**Proyecto 2 – Creación de Aplicación web con HTML, CSS, JS**

**I - Introducción**

Anteriormente habíamos visto la aplicación de detección de caras con OpenCV, la cual es la librería de visión por computador hoy por hoy por excelencia debido al gran soporte y la adopción por muchas compañías debido a la implementación de muchos algoritmos.

Si bien OpenCV es dedicada para el procesamiento de imágenes, últimamente se han añadido librerías y funciones que ayudan a la parte de DevOps (Development and Operations) para la parte de administración de proyectos de Vision además de la integración con paquetes que importan redes neuronales para hacer aplicaciones más ricas.

Ahora, imaginese que tiene todo lo anterior realizado con OpenCV, su aplicación es excelente pero lo desea poner en producción. ¡No puedes!, ¿por qué?, su aplicación es de escritorio y solo uds puede y sabe utilizarla, nadie más.

Es por esto la necesidad de crear interfaces gráficas que sean de utilidad para que el usuario final pueda integrarla fácilmente.

No solo sería para que el usuario pueda compartirla, sino también para recavar información si se puede, es decir, esta información puede servir para generar valor, saber que desea el usuario y hacer marketing estratégico. Como dice el dicho: ‘Si ud. no paga por lo que consume, ud. es el producto’.

No realizaremos integración esta vez a base de datos, pero es una posibilidad como se comentó en el párrafo anterior de recopilar datos del cliente o comportamiento, nos centraremos en realizar una aplicación web integrada con OpenCV.

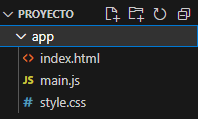
**II - ¿Qué es HTML, JS, CSS?**

Historia larga corta, HTML es un “lenguaje” que se utiliza para la creación de páginas web, CSS obliga a la pagina tener estilos, JS es el rey de la web y es la librería pro defecto que domina ahora mismo, si desea tener un trabajo como desarrollador web es indispensable dominar estas herramientas, sin embargo hay frameworks como ReactJS o Angular que simplifican un poco más el trabajo, pero igualmente se basan en aplicaciones web.

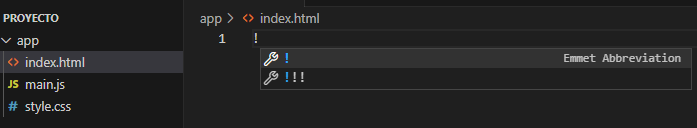
**III – Desarrollo del Proyecto**

**III.1 – Creación de Archivos y Página Inicial**

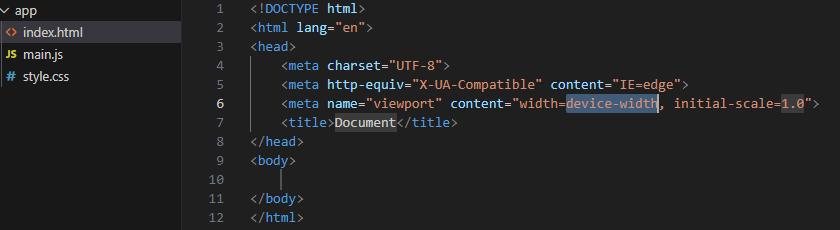
* Lo primero es crear la siguiente estructura de archivos



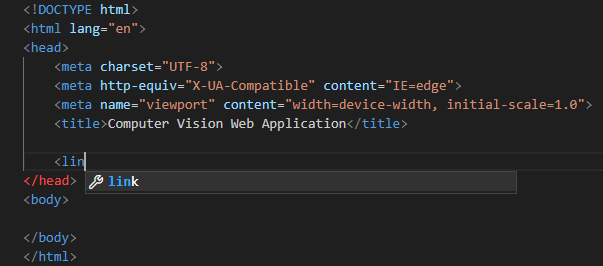
* Si está utilizando Visual Studio Code, es fácil empezar con HTML utilizando Intellisense, seleccione el símbolo de exclamación y se verá esta opción.



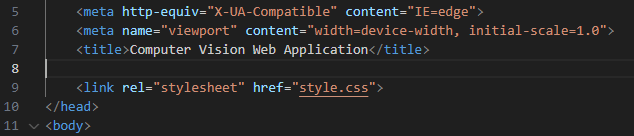
* Seguido al presionar ENTER, terminará con la siguiente parte de archivos.



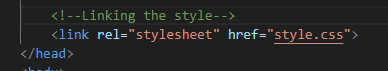
* Cambie el HTML Tag <title> por el nombre de la aplicación que va a cargar.
* Seguido cree el link



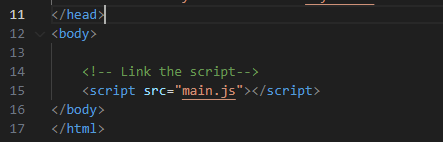
* Ingrese la referencia al archivo style.css



* Para los comentarios ingrese en la linea superior un texto descriptivo en este caso



* Repita los mismos pasos para agregar main.js pero agregando un script

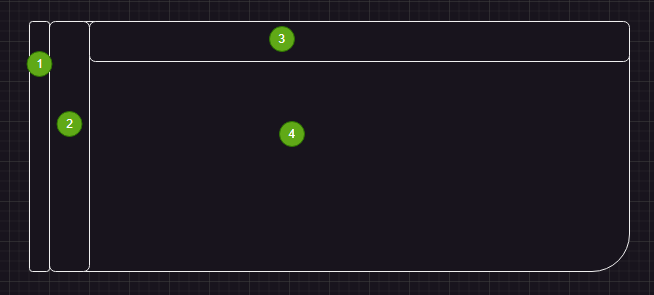


* Si salvamos la página y la desplegamos en un navegador observaremos que el titulo de la página corresponde al título del app que estamos creando.



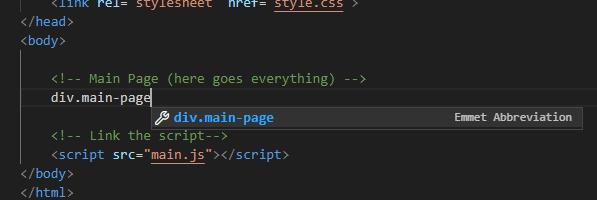
**III.2 – Creación de la estructura de la aplicación**

Como mencionamos anteriormente queremos crear la siguiente aplicación:



1. Barra de Estilo: Solo decorativa
2. Barra de Menu y Ajustes de Usuario: Tiene los controles
3. Encabezado: Tiene solo texto descriptivo de la aplicación
4. Visualizador: Tiene las acciones ejecutadas por la barra de menú y ajustes

* Ahora seguiremos creando la aplicación. Creamos una división que contendrá toda la página.
* Ingrese el comentario HTML para solo saber que incluye esta sección
* Escriba div.main-page y presione ENTER para completar



* El resultado es la creación del div para la página
* Expanda el div hasta el final para poder seguir agregando HTML



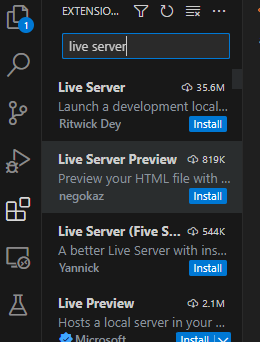
* Agregue los divs para las partes de la página que vamos a completar
* Etiquetelos con comentarios para luego buscar fácilmente



* Dentro del visualizador cree más divs para ubicar un header de titulo y otro de el mismo resultado de la aplicación



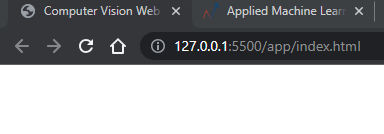
* Probemos instalando la extensión de Visual Studio Code Llamada Live Server a ver que sucede



* Luego de instalarla presionamos click derecho en el index.html y finalmente Open in Live Server

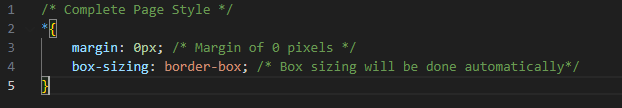


* Vemos que no existe aún contenido y esto es correcto.

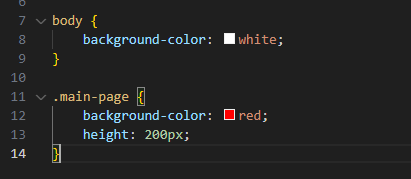


**III.3 – Creación de Estilos**

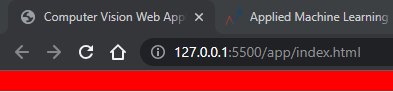
* Abrir el archivo style.css que son los estilos o formas que tomará en la página web e insertar el estilo que se le aplicará a TODA LA PAGINA con el SIMBOLO \* como se muestra en la figura inferior.



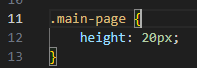
* Ingrese ahora la descripción del cuerpo de la página, cuando no tienen asterisco se aplican a secciones específicas.



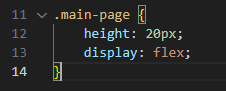
* Si salva y regresa a la página verá que ha cambiado el estilo, ahora se muesra roja y blanca



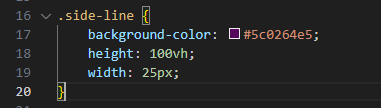
* Cambie los 200px a 100vh, que quiere decir que veremos al 100% toda la vista del color deseado y removeremos el color de fondo, solo era para entender que está pasando

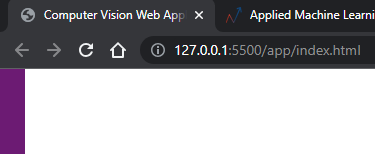


* Display flex permite alterar los componetes internos de la página, es lo único que necesita entender por el momento.

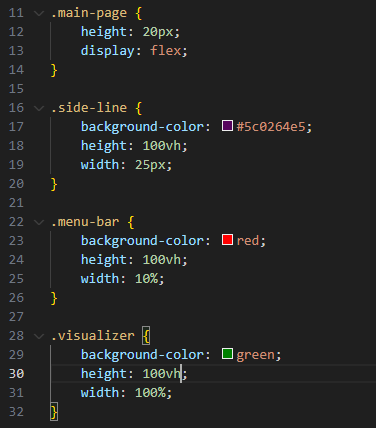


* Ahora crearemos los estilos adicionales para la caja sideline, use el color que guste

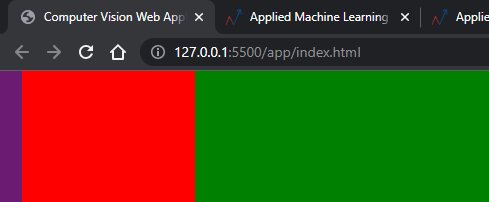




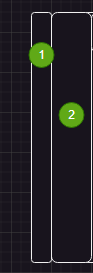
* Haga los siguientes cambios para los otros esitlos de la página



* Pruebe la pagina y modifique su alto y ancho para que observe las acciones. Los colores rojo y verde son solo para entender que está sucediendo.

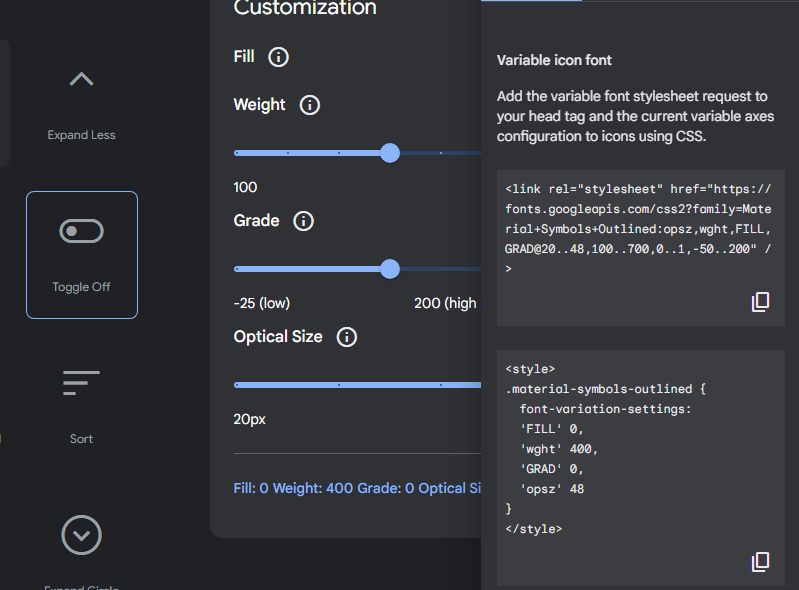


**III.3 – Ingreso de íconos y creación de estilos**



Para utilizar íconos por defecto podemos ir a: <https://fonts.google.com/icons>

Estos los podemos utilizar por ejemplo copiando el link web y estilo

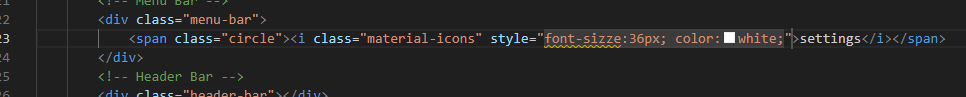


Más información de su uso en :

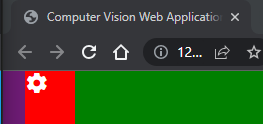
* <https://developers.google.com/fonts/docs/getting_started>
* <https://developers.google.com/fonts/docs/material_icons>
* Agregamos entonces los iconos como se muestra



* En la sección de la barra de menú agregamos los íconos
* Debe terminar dentro del div de menú como sigue



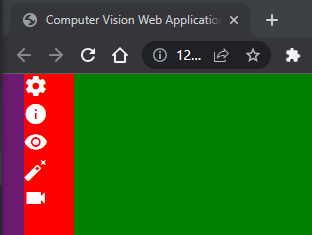
* Revisando la página web tenemos:



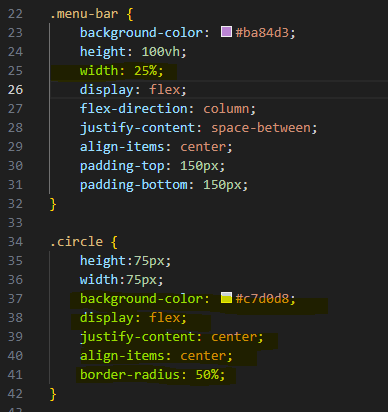
* Repetimos la misma operación para alrededor de 5 iconos adicionales de materials de google como sigue

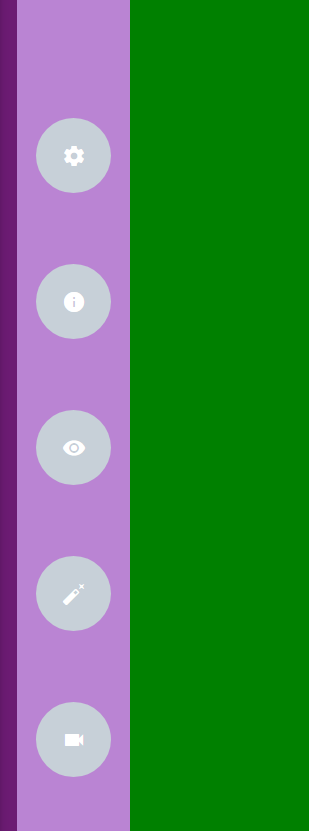


* Revisando la página tenemos

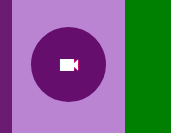


* Ahora vamos a formatear el menú bar, favor tomese el tiempo de saber que hace cada uno de estas líneas, puede borrar una por una e ir viendo los cambios a medida que actualiza la página.





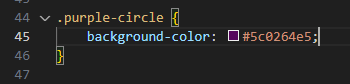
* Para cambiar el último ícono a otro tono como se muestra en la inferior



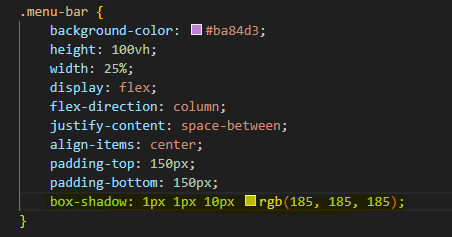
* Modificar style.css primero modificando la página web como una nueva clase purple-circle



* Finalmente modifique el css



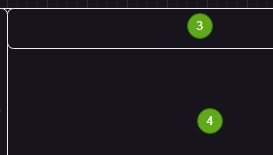
* NOTA: Para ver y entender interactivamente puede realizarlo observando la siguiente página web: <https://developer.mozilla.org/en-US/docs/Web/CSS/box-shadow>
* Finalmente para crear una sombra en su página para mejorar el estilo ud puede ubicar



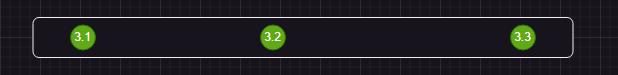
* Y al visualizar observará que existe una sombra que se nota en ambas barras



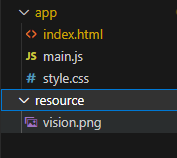
**III.4 – Creación de paneles de visualización**



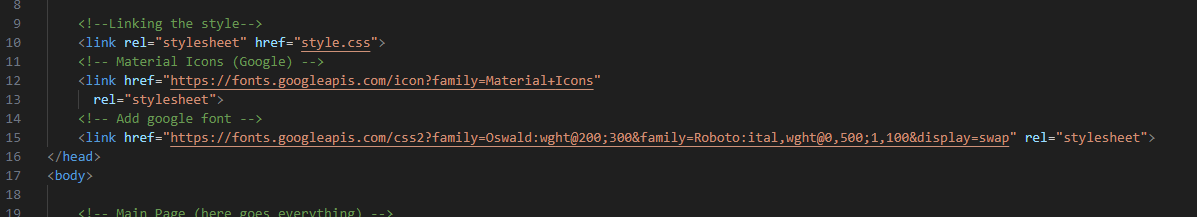
* Para esta sección crearemos la parte de encabezado de la aplicación que la tendremos por medio de 3 separadores, uno vación y dos con información

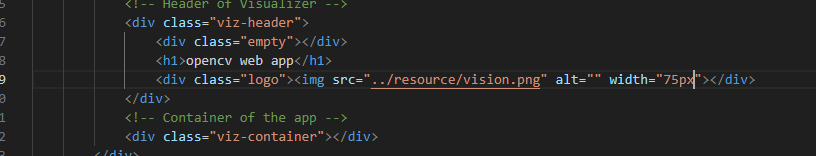


* Cree una carpeta de recursos adincionales y descargue la figura que desea

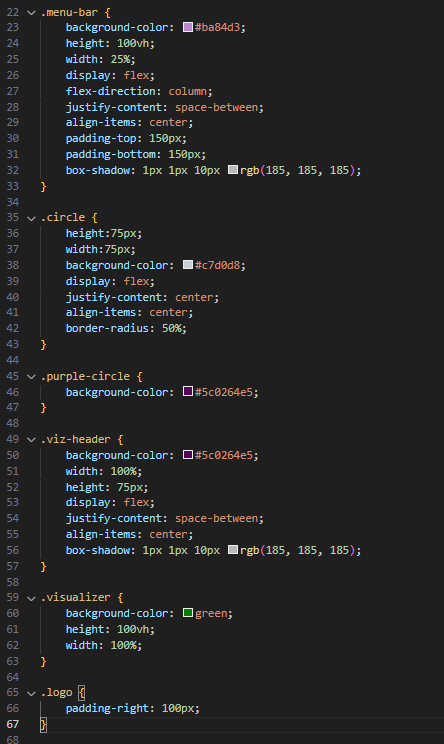


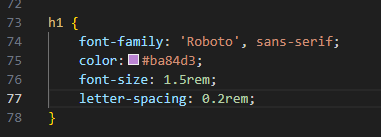
* En el HTML agregue dos elementos adicionales a visualizar debajo del div de visualización proveniente de google fonts



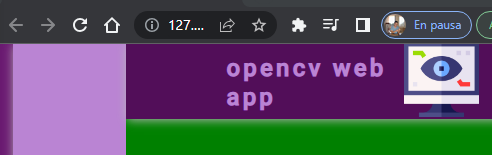


* Agregue los estilos para cada caso como se muestra



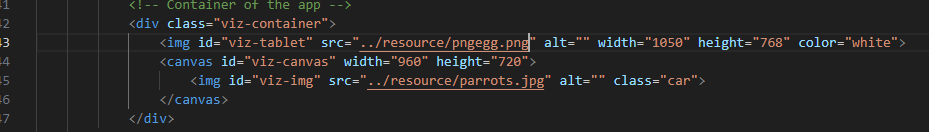


* Valide la página

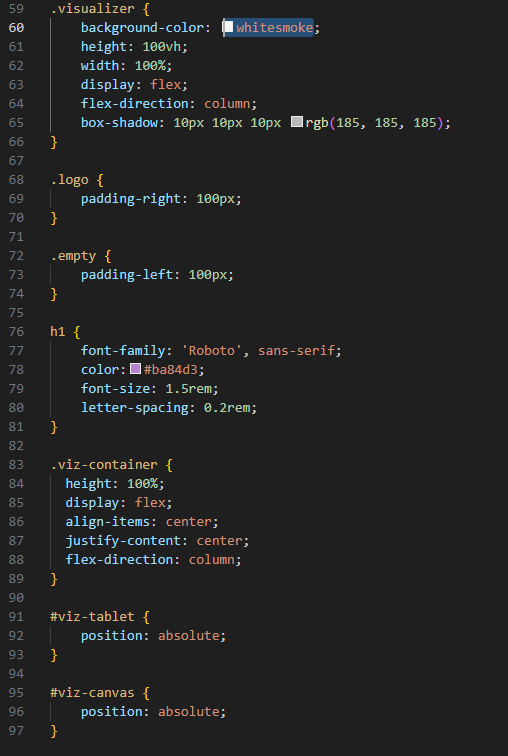


**III.5 – Creación de contenedor de visualización**

* Para opencv desplegarse satisfactoriamente necesita tener una canvas



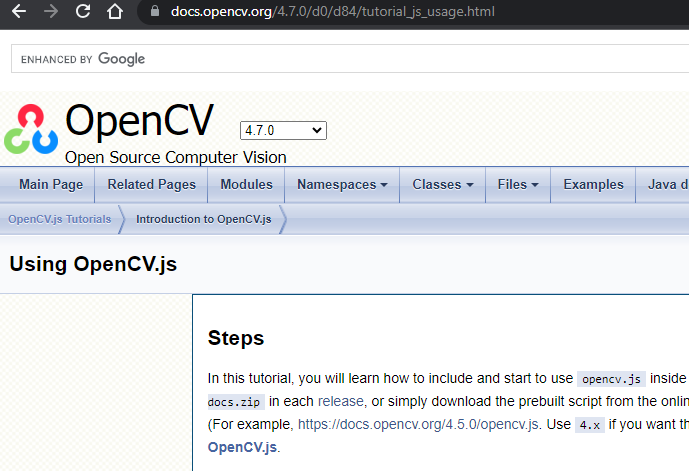
* Para finalmente modificar los estilos



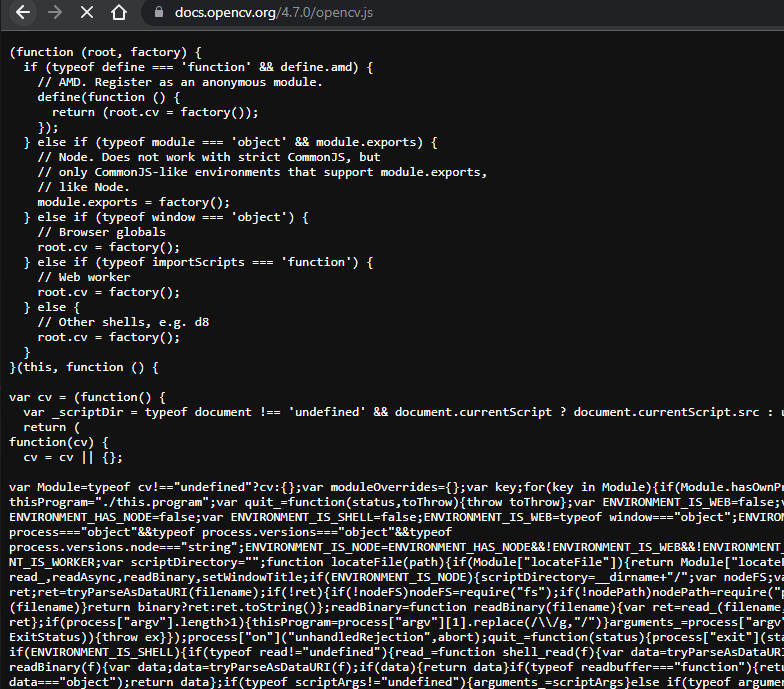
**IV – Uso de opencv en la web**

**IV.1 – Importación**

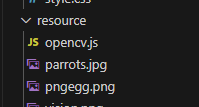
Ahora vieen la parte interesante, como utilizar OpenCVJS. Para esto nos moveremos a



* Al hacer click en el vinculo vemos que para descargar la version en javascript debemos descargar el archivo js que tiene opencv



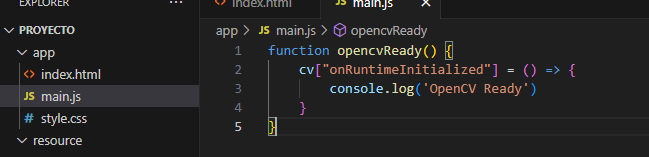
* Descargue opencv en la carpeta de recursos



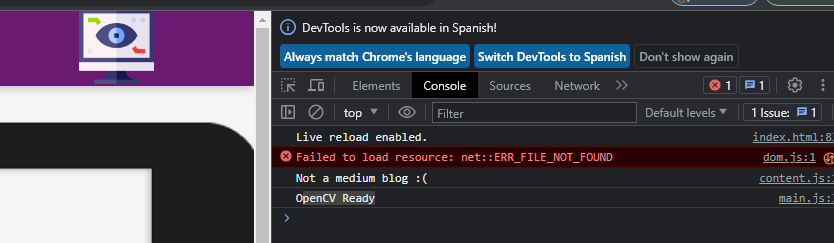
* Ingrese opencv en la pagina web



* Escriba la función de arranque de opencv

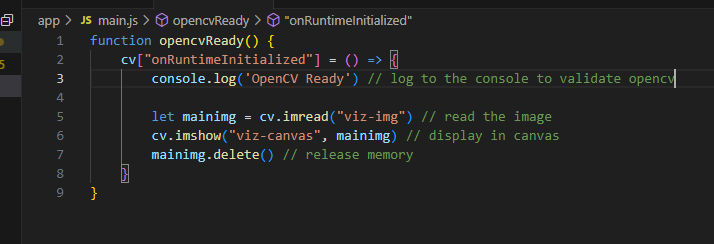


* Finalmente arranque el servidor y valida que se carga opencv correctamente (con F12 podemos validar con las herramientas del desarrollador de google)
* Note que sale el mensaje ‘OpenCV Ready’

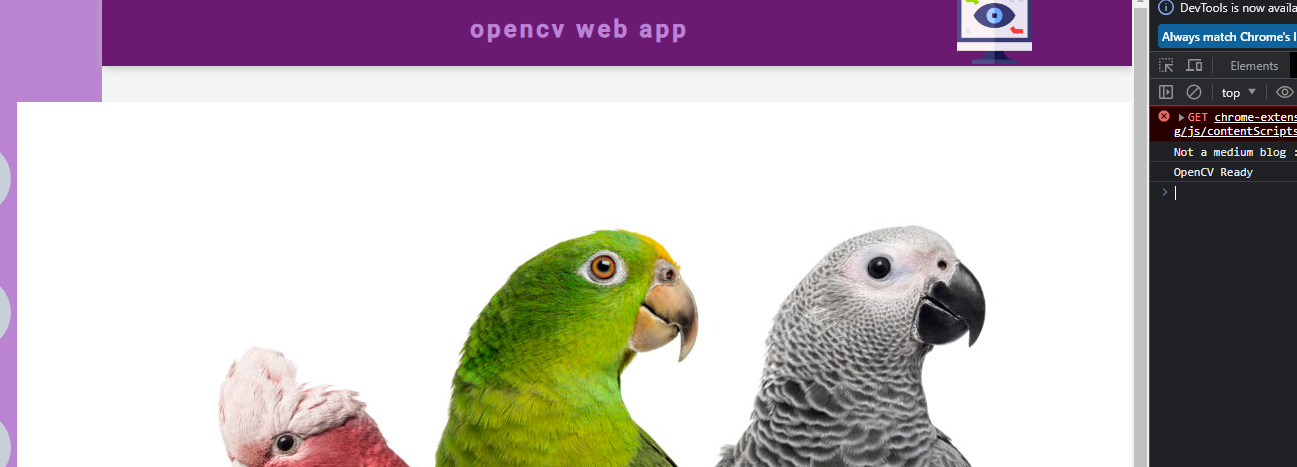


**IV.2 – Añada la imagen inicial**

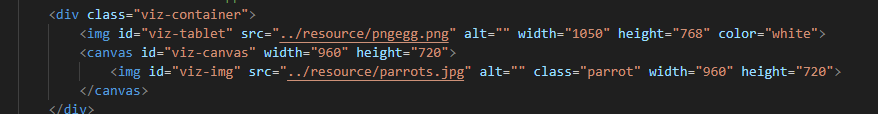
* Finalmente modifique un poco más la función original y llame a la imagen de los parrots.png



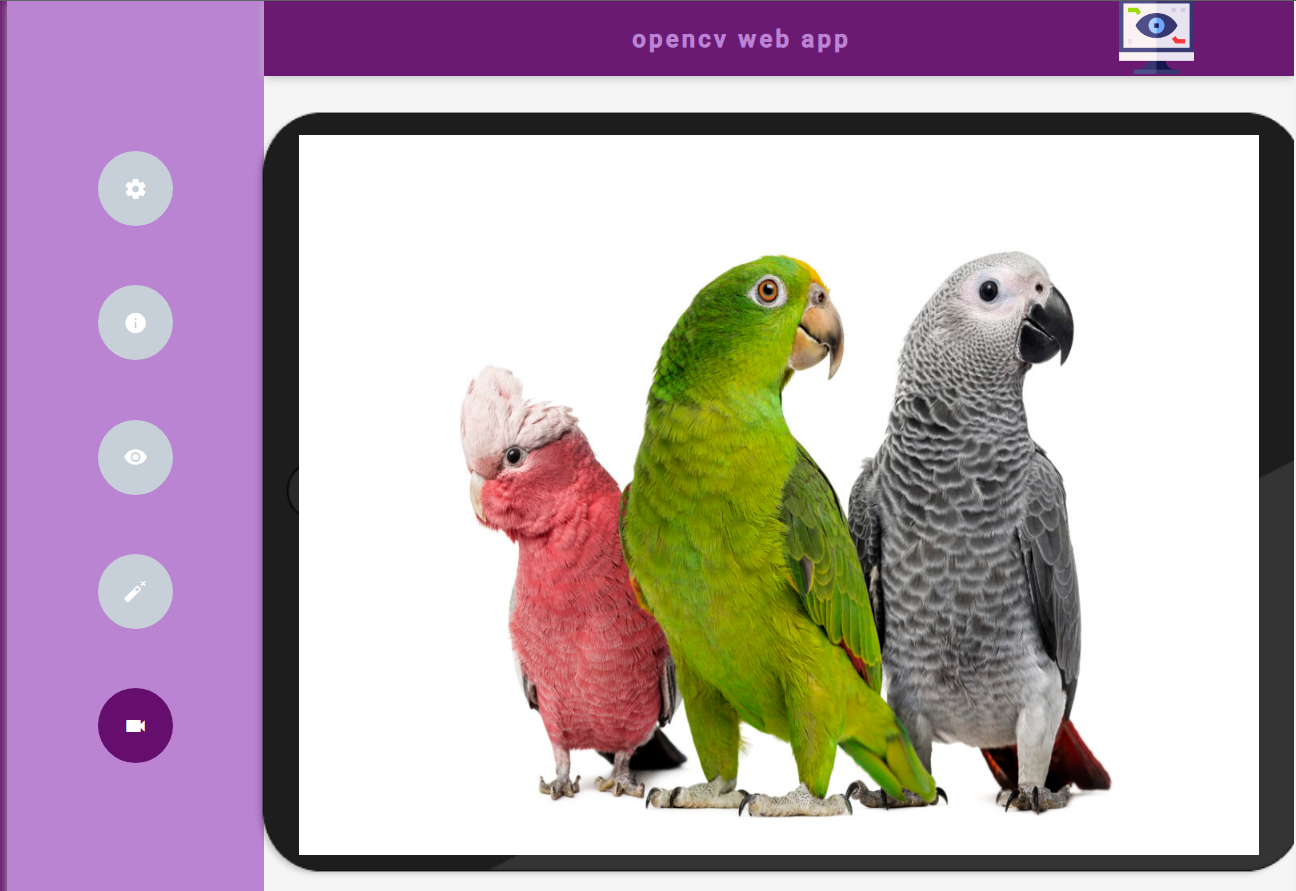
* Actualice el navegador, la imagen sale pero de un tamaño muy superior, aún así cumple el objetivo de desplegar por opencv



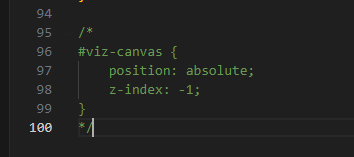
* Para acomodar este pequeño error simplemente modifique el archivo html para que el canvas tenga el mismo tamaño de la image

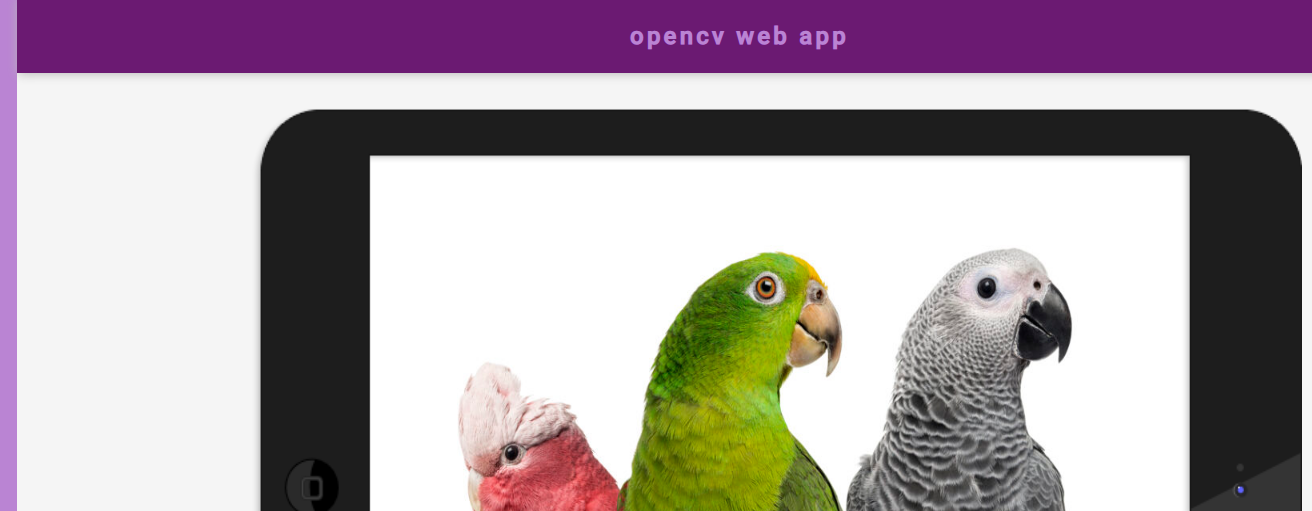


* Despliegue nuevamente



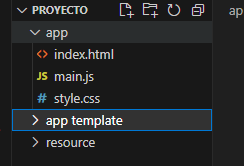
* Para crear el efecto de que la Tablet está sobre la imagen deberíamos poder poner en el estilo css z-index: -1, pero en mi caso no funciona, por lo que simplemente comentaré el código.



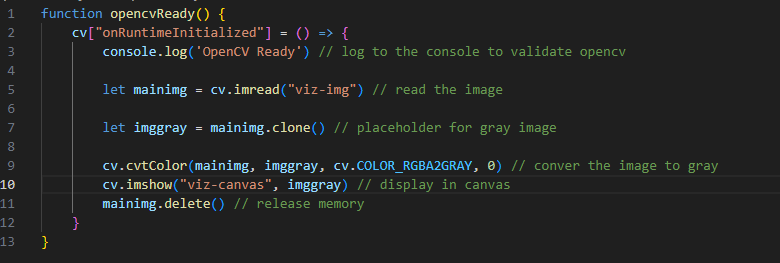


**IV.3 – Agregando Funciones**

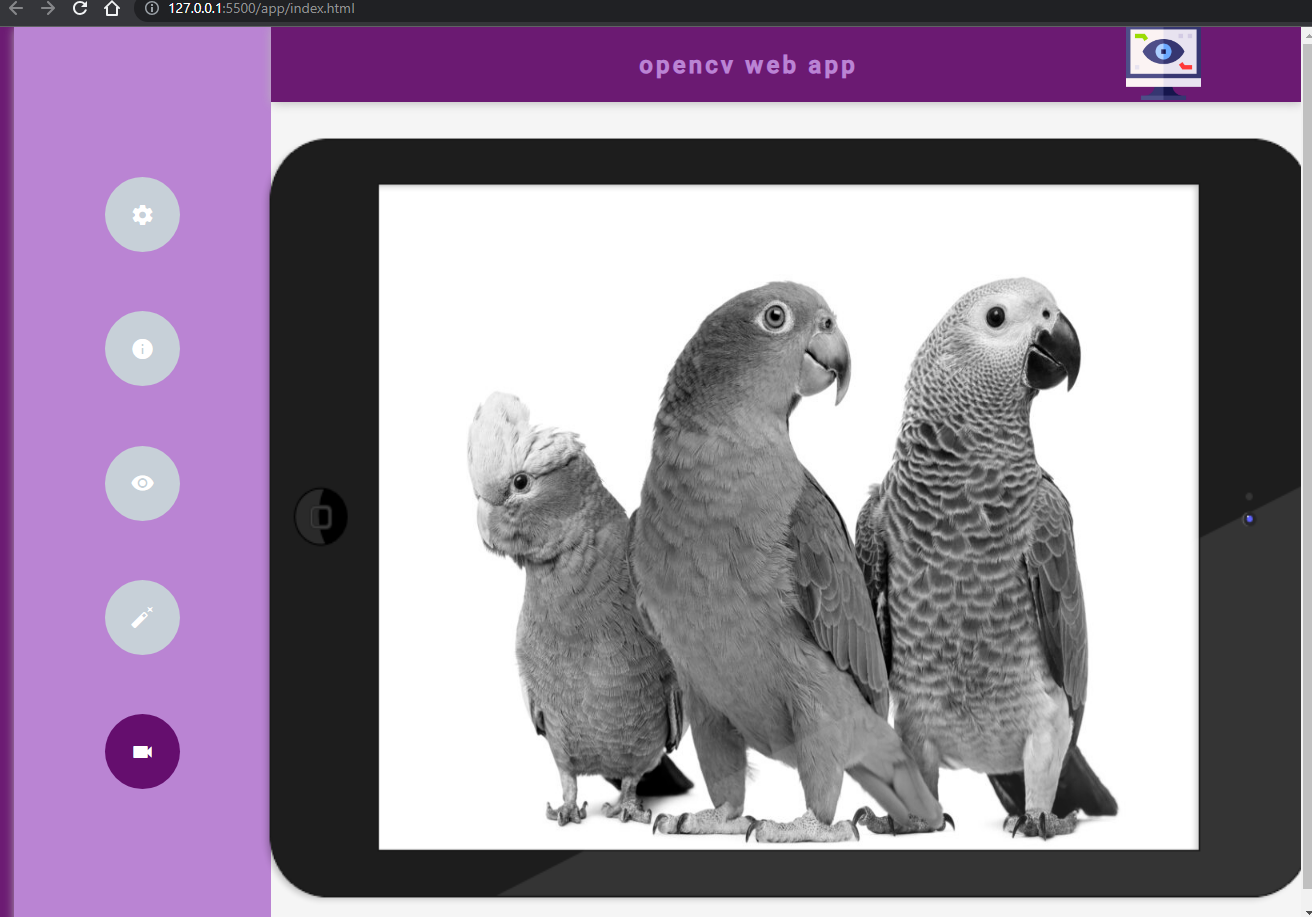
* Primero guardaremos el proyecto anterior como plantilla, copiaremos todo a otro archivo



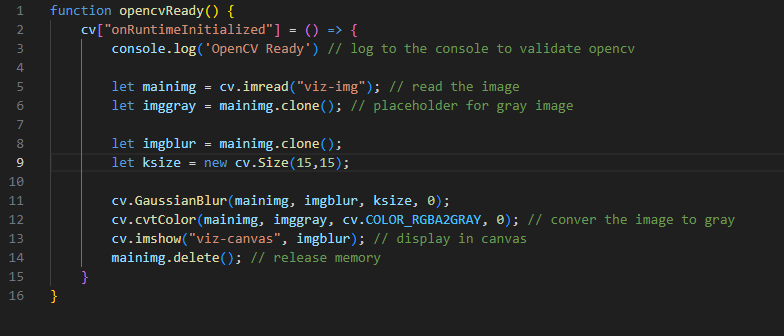
* Ahora seguiremos modificando la aplicación para transformar la imagen a gris



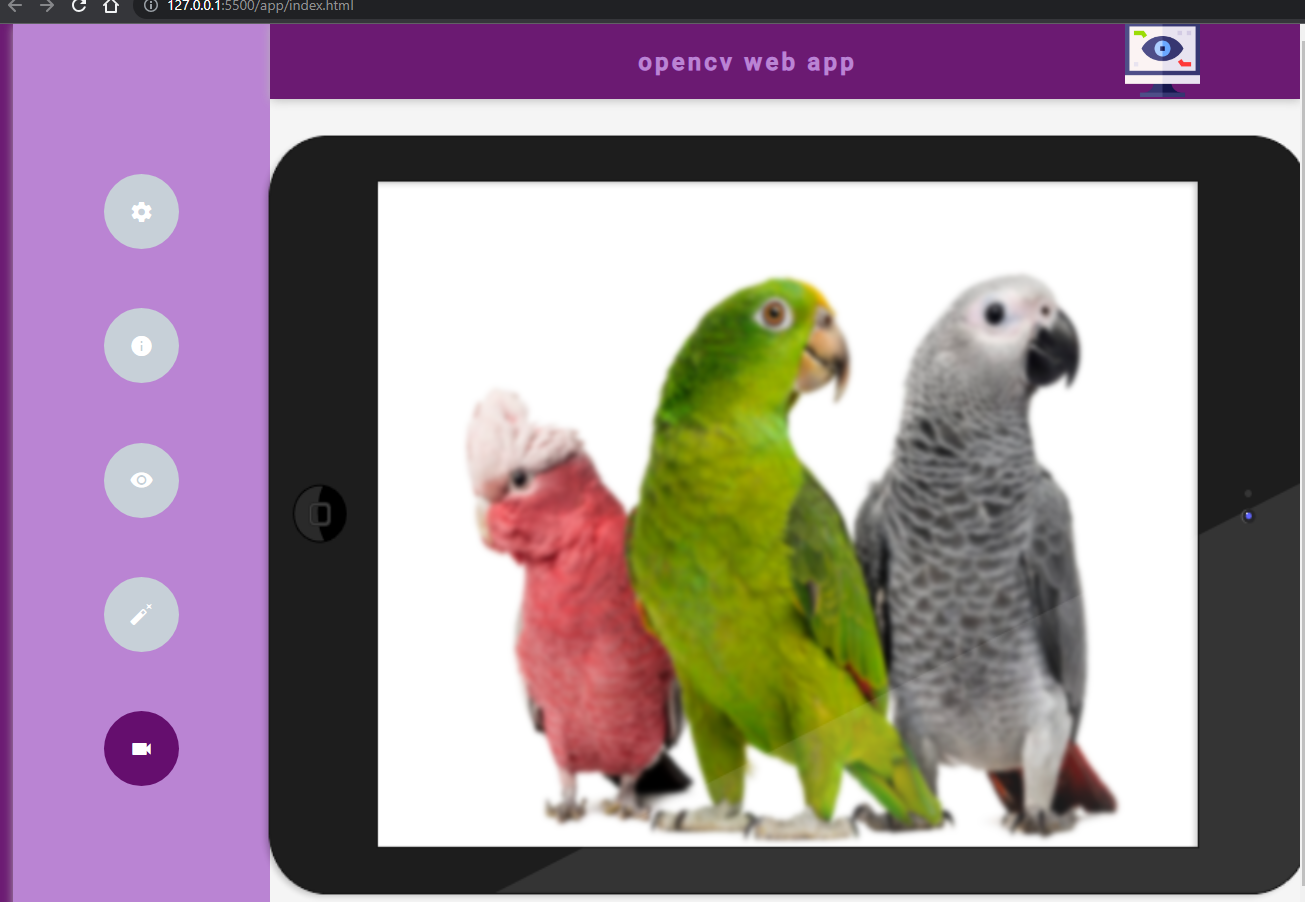
* Despliegue la imagen para ver el resultado



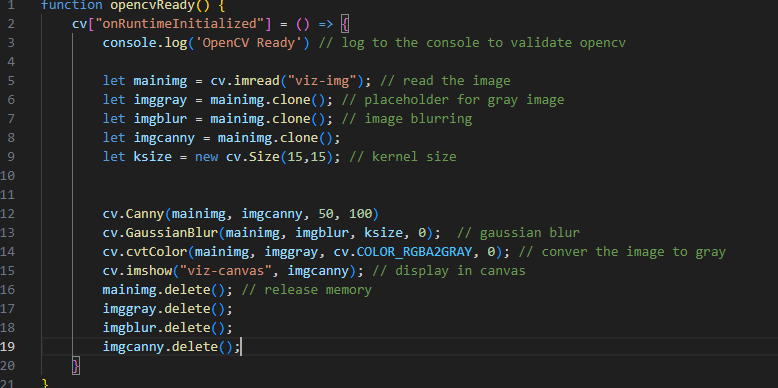
* Ahora añadiremos la función de difuminación gausiana



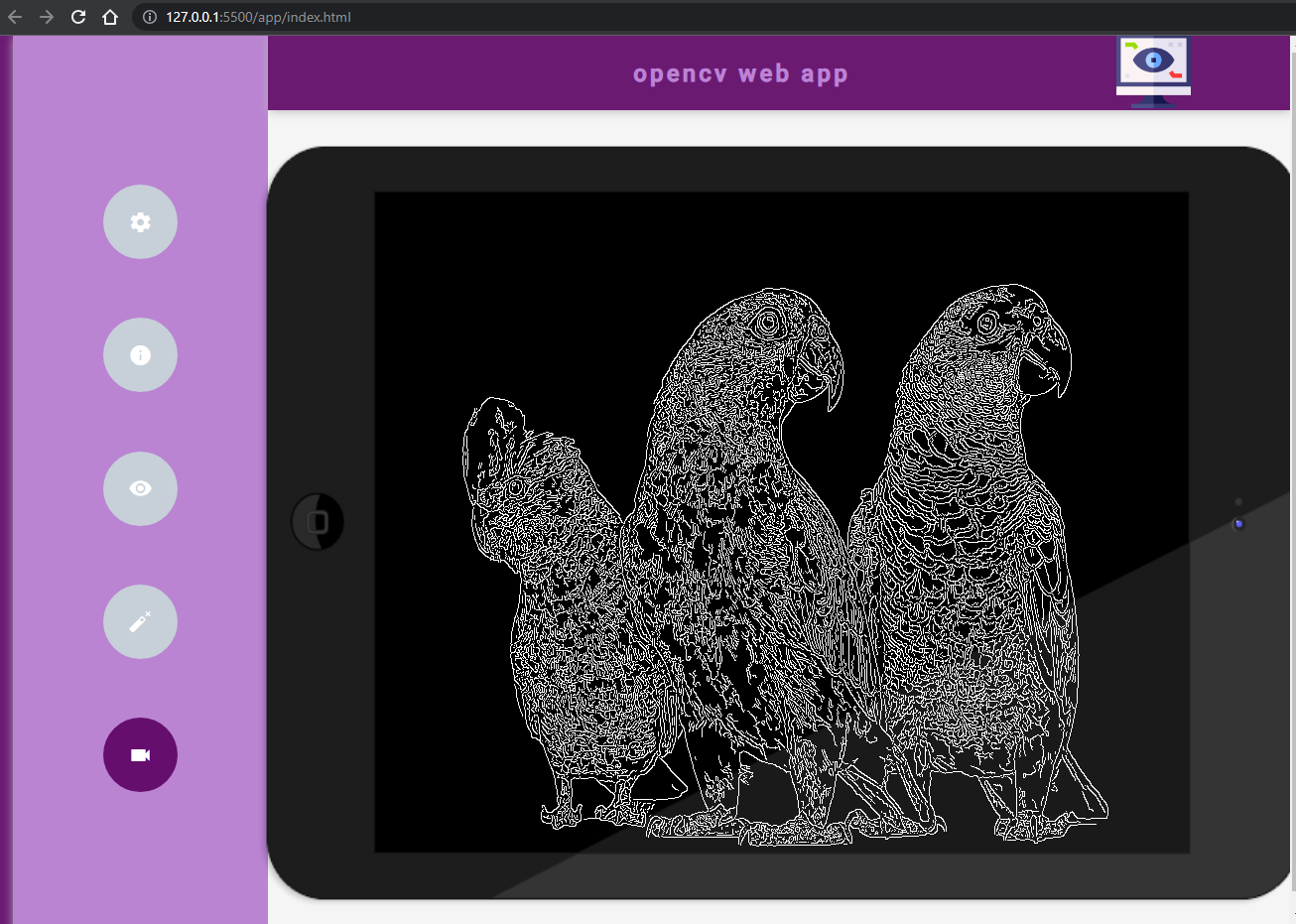
* Validamos las funciones y notamos que la imagen original está difuminada



* Probemos la función de detección de bordes
* NOTA: Para el código previo se olvidó borrar las imágenes, si no hace esto, en aplicaciones de streaming de video podría tener un fallo por volcado de memoria en el navegador, recuerde siempre revisar la ventana de procesos y validar que google Chrome no está aumentando en meomia (Presionando SHIFT+ESC sobre la imagen desplegada)

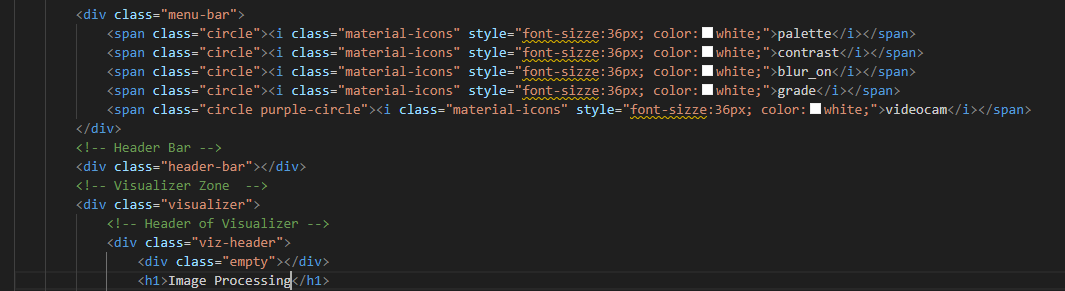


* Validemos

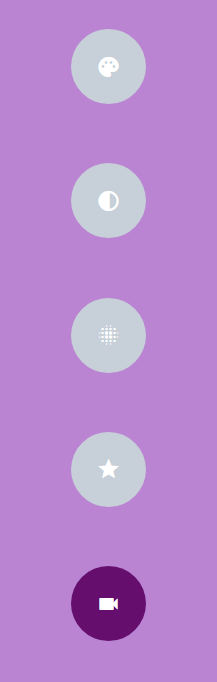


**V – Separación de las funciones básicas de navegación**

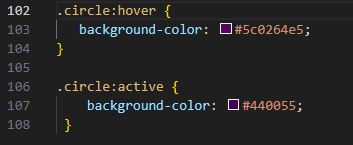
* Primeraemnte cambiemos las apariencia de los botones básicos y el nombre de la aplicación



* Seguido validemos los cambios



* Añadamos las funciones de hover (cuando se pasa el mouse sobre el botón) y cuando se mantiene activo

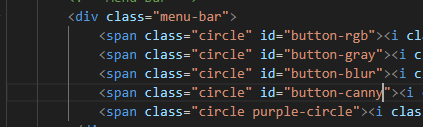


* Validamos hover y active (click) respectivamente

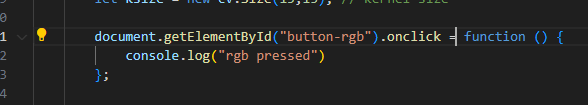
 

**VI – Separación de funciones básicas de procesamiento de imágenes**

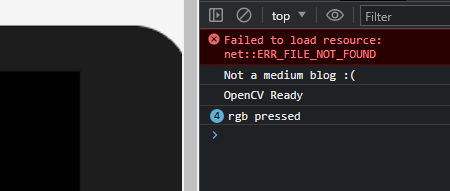
* Agregamos los ids para verlos en la página web



* Finalmente ubicamos en el código JS lo siguiente

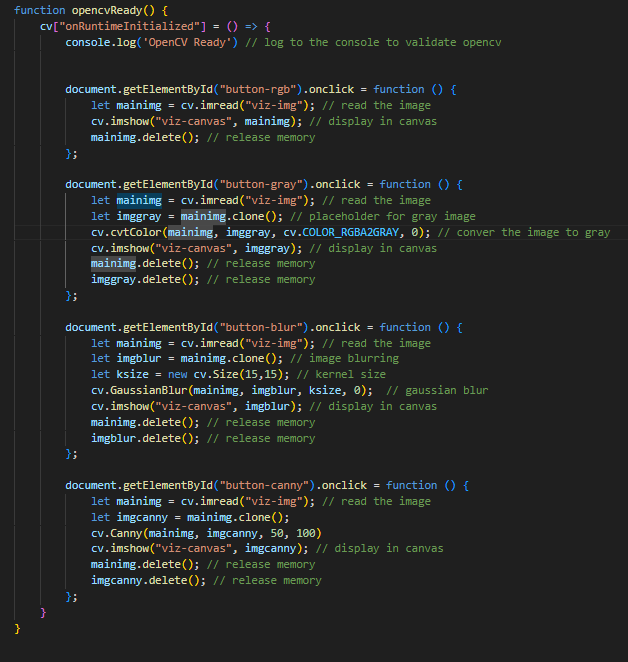


* Validamos con las herramientas del desarrollador al presionar el botón

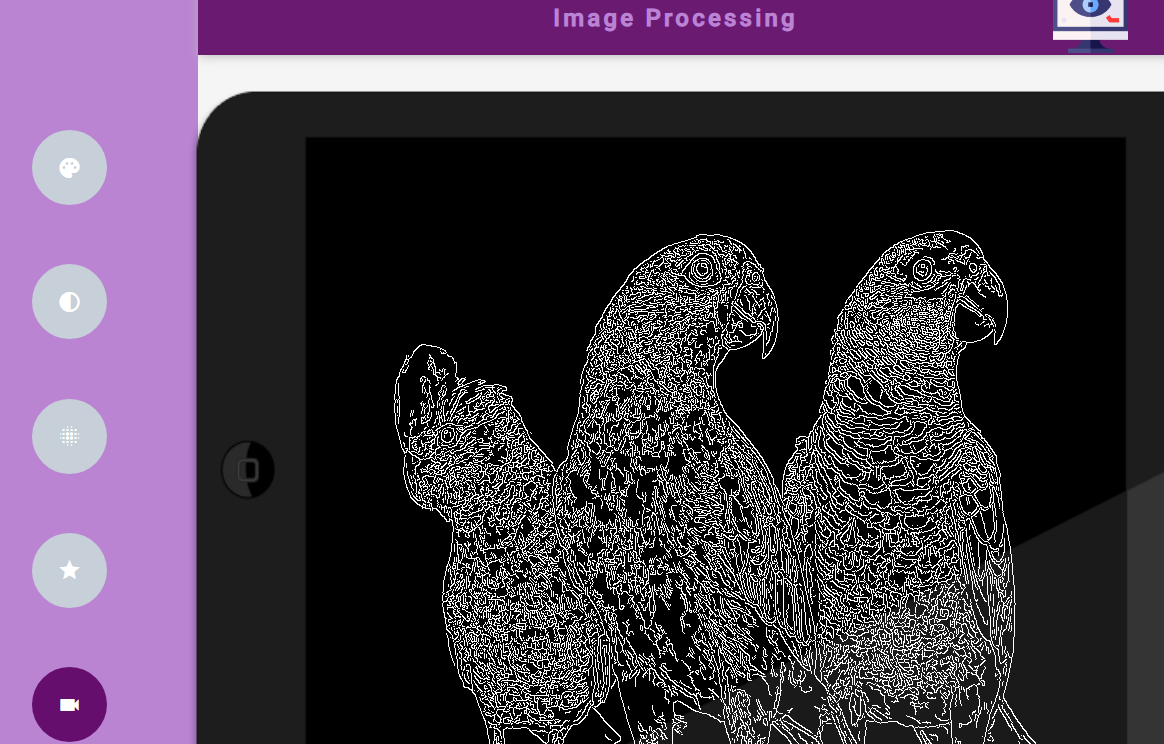


**VI – Ordenamiento de las funciones de OpenCV**

* Ahora es el momento final. Ordenaremos las funciones de OpenCV en el JS

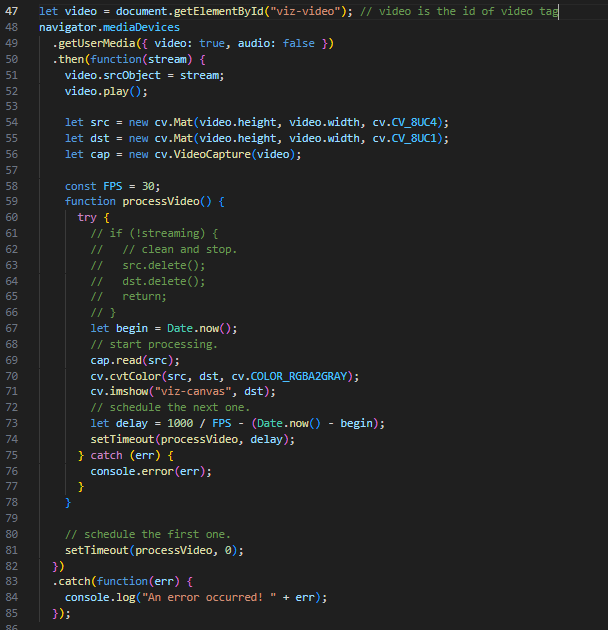


* Validamos la salida haciendo click en cada botón

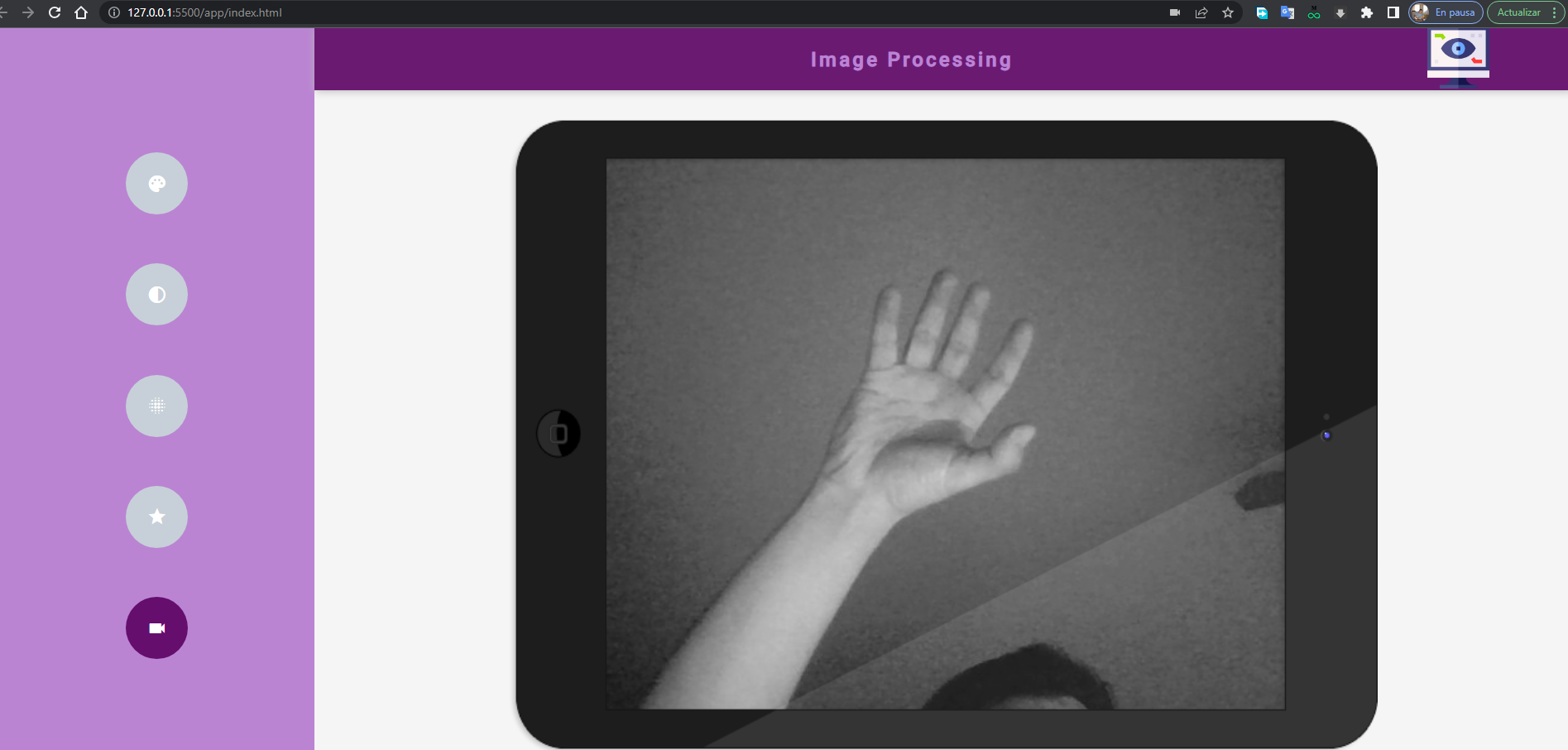


**[OPCIONAL] – Despliegue de video en tiempo real**

* Para despliegue de video en tiempo real ingrese el siguiente código en main.js
* NOTA: Por el momento con este código las imágenes quedarán deshabilitadas



* Valide la aplicación



**[APENDICE] – Lista de Código**

* index.html

<!DOCTYPE html>

<html lang="en">

<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>Computer Vision Web Application</title>

<!-- Add OpenCV -->

<script async src="../resource/opencv.js" onload="opencvReady()"></script>

<!--Linking the style-->

<link rel="stylesheet" href="style.css">

<!-- Material Icons (Google) -->

<link href="https://fonts.googleapis.com/icon?family=Material+Icons"

rel="stylesheet">

<!-- Add google font -->

<link href="https://fonts.googleapis.com/css2?family=Oswald:wght@200;300&family=Roboto:ital,wght@0,500;1,100&display=swap" rel="stylesheet">

</head>

<body>

<!-- Main Page (here goes everything) -->

<div class="main-page">

<!-- Side Line -->

<div class="side-line"></div>

<!-- Menu Bar -->

<div class="menu-bar">

<span class="circle" id="button-rgb"><i class="material-icons" style="font-sizze:36px; color:white;">palette</i></span>

<span class="circle" id="button-gray"><i class="material-icons" style="font-sizze:36px; color:white;">contrast</i></span>

<span class="circle" id="button-blur"><i class="material-icons" style="font-sizze:36px; color:white;">blur\_on</i></span>

<span class="circle" id="button-canny"><i class="material-icons" style="font-sizze:36px; color:white;">grade</i></span>

<span class="circle purple-circle" id="button-video" onclick="processVideo()"><i class="material-icons" style="font-sizze:36px; color:white;">videocam</i></span>

</div>

<!-- Header Bar -->

<div class="header-bar"></div>

<!-- Visualizer Zone -->

<div class="visualizer">

<!-- Header of Visualizer -->

<div class="viz-header">

<div class="empty"></div>

<h1>Image Processing</h1>

<div class="logo"><img src="../resource/vision.png" alt="" width="75px"></div>

</div>

<!-- Container of the app -->

<div class="viz-container">

<img id="viz-tablet" src="../resource/pngegg.png" alt="" width="1050" height="768" color="white">

<canvas id="viz-canvas" width="960" height="720">

<img id="viz-img" src="../resource/parrots.jpg" alt="" class="parrot" width="960" height="720">

<video id="viz-video" width="960" height="720"></video>

</canvas>

</div>

</div>

</div>

<!-- Link the script-->

<script src="main.js"></script>

</body>

</html>

* main.js

**function** opencvReady() {

cv["onRuntimeInitialized"] = () => {

console.log('OpenCV Ready') // log to the console to validate opencv

document.getElementById("button-rgb").onclick = **function** () {

video.pause()

**let** mainimg = cv.imread("viz-img"); // read the image

cv.imshow("viz-canvas", mainimg); // display in canvas

mainimg.**delete**(); // release memory

};

document.getElementById("button-gray").onclick = **function** () {

**let** mainimg = cv.imread("viz-img"); // read the image

**let** imggray = mainimg.clone(); // placeholder for gray image

cv.cvtColor(mainimg, imggray, cv.COLOR\_RGBA2GRAY, **0**); // conver the image to gray

cv.imshow("viz-canvas", imggray); // display in canvas

mainimg.**delete**(); // release memory

imggray.**delete**(); // release memory

};

document.getElementById("button-blur").onclick = **function** () {

**let** mainimg = cv.imread("viz-img"); // read the image

**let** imgblur = mainimg.clone(); // image blurring

**let** ksize = **new** cv.Size(**15**,**15**); // kernel size

cv.GaussianBlur(mainimg, imgblur, ksize, **0**); // gaussian blur

cv.imshow("viz-canvas", imgblur); // display in canvas

mainimg.**delete**(); // release memory

imgblur.**delete**(); // release memory

};

document.getElementById("button-canny").onclick = **function** () {

**let** mainimg = cv.imread("viz-img"); // read the image

**let** imgcanny = mainimg.clone();

cv.Canny(mainimg, imgcanny, **50**, **100**)

cv.imshow("viz-canvas", imgcanny); // display in canvas

mainimg.**delete**(); // release memory

imgcanny.**delete**(); // release memory

};

}

}

* Main.js

**let** video = document.getElementById("viz-video"); // video is the id of video tag

navigator.mediaDevices

.getUserMedia({ video: **true**, audio: **false** })

.then(**function**(stream) {

video.srcObject = stream;

video.play();

**let** src = **new** cv.Mat(video.height, video.width, cv.CV\_8UC4);

**let** dst = **new** cv.Mat(video.height, video.width, cv.CV\_8UC1);

**let** cap = **new** cv.VideoCapture(video);

**const** FPS = **30**;

**function** processVideo() {

**try** {

// if (!streaming) {

// // clean and stop.

// src.delete();

// dst.delete();

// return;

// }

**let** begin = Date.now();

// start processing.

cap.read(src);

cv.cvtColor(src, dst, cv.COLOR\_RGBA2GRAY);

cv.imshow("viz-canvas", dst);

// schedule the next one.

**let** delay = **1000** / FPS - (Date.now() - begin);

setTimeout(processVideo, delay);

} **catch** (err) {

console.error(err);

}

}

// schedule the first one.

setTimeout(processVideo, **0**);

})

.**catch**(**function**(err) {

console.log("An error occurred! " + err);

});

* style.css

/\* Complete Page Style \*/

\*{

margin: **0**px; /\* Margin of 0 pixels \*/

box-sizing: border-box; /\* Box sizing will be done automatically\*/

}

body {

background-color: white;

}

.main-page {

height: **20**px;

display: flex;

}

.side-line {

background-color: #**5**c0264e5;

height: **100**vh;

width: **25**px;

}

.menu-bar {

background-color: #ba84d3;

height: **100**vh;

width: **25**%;

display: flex;

flex-direction: column;

justify-content: space-between;

align-items: center;

padding-top: **150**px;

padding-bottom: **150**px;

box-shadow: **1**px **1**px **10**px rgb(**185**, **185**, **185**);

}

.circle {

height:**75**px;

width:**75**px;

background-color: #c7d0d8;

display: flex;

justify-content: center;

align-items: center;

border-radius: **50**%;

}

.purple-circle {

background-color: #**5**c0264e5;

}

.viz-header {

background-color: #**5**c0264e5;

width: **100**%;

height: **75**px;

display: flex;

justify-content: space-between;

align-items: center;

box-shadow: **1**px **1**px **10**px rgb(**185**, **185**, **185**);

}

.visualizer {

background-color: whitesmoke;

height: **100**vh;

width: **100**%;

display: flex;

flex-direction: column;

box-shadow: **10**px **10**px **10**px rgb(**185**, **185**, **185**);

}

.logo {

padding-right: **100**px;

}

.empty {

padding-left: **100**px;

}

h1 {

font-family: 'Roboto', sans-serif;

color:#ba84d3;

font-size: **1.5**rem;

letter-spacing: **0.2**rem;

}

.viz-container {

height: **100**%;

display: flex;

align-items: center;

justify-content: center;

flex-direction: column;

}

#viz-tablet {

position: absolute;

}

/\*

#viz-canvas {

position: absolute;

z-index: -1;

}

\*/

.circle:hover {

background-color: #**5**c0264e5;

}

.circle:active {

background-color: #**440055**;

}