+posBase
          valPosBarra_pecasboas="-0.15 "+posYbarra_pecasboas+" 0.005"

          posYbarra_pecasruins=(valBarra_pecasruins/2)+posBase
          valPosBarra_pecasruins="0.15 "+posYbarra_pecasruins+" 0.005"

          barra_pecasboas.setAttribute("height",valBarra_pecasboas)
          barra_pecasboas.setAttribute("position",valPosBarra_pecasboas)
          qtde_pecasboas.setAttribute("value",pecasboas)

          barra_pecasruins.setAttribute("height",valBarra_pecasruins)
          barra_pecasruins.setAttribute("position",valPosBarra_pecasruins)
          qtde_pecasruins.setAttribute("value",pecasruins)
        })
    }
  </script>

```

```

    }
    const int=setInterval(controle,3000)
  </script>
</body>
</html>

```

Como explicado anteriormente este código cria um processo de Realidade Aumentada pelo MindAR que usa a base do AFrame, sendo assim qualquer inclusão de elementos visuais em nossa dashboard seguirá as regras do AFrame.

## EXPLICANDO O CÓDIGO

Inicialmente temos as definições iniciais da página com as indicações nas linhas 4, 5 e 6 dos frameworks que serão utilizados.

```

<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <script src="https://cdn.jsdelivr.net/gh/hiukim/mind-ar-js@1.1.5/dist/mindar-
image.prod.js"></script>
    <script src="https://aframe.io/releases/1.2.0/aframe.min.js"></script>
    <script src="https://cdn.jsdelivr.net/gh/hiukim/mind-ar-js@1.1.5/dist/mindar-image-
aframe.prod.js"></script>
  </head>
  <body>

```

Na sequência encontramos toda codificação dos elementos visíveis na RA, como dito anteriormente usamos o framework AFrame como base para o MindAR.

```

<a-scene mindar-image="imageTargetSrc: https://cdn.jsdelivr.net/gh/brunosenai/ra/targets.mind;">
  <a-camera look-controls></a-camera>
  <a-entity mindar-image-target="targetIndex: 0" position="0 0 -1">
    <a-plane color="#888" opacity="0.5" position="0 0 0" height="0.5" width="1" rotation="0 0 0">
      <a-text value="SENAI - CENTRO 4.0" position="-0.2 0.2 0" scale="0.2 0.2 0.2"></a-text>

      <a-box color="rgba(64, 64, 255, 0.2)" depth="0.005" height="0" width="0.05" position="-0.3
0.2 0" id="id_barra_pecasboas"></a-box>
      <a-text value="PecasBoas" position="-0.20 -0.22 0" scale="0.1 0.1 0.1"></a-text>
      <a-text value="000" position="-0.20 -0.25 " scale="0.1 0.1 0.1" id="qtde_pecasboas"></a-text>

      <a-box color="rgba(255, 64, 64, 0.2)" depth="0.005" height="0" width="0.05" position="0 0.2
0" id="id_barra_pecasruins"></a-box>
      <a-text value="PecasRuins" position="0.1 -0.22 0" scale="0.1 0.1 0.1"></a-text>
      <a-text value="000" position="0.1 -0.25 0" scale="0.1 0.1 0.1" id="qtde_pecasruins"></a-text>

      <a-box color="#aaa" depth="0.005" height="0.01" width="0.8" position="0 -0.2 0.01"
id="linhaBasse"></a-box>
    </a-plane>
  </a-entity>
</a-scene>

```

Definimos a escala máxima para 0.35, este valor está de acordo para o tamanho da área que iremos usar e apresentar em nossa aplicação, de acordo com o conteúdo desenvolvido este valor poderá ser maior ou menor

```
const escalaMax=0.35
```

Em seguida associamos em nosso script os elementos HTML que precisaremos manipular.

```

const barra_pecasboas=document.getElementById("id_barra_pecasboas")
const barra_pecasruins=document.getElementById("id_barra_pecasruins")
const qtde_pecasboas=document.getElementById("qtde_pecasboas")
const qtde_pecasruins=document.getElementById("qtde_pecasruins")

```



Definimos as variáveis que iremos usar em nosso projeto, lembrando de cada projeto precisará de suas próprias variáveis.

Estas variáveis receberão o cálculo para reposicionamento das barras, no AFrame o componente “a-box” que estamos usando para desenhar as barras é alinhado pelo centro e não pela base, em nosso gráfico as barras aumentam ou diminuem pela base, assim faremos um cálculo bem simples que reposicionaremos as barras no local que queremos a cada vez que for alterado seu tamanho.

```
let posYbarra_pecasboas=posYbarra_pecasruins=0
```

Estas variáveis servirão para receber o valor de altura das barras já calculado de acordo com o valor recebido pela API e a escala da nossa aplicação.

```
let valBarra_pecasboas=valBarra_pecasruins=0
```

Estas variáveis irão receber o valor final calculado da posição da barra para ser usado no posicionamento da barra na aplicação.

```
let valPosBarra_pecasboas=valPosBarra_pecasruins="0 0 0"
```

Variável com o valor da posição da base para as barras dos gráficos.

```
let posBase=-0.2
```

Estas variáveis receberão os valores diretos retornados pela API

```
let pecasboas=0  
let pecasruins=0  
let pecastotais=0
```

Definição da função que irá chamar a API, dentro da função é indicado o endpoint da API, usamos a função fetch para chamar a API e gerenciamos a aquisição dos dados.

```
const controle=()=>{  
  const endpoint="http://127.0.0.1:1880/mbi"  
  fetch(endpoint)  
    .then(request => request.json())  
    .then(dados=>{
```

Recebemos os dados da API e atribuímos às respectivas variáveis.

```
pecasboas=(dados.pecasboas)  
pecasruins=(dados.pecasruins)  
pecastotais=pecasboas+pecasruins
```

Cálculo de posicionamento e valor das barras que serão apresentadas na aplicação

```
valBarra_pecasboas=(pecasboas-0)*(escalaMax-0)/(pecastotais-0)+0  
valBarra_pecasruins=(pecasruins-0)*(escalaMax-0)/(pecastotais-0)+0  
  
posYbarra_pecasboas=(valBarra_pecasboas/2)+posBase  
valPosBarra_pecasboas="-0.15 "+posYbarra_pecasboas+" 0.005"  
  
posYbarra_pecasruins=(valBarra_pecasruins/2)+posBase  
valPosBarra_pecasruins="0.15 "+posYbarra_pecasruins+" 0.005"
```

Note que nos cálculos acima usamos um mapeamento de valores, pois precisamos transformar os valores recebidos pela API em valores para as barras dos gráficos.

Usamos a seguinte regra:

$$\frac{(\text{valor\_recebido\_api} - \text{valor\_minimo\_que\_podera\_ser\_recebido}) * (\text{valor\_maximo\_escala} - \text{valor\_minimo\_esacala})}{(\text{valor\_maximo\_que\_podera\_ser\_recebido} - \text{valor\_minimo\_que\_podera\_ser\_recebido}) + \text{valor\_minimo\_esacala}}$$

O que resultou nesta construção:

$$(\text{pecasboas}-0)*(\text{escalaMax}-0)/(\text{pecastotais}-0)+0$$

Assim relacionamos o valor máximo sendo o de pecastotais com o valor máximo da escala que definimos que é 0.35.

Em seguida atualizamos as propriedades das barras com os valores calculador.

```
barra_pecasboas.setAttribute("height",valBarra_pecasboas)
barra_pecasboas.setAttribute("position",valPosBarra_pecasboas)
qtde_pecasboas.setAttribute("value",pecasboas)

barra_pecasruins.setAttribute("height",valBarra_pecasruins)
barra_pecasruins.setAttribute("position",valPosBarra_pecasruins)
qtde_pecasruins.setAttribute("value",pecasruins)
```

Para finalizar chamamos a função de controle das barras a cada 3 segundos.

```
const int=setInterval(controle,3000)
```

Rodando o arquivo HTML o resultado a seguir será mostrado quando apontarmos o algo para a webcam. Obviamente com os valões de peças boas e peças ruins diferentes dos valores mostrados a seguir.



## DISPONIBILIZANDO A APLICAÇÃO RA ONLINE


Caso você queira disponibilizar a aplicação online, você poderá hospedar em um servidor web de sua escolha, ou simplesmente usar um sistema online como o Replit ( <https://replit.com/> ) ou o Glitch ( <https://glitch.com/> ), em ambos é necessário fazer um cadastro.

### REPLIT

No Replit basta criar um novo projeto Repl usando o template “HTML, CSS, JS” como título que você desejar, em nosso caso usei “RA-MBI”

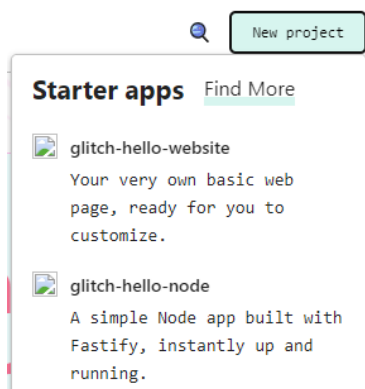
The screenshot shows the 'Create a Repl' interface. At the top left is the title 'Create a Repl'. To its right is a button 'Import from GitHub' with a close icon. Below the title, there are two main sections. The left section is for selecting a template, with a dropdown menu currently showing 'HTML, CSS, JS'. Below this, the 'HTML, CSS, JS' template is highlighted, showing its icon, a 'Languages' tag, a description 'The languages that make up the web. HTML provides the basic structure, CSS controls formatting, and JavaScript...', the Replit logo, and statistics '1.5K + 7.1M'. The right section is for configuring the Repl, with fields for 'Title' (containing 'RA-MBI'), 'Owner' (containing 'brcampos'), and 'Public' status (selected). Below these is a button 'Power Up to make private' and a large blue button '+ Create Repl'.

Assim que criado apague o código básico e cole o que construímos no Visual Studio Code.

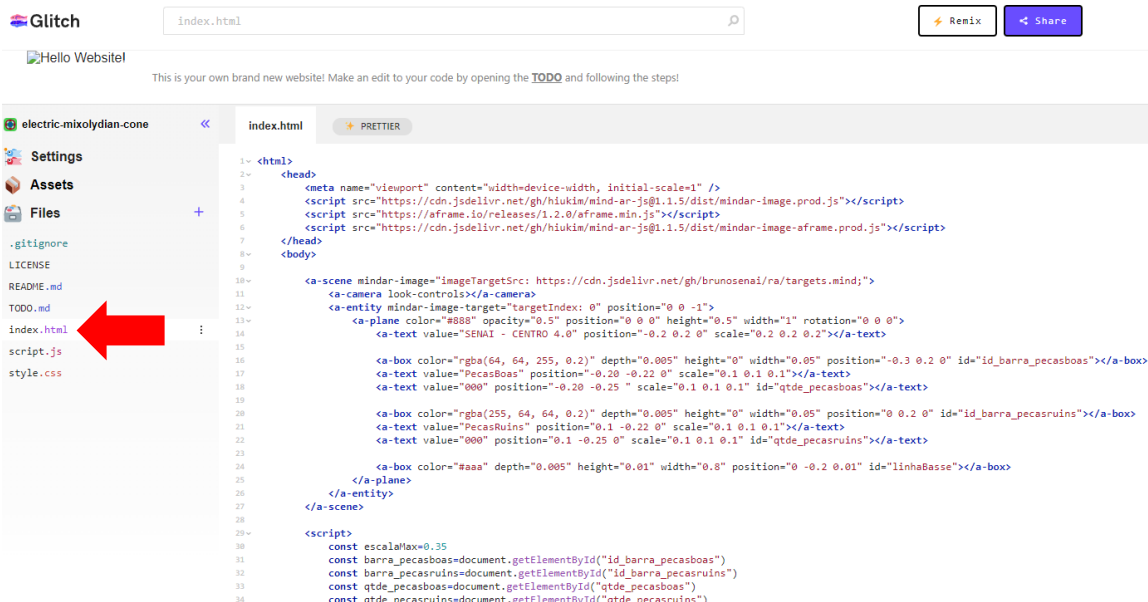
Após colado basta clicar em RUN, na janela “Webview” será rodado o projeto, para abrir o projeto em uma nova janela e assim podermos compartilhar com que for necessário basta clicar no botão “Open in a new tab” 

## GLITCH

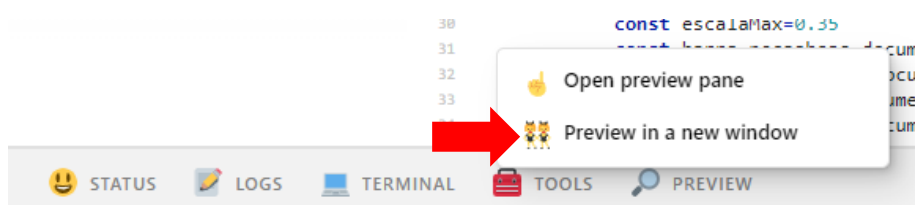
No Glith (<https://glitch.com/>) também é necessário criar uma conta e logar. Após o log clique no botão “New Project” e selecione a opção “glitch-hello-website”.



No painel da esquerda selecione o arquivo “index.html”, apague o código e cole o que criamos no Visual Studio Code.



Para ver o resultado use o botão “PREVIEW” na barra inferior selecionando “Preview in a new window”



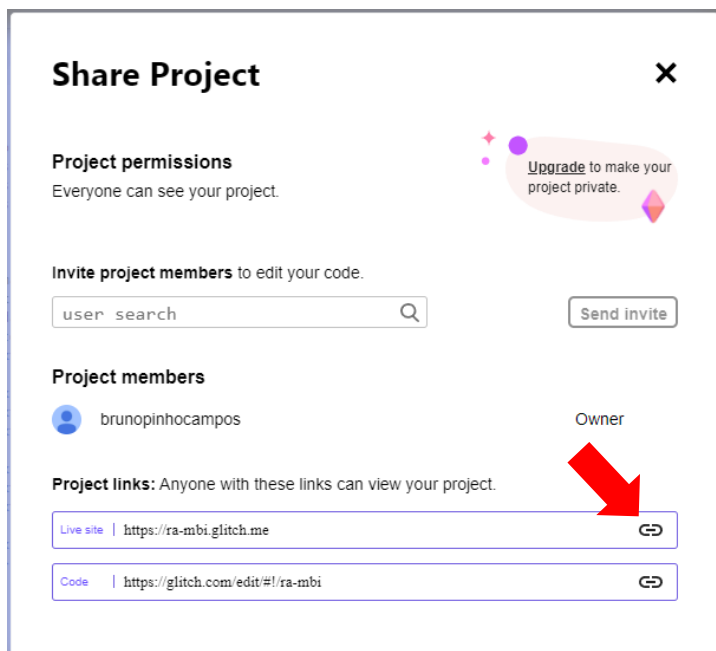
A página será aberta em uma nova janela.

Para mudar o nome do projeto no Glitch, clique em “Settings – Edit Project details”. No campo “PROJECT NAME” digite o nome do projeto e clique no botão “Save”.

Em nosso caso usamos o nome “RA-MBI”

Para compartilhar o código, usamos o botão “Share” na parte superior.

Ao clicar no botão “Share” a janela de compartilhamento será mostrada então basta copiar o link disponibilizado por parte inferior, Live site.



Cole em uma nova janela ou aba do browser ou envie para alguém que você queira compartilhar a aplicação, lembre-se que para ver a aplicação a pessoa precisará do alvo.

E assim chegamos ao fim do nosso tutorial e agora você já sabe um pouco sobre como podemos gerar e obter dados de APIs e como podemos programar sistemas para visualização e gerenciamento destes dados.

Esperamos que o conteúdo tenha sido proveitoso, desejamos sucesso em sua caminhada e até a próxima.