

INSTITUTO TECNOLÓGICO DE TIJUANA
DEPARTAMENTO DE SISTEMAS



MANUAL TÉCNICO PARA PRONOSTICADOR DEMOGRÁFICO

PRESENTA:

Oscar Hernández Medina
18212196

Alex Rivera Pérez
18212259

BAJO LA ASESORÍA:

INTERNA: Dr. Arnulfo Alanis Garza

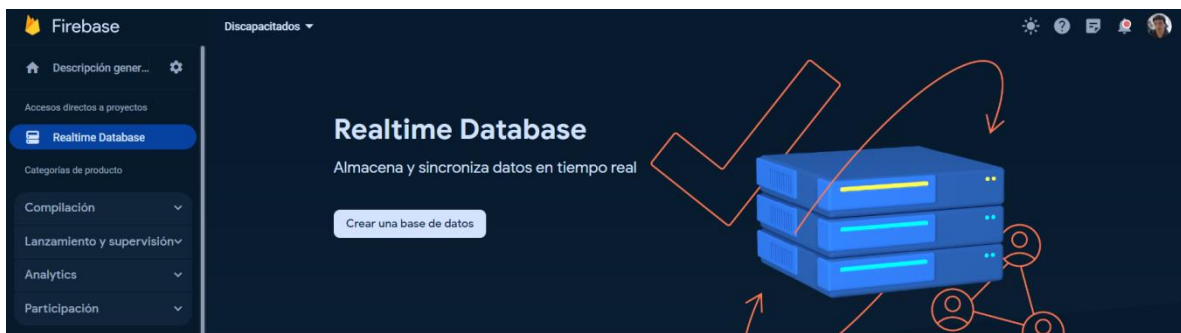
EXTERNA: Dr. Bogar Yail Márquez Lobato

ÍNDICE DE CONTENIDO

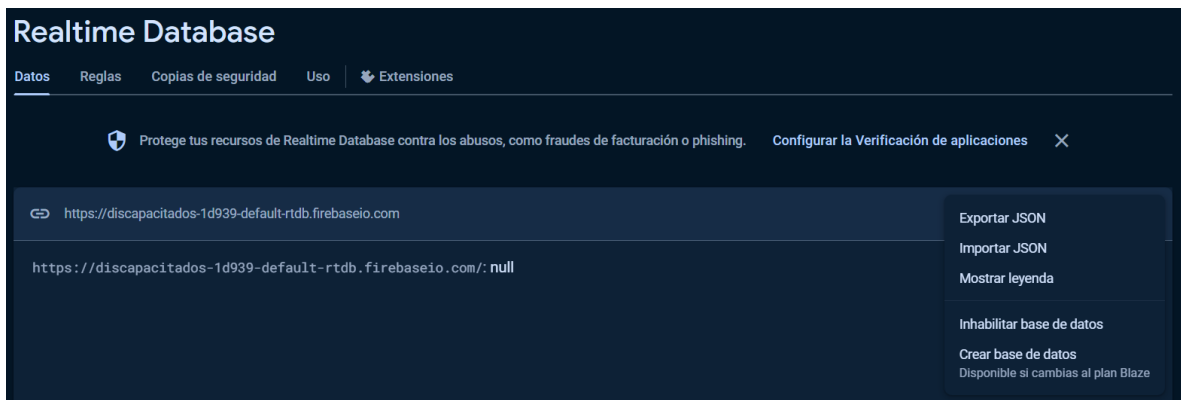
BASE DE DATOS	3
API	5
MODELOS DE APRENDIZAJE	6
EXPORTACIÓN DE MODELOS DE APRENDIZAJE	9
IMPORTANDO MODELOS EN JAVASCRIPT	9
PÁGINA INICIO	11
PÁGINA MODELOS	13
PÁGINA SOBRE NOSOTROS	15
CÓDIGO CSS	16

BASE DE DATOS

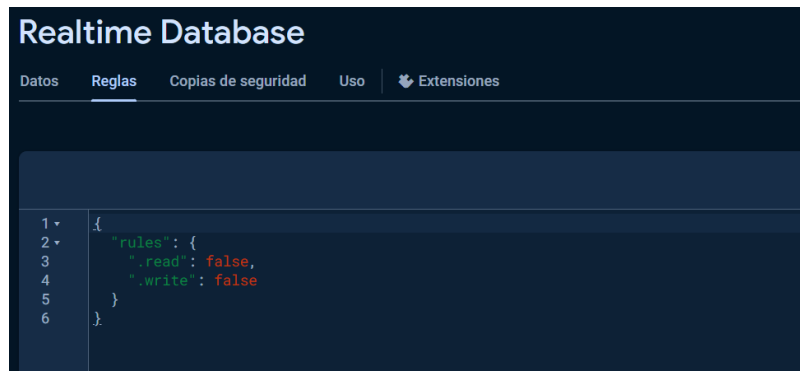
Para crear la base de datos donde almacenaremos la información necesaria para entrenar nuestros modelos, lo primero que debemos hacer es crear una cuenta de Firebase. Una vez creada nuestra cuenta en Firebase necesitamos crear un proyecto, luego seleccionamos la opción 'Compilación' en el menú izquierdo y seleccionamos 'Realtime Database', y posteriormente pulsamos el botón 'Crear una base de datos'.



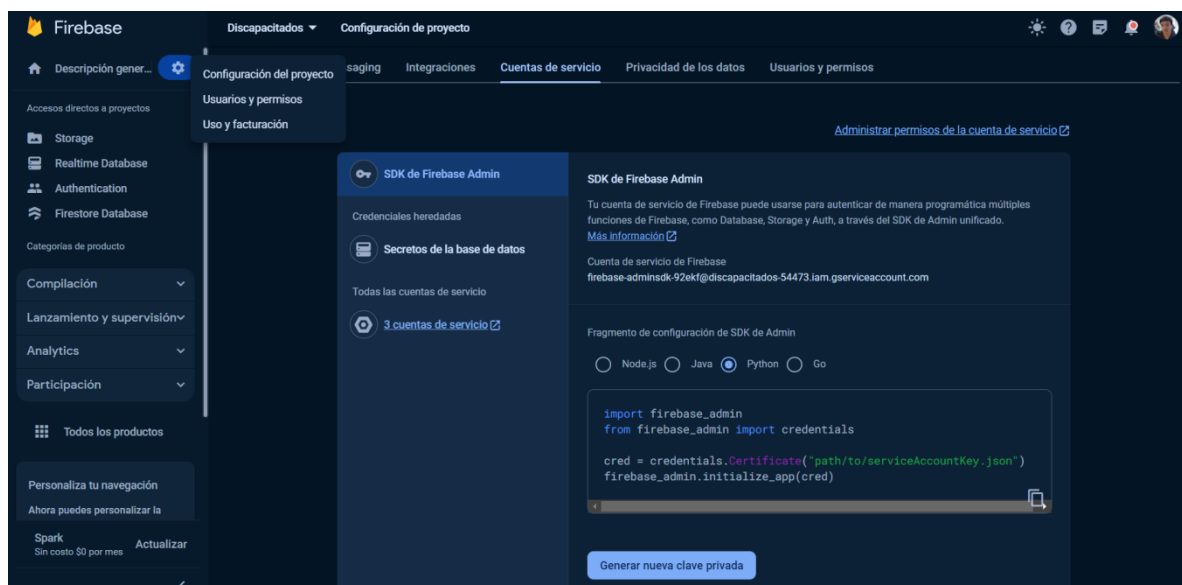
Una vez creada la base de datos pulsamos los 3 puntos en el lado derecho, elegimos la opción 'Importar JSON' y seleccionamos el archivo JSON que contiene la información que utilizaremos.



Luego de importar nuestro archivo JSON iremos al menú de la parte superior y pulsaremos donde dice 'Reglas', debemos asegurarnos de que tanto en la propiedad 'read' como en 'write' diga 'false'. Eso es para tener la certeza de que no cualquier persona tendrá acceso y podrá modificar la información de nuestra base de datos.



Para poder acceder a la información de nuestra base de datos, desde el API debemos crear una credencial que es como una llave que nos brindara el acceso. En el menú de la parte izquierda seleccionamos 'Descripción general' y elegimos la opción 'Configuración del proyecto', luego vamos a 'Cuentas de servicio' y pulsamos en 'Generar nueva clave privada' la cual nos generara un archivo JSON que deberemos de guardar.



API

Ahora podemos proseguir con la creación del api. Para ello importaremos la librería 'fastapi' para crear un api de forma rápida y sencilla, y 'firebase_admin' para poder autenticar nuestro acceso a la base de datos.

```
from fastapi import FastAPI
from fastapi.responses import JSONResponse
from firebase_admin import credentials, initialize_app, db
```

Para la creación del api primero ocupamos crear una variable que llamaremos 'cred' donde almacenaremos el archivo JSON que guardamos anteriormente y que nos sirve de credencial. Luego creamos una variable llamada 'firebase_app' donde iniciaremos la conexión a la base de datos, para ello ocupamos la variable donde almacenamos el archivo JSON y la dirección url que nos genera firebase.

```
app=FastAPI()

cred=credentials.Certificate('./json/discapitados-54473-firebase-adminsdk-92ekf-f52b6a0147.json')
firebase_app=initialize_app(cred,{ 'databaseURL': 'https://discapitados-54473-default-rtdb.firebaseio.com/' })
```

Lo que sigue es crear la dirección a la cual el usuario podrá acceder para obtener la información de la base de datos. Para ello crear una variable llamada 'app' donde almacenaremos el método 'FastAPI', luego debemos crear la dirección que en nuestro caso será '/Data' y por último falta definir el método que se ejecutará al acceder a dicha dirección.

Dentro de nuestro método crearemos una referencia a la raíz de la base de datos ('/'), obtenemos los datos en esa dirección con el método 'get' y lo devolvemos como una respuesta en formato JSON.

```
app = FastAPI()

# Rutas de la bd
@app.get("/Data")
def main():
    # Lectura de datos desde Firebase
    ref = db.reference('/')
    data = ref.get()
    return JSONResponse(content=data)
```

A continuación, se muestra un ejemplo de cómo podemos acceder a la información de la base de datos mediante el api que acabamos de crear.

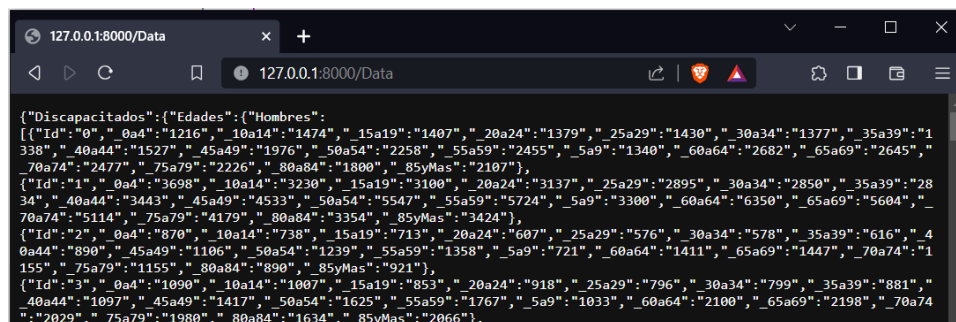
Primero ocupamos importar la librería uvicorn que nos ayudara a correr el api, una vez importada la librería abrimos una terminal y ponemos el siguiente comando: 'uvicorn api:app' siendo 'api' el nombre del archivo donde creamos el api y 'app' el nombre de la variable que creamos anteriormente donde almacenamos el método 'FastAPI'.



```
> json
> Modelos_H5
> Modelos.js
api.py M
main.py
prueba.py
prueba2.py

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\alexm\OneDrive\Documentos\Proyectos\Discapacitados> cd Api
PS C:\Users\alexm\OneDrive\Documentos\Proyectos\Discapacitados\Api> uvicorn api:app
INFO: Started server process [16568]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

Ahora debemos poner la dirección que nos proporciona uvicorn seguido de la dirección que nosotros creamos, la cual fue '/Data' en nuestro navegador web y podremos ver toda la información que contiene la base de datos.



```
127.0.0.1:8000/Data
{"Discapacitados":{"Edades":{"Hombres":[{"Id":"0","0a4":"1216","10a14":"1474","15a19":"1407","20a24":"1379","25a29":"1430","30a34":"1377","35a39":"1338","40a44":"1527","45a49":"1976","50a54":"2258","55a59":"2455","5a9":"1340","60a64":"2682","65a69":"2645","70a74":"2477","75a79":"2226","80a84":"1800","85yMas":"2107"}],{"Id":"1","0a4":"3698","10a14":"3230","15a19":"3100","20a24":"3137","25a29":"2895","30a34":"2850","35a39":"2834","40a44":"3443","45a49":"4533","50a54":"5547","55a59":"5724","5a9":"3300","60a64":"6350","65a69":"5604","70a74":"5114","75a79":"4179","80a84":"3354","85yMas":"3424"}],{"Id":"2","0a4":"870","10a14":"738","15a19":"713","20a24":"607","25a29":"576","30a34":"578","35a39":"616","40a44":"890","45a49":"1106","50a54":"1239","55a59":"1358","5a9":"721","60a64":"1411","65a69":"1447","70a74":"1155","75a79":"1155","80a84":"890","85yMas":"921"}],{"Id":"3","0a4":"1090","10a14":"1007","15a19":"853","20a24":"918","25a29":"796","30a34":"799","35a39":"881","40a44":"1097","45a49":"1417","50a54":"1625","55a59":"1767","5a9":"1033","60a64":"2100","65a69":"2198","70a74":"2029","75a79":"1980","80a84":"1634","85yMas":"2066"}]}}
```

MODELOS DE APRENDIZAJE

Lo que sigue es crear los modelos de aprendizaje y exportarlos para poder usarlos desde una aplicación web. Para ello lo primero es hacer una petición (con el método 'get' de la librería 'request') al api que creamos para obtener los datos con los que entrenaremos nuestros modelos. Hacemos una comprobación al hacer la petición, si nos devuelve un código 200 significa que la petición se realizó correctamente, y obtenemos todos los datos en formato json. Lo que sigue es filtrar solo la información que necesitaremos y la guardamos en arreglos, en este caso lo guardamos en los

arreglos: D (discapitados generales), HD (hombres discapacitados), MD (Mujeres discapacitados) y P (población).

```
peticion=requests.get('http://127.0.0.1:8000/Data')

#Datos discapacitados (general)
if(peticion.status_code==200):
    data=peticion.json()
    for x in range(0,len(data['Discapacitados']['General'])):
        D.append(data['Discapacitados']['General'][x]['Discapacitados'])
        HD.append(data['Discapacitados']['General'][x]['HombresD'])
        MD.append(data['Discapacitados']['General'][x]['MujeresD'])

#Generacion de arreglo para el modelo MDiscapacitados
if(peticion.status_code==200):
    data=peticion.json()
    for x in range(0,len(data['Poblacion'])):
        P.append(data['Poblacion'][x]['Poblacion'])
```

Se hace lo mismo para toda la demás información que necesitaremos, guardando dicha información en un arreglo diferente ya que posteriormente lo necesitaremos para entrenar los modelos de aprendizaje.

Crearemos un método para crear el modelo de aprendizaje ya que necesitaremos crear cinco modelos de aprendizaje que, aunque tendrán la misma estructura cada uno realizara una predicción diferente. Este modelo tendrá cinco parámetros: input, output, y, x y épocas. Input indica la forma que tendrá el arreglo de entrada (por ejemplo (3,) indica que el arreglo tiene 3 columnas), output la forma que tendrá el arreglo de salida de forma similar a input, x indica el arreglo de entrada, y el arreglo de salida y épocas la cantidad de épocas o pasadas que hará el modelo al entrenarse. Definimos el tipo de modelo que crearemos con 'keras.Sequential()' en este caso indicamos que crearemos un modelo secuencial el cual indica que pondremos una capa después de otra. Definimos la forma de la entrada con 'keras.Input'. Lo que sigue es definir las capas, en este caso serán 3 capas densas con 64, 64 y 32 neuronas respectivamente y todas las capas usarán la función de activación relu. Por último, definimos la capa de salida con la forma que tendrá dicha salida.

Usamos el método 'compile' para compilar el modelo, para el optimizador usamos Adam, un índice de aprendizaje de 0.001 y el error cuadrático medio para la función de perdida. Por último, usamos el método fit para entrenar el modelo, como parámetros le pasamos el arreglo de entrada, arreglo de salida, cantidad de épocas y el verbose lo definimos en 0 para que no nos muestre información en consola cada que se complete una época.

```
#Funcion para la creacion del modelo
def Modelo(input,output,x,y,epocas):

    #Creacion de modelo de prediccion para hombres
    modelo = keras.Sequential()
    modelo.add(keras.Input(shape=(input,)))
    modelo.add(layers.Dense(64, activation='relu',name='oculta1'))
    modelo.add(layers.Dense(64, activation='relu',name='oculta2'))
    modelo.add(layers.Dense(32, activation='relu',name='oculta3'))
    modelo.add(layers.Dense(output,name='oculta4'))

    modelo.compile(
        optimizer=keras.optimizers.Adam(learning_rate=0.0001),
        loss=keras.losses.MeanSquaredError()
    )

    #Entrenar modelo
    modelo.fit(x,y, epochs=epocas, verbose=0)

    return modelo
```


Para crear un modelo de aprendizaje diferente solo llamamos al método creado anteriormente, le pasamos los parámetros correspondientes (forma del arreglo de entrada y salida, arreglo de entrada y salida y cantidad de épocas) y cada modelo lo almacenamos en una variable diferente.

```
MDiscapacitados=Modelo(1,3,x,y,3500)
MHombresTipo=Modelo(1,6,x,y,3500)
MMujeresTipo=Modelo(1,6,x,y,3500)
MHombresEdad=Modelo(1,18,x,y,3500)
MMujeresEdad=Modelo(1,18,x,y,3500)
```

EXPORTACIÓN DE MODELOS DE APRENDIZAJE

Para exportar un modelo primero debemos exportarlo en formato h5, eso se hace con el método 'save' sobre el modelo que queramos exportar, seguido de la ubicación donde se guardara. Luego debemos importar ese modelo y guardarlo en una variable para posteriormente convertir dicho modelo de formato h5 a json para que pueda ser importado desde una aplicación web. Se hace lo mismo con cada uno de los demás modelos.

```
#Exportar modelos en formato H5
MDiscapacitados.save('./Api/Modelos_H5/General.h5')

#Cargar modelo en formato H5
General=keras.models.load_model('./Api/Modelos_H5/General.h5')

#Exportar modelos en formato js
tfjs.converters.save_keras_model(General, './Api/Modelos_js/General')
```

IMPORTANDO MODELOS EN JAVASCRIPT

Para usar los modelos en la aplicación web lo primero que hicimos fue importar la librería de tensorflowjs y desde JavaScript mediante el método 'loadLayerModel' seleccionar la ubicación del modelo de predicción que vayamos a importar. Una vez importado el modelo usamos el método 'predict' para hacer una predicción y devolvemos la predicción en forma de arreglo. Para los demás modelos se realiza de

forma similar simplemente creando una función diferente y cargando el modelo que se vaya a utilizar.

```
//Prediccion de discapacitados general (Hombres-Mujeres)
async function modeloGeneral(){
  const model = await tf.loadLayersModel('../Api/Modelos_js/General/model.json');
  const poblacion=document.getElementById('txtPoblacion').value
  const prediccion = model.predict(tf.tensor2d([parseFloat(poblacion)], [1, 1]));
  //Convertir el tensor 'prediccion' en un arreglo unidimensional
  return Array.from(prediccion.dataSync())
}
```

Para la creación de las gráficas usaremos la librería chartjs, ahí simplemente ponemos las etiquetas que queramos en la propiedad 'labels', color de la gráfica y borde en la propiedad 'backgroundColor', 'borderColor' respectivamente y ponemos la información que vamos a mostrar en 'data'. Para las demás graficas simplemente modificamos las propiedades anteriormente mostradas.

```
//GRAFICA GENERAL
function GraficaGeneral(datos) {
  var ctx = document.getElementById('GraficaGeneral').getContext('2d');

  // Datos para la gráfica de barras
  var datosGrafica = {
    labels: ['Total', 'Hombres', 'Mujeres'],
    datasets: [{
      data: datos,
      backgroundColor: ['rgba(255, 206, 86, 0.2)', 'rgba(54, 162, 235, 0.2)', 'rgba(255, 99, 132, 0.2)'],
      borderColor: ['rgba(255, 206, 86, 1)', 'rgba(54, 162, 235, 1)', 'rgba(255,99,132,1)'],
      borderWidth: 2,
      borderRadius:5,
    }]
  };
};
```

Una vez obtenido el arreglo donde se almacena cada una de las diferentes predicciones lo último que sigue es pasar dicho arreglo a su correspondiente grafica para poder mostrar dicha información en la aplicación web.

```
//Obtencion de los arreglos
const general=await modeloGeneral()
const hombresTipo=await modeloHombresTipo(general[1])
const mujeresTipo=await modeloMujeresTipo(general[2])
const hombresEdad=await modeloHombresEdad(general[1])
const mujeresEdad=await modeloMujeresEdad(general[2])

//DESPLIEGUE DE INFORMACION
GraficaGeneral(general)
GraficaTipoComparativa(hombresTipo,mujeresTipo)
GraficaEdadComparacion(hombresEdad,mujeresEdad)
GraficaTotalTipo(hombresTipo,mujeresTipo)
GraficaTotalEdad(hombresEdad,mujeresEdad)
```

PÁGINA INICIO

La página principal tiene una estructura simple, no tiene partes dinámicas, solo es texto y los links de los manuales junto al repositorio de GitHub. Hablaremos de lo más relevante.

Conectamos la página con la hoja de estilos “style.css” y con la biblioteca Chart.js mediante las etiquetas “link” y “script”.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Pronosticador Demográfico</title>
  <link rel="stylesheet" type="text/css" href="style.css">
  <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/3.7.0/chart.min.js"></script>
</head>
```

Creamos los contenedores “container” y “content”, “container” abarca la totalidad del contenido de la etiqueta “body” exceptuando al menú, en “content” encontramos todo el texto de la página, desde “¿Qué es esta página?” hasta los manuales y el repositorio, incluyendo los links.

```
<div class="container">
  <div class="content">
    <!-- Contenido de la página aquí -->
    <h2>¿Qué es esta página?</h2>
    <p>Es un proyecto web de código abierto, tanto para curiosos como para programadores, consiste en cinco modelos de aprendizaje automático entrenados para predecir características demográficas a partir de una población. El propósito del proyecto es servir como un recurso para cualquiera que le encuentre utilidad, como podrían ser investigadores, analistas, programadores o beneficiencia.</p>
    <p>Estos modelos fueron entrenados con datos oficiales de la INEGI y los pueden encontrar más abajo, junto al repositorio que contiene todo el código y recursos utilizados. También hay disponibles dos manuales para quien guste conocer el proyecto en profundidad, un manual de usuario para el público en general y un manual técnico para especialistas.</p>

    <h2>¿Qué es un modelo de aprendizaje automático?</h2>
    <p>El aprendizaje automático es una rama de la inteligencia artificial que se basa en la idea de que los sistemas pueden aprender de datos, identificar patrones y tomar decisiones con mínima intervención humana. En términos más técnicos, un modelo de aprendizaje automático es un algoritmo o fórmula matemática que la computadora utiliza para tomar decisiones basadas en los datos que se le proporcionan.</p>
    <p>Es importante tener en cuenta que los modelos de aprendizaje automático no son infalibles y su rendimiento puede variar dependiendo de la calidad y cantidad de los datos de entrenamiento. También pueden cometer errores, especialmente cuando se encuentran con situaciones que no se parecen a los datos con los que fueron entrenados.</p>

    <h2>Manual de usuario</h2>
    <p>El manual de usuario esta destinado a quienes desean probar el resultado final de la API con los modelos de aprendizaje entrenados<br>
    <a href="https://www.anychart.com/es/products/anychart/gallery/">Manual de usuario (PDF)</a>

    <h2>Manual técnico</h2>
    <p>El manual técnico esta pensado para programadores e investigadores, quienes desean conocer el funcionamiento de la API y los modelos<br>
    <a href="https://www.anychart.com/es/products/anychart/gallery/">Manual de técnico (PDF)</a>

    <h2>GitHub</h2>
    <p>En el siguiente repositorio encontraran el código utilizado para desarrollar la API y los modelos de aprendizaje automático.</p>
    <a href="https://www.anychart.com/es/products/anychart/gallery/">Repositorio de GitHub</a>
  </div>
  <script src="script.js"></script>
</div>
```

PÁGINA MODELOS

Hacemos la conexión con las librerías TensorFlow.js y Chart.js, también conectamos la hoja de estilos y el archivo JavaScript.

```
<head>
  <meta charset="UTF-8">
  <title>Pronosticador Demográfico</title>
  <link rel="stylesheet" type="text/css" href="style.css">
  <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
  <script src="https://cdn.jsdelivr.net/npm/chart.js@4.4.0/dist/chart.umd.min.js"></script>
  <script src="./script.js"></script>
</head>
```

Dentro de “content” encontramos la línea de tiempo.

```
<div style="padding-top: 40px;padding-bottom: 20px;">
  <label for="slider">Población en Mexico</label><br>
  <input type="range" value="126014024" id="slider" class="deslizador" min="0" max="130000000" list="anios" oninput="mostrarValor(
  <datalist id="anios" class="datalist">
    <option value="16552722" label="1930" id="1930"></option>
    <option value="19653552" label="1940"></option>
    <option value="25791017" label="1950"></option>
    <option value="34923129" label="1960"></option>
    <option value="48225238" label="1970"></option>
    <option value="66846833" label="1980"></option>
    <option value="81249645" label="1990"></option>
    <option value="91158290" label="1995"></option>
    <option value="97483412" label="2000"></option>
    <option value="103263388" label="2005"></option>
    <option value="112336538" label="2010"></option>
    <option value="126014024" label="2020"></option>
  </datalist>
  <div class="etiquetasAnios">
    <label class="anio">1930</label>
    <label class="anio">1940</label>
    <label class="anio">1950</label>
    <label class="anio">1960</label>
    <label class="anio">1970</label>
    <label class="anio">1980</label>
    <label class="anio">1990</label>
    <label class="anio">1995</label>
    <label class="anio">2000</label>
    <label class="anio">2005</label>
    <label class="anio">2010</label>
    <label class="anio">2020</label>
  </div>
</div>
```

Seguido a ello, el código del botón “Cantidad de habitantes”, activado con el evento onclick “obtenerPrediccion”

```
<div style="padding-bottom: 30px;padding-top: 30px;">
  <label for="miCasilla">Cantidad de habitantes:</label>
  <input type="text" id="txtPoblacion" class="miCasilla" value="126014024">
  <button type="button" onclick="obtenerPrediccion()" class="boton-estilo" id="btnCalcular">Predicción</button>
</div>
```

Por último, tenemos el código que invoca a las gráficas. Las gráficas son esencialmente etiquetas “canvas” que, utilizando la biblioteca Chart.js, se les da la apariencia y funciones de gráficas. La clase “div_Grafica” es CSS meramente estético, lo importante es el id, con el cual se identifican para utilizarse en el archivo JavaScript.

```
<div class="charts-container">
  <div class="div_Grafica">
    <canvas id="GraficaGeneral" style="width: 70%;"></canvas>
    <label>Esta grafica utiliza el modelo 'MDiscapacitados' el cual predice la cantidad de discapacitados en una población dada.
  </div>
  <div class="div_Grafica">
    <canvas id="GraficaEdadComparativa" style="width: 80%;"></canvas>
    <label>Esta grafica utiliza los modelos 'MHombresEdad' y 'MMujeresEdad' para comparar la cantidad de hombres y mujeres disca
      dividido por el rango de edad, en base a una cantidad de discapacitados dada.
    </label>
  </div>
  <div class="div_Grafica">
    <canvas id="GraficaTotalEdad" style="width: 50%;"></canvas>
    <label>Esta grafica utiliza los modelos 'MHombresEdad' y 'MMujeresEdad' para calcular la suma total de discapacitados en cad
  </div>
  <div class="div_Grafica">
    <canvas id="GraficaTipoComparativa" style="width: 70%;"></canvas>
    <label>Esta grafica utiliza los modelos 'MHombresTipo' y 'MMujeresTipo' para comparar la cantidad de hombres y mujeres disca
  </div>
  <div class="div_Grafica">
    <canvas id="GraficaTotalTipo" style="width: 60%;"></canvas>
    <label>Esta grafica utiliza los modelos 'MHombresTipo' y 'MMujeresTipo' para calcular la suma total de discapacitados en cad
  </div>
</div>
```

PÁGINA SOBRE NOSOTROS

El “content” empieza con la presentación de los desarrolladores.

```
<div class="container">
  <div class="content">
    <!-- Contenido de la página aquí -->
    <h2>Sobre Nosotros</h2>
    <p>Somos dos estudiantes del Instituto Tecnológico de Tijuana, estamos en nuestro 11vo semestre, en primera instancia, desarrollamos este proyecto para acreditar nuestra residencia, sin embargo, estamos genuinamente interesados en explorar las capacidades de la inteligencia artificial para el desarrollo de nuevas herramientas que repercutan en múltiples campos, por ello, trabajamos con fervor en este proyecto, con potencial en el área humanitaria, inform
  </div>
</div>
```

Termina con el código que despliega las fotos de los desarrolladores, sus nombres y correos electrónicos.

```
<div class="container-pictures">
  <div class="about-me">
    <div class="picture">
      
    </div>
    <div class="picture">
      
    </div>
  </div>
  <div class="about-me">
    <div class="description">
      <p style="float: right;margin-right: 70px;">
        Oscar Hernández Medina<br>
        oscar.hernandez18@tectijuana.edu.mx
      </p>
    </div>
    <div class="description">
      <p style="float: left;margin-left: 70px;">
        Alex Rivera Perez <br>
        alex.rivera18@tectijuana.edu.mx
      </p>
    </div>
  </div>
</div>
```

CÓDIGO CSS

Todo el código CSS está en una sola hoja de estilos, empezando por la clase que da estilo a la etiqueta “body”, definimos el tipo de letra, color del fondo, margen y relleno.

```
body {  
  font-family: Arial, sans-serif;  
  background-color: #a2d9ce;  
  margin: 0;  
  padding: 0;  
}
```

En el encabezado definimos el color de fondo ligeramente mas oscuro que el anterior, relleno de 20 pixeles y centramos el texto.

```
header {  
  background-color: #4ecca3;  
  padding: 20px;  
  text-align: center;  
}
```

Tenemos tres variaciones para el menú, primero aplicamos FlexBox, centramos el contenido horizontalmente y usamos el mismo color de fondo que el encabezado.

```
nav {  
  display: flex;  
  justify-content: center;  
  background-color: #4ecca3;  
}
```


A las etiquetas “a” dentro del menú les definimos el texto en blanco, quitamos el subrayado de los enlaces, agregamos relleno a los lados, arriba y abajo, ponemos el texto en negritas y definimos el tamaño en 20 pixeles.

```
nav a {  
    color: #fff;  
    text-decoration: none;  
    padding: 10px 20px;  
    font-weight: bold;  
    font-size: 20px;  
}
```

Oscurecemos el color de fondo cuando el cursor este sobre los botones del menú.

```
nav a:hover {  
    background-color: #34a489;  
}
```

La clase “contianier” la aplicamos en el primer contenedor, usamos FlexBox, centramos el contenido de manera horizontal y luego vertical, establecemos la altura automática, agregamos relleno arriba y abajo.

```
.container {  
    display: flex;  
    justify-content: center;  
    align-items: center;  
    height: auto;  
    margin-top: 70px;  
    margin-bottom: 70px;  
}
```

La clase “content” la aplicamos al segundo contenedor, que se encuentra dentro del primero. Centramos el texto, hacemos el fondo blanco, agregamos relleno a los cuatro lados, redondeamos las esquinas 10 pixeles y definimos el ancho al 65% del tamaño del primer contenedor.

```
.content {  
  text-align: center;  
  background-color: #fff;  
  padding: 20px;  
  border-radius: 10px;  
  width: 65%;  
}
```

La clase “charts-container” se aplica a un tercer contenedor dentro del segundo contenedor, el cual contiene las gráficas en la página “Modelos”. Aplicamos FlexBox, distribuimos el contenido uniformemente y definimos la dirección del contenido en vertical.

```
.charts-container {  
  display: flex;  
  justify-content: space-around;  
  flex-direction: column;  
}
```

Le damos estilo a todas las etiquetas de texto “p”, justificamos el texto y limitamos su anchura para que no se extienda más allá del tamaño del contenedor.

```
p {  
  text-align: justify;  
  max-width: 100%;  
}
```

Damos estilo al botón “Predicción” de la página “Modelos”, definimos el color de fondo, quitamos los bordes, hacemos el texto color negro, centramos el texto, lo definimos como un elemento de bloque en línea, cambiamos el tamaño del texto, agregamos una transición, cambiamos el cursor cuando este sobre el botón, agregamos relleno en los cuatro lados, redondeamos las esquinas y ponemos el texto en negritas.

```
.boton-estilo {  
    background-color: #4ecca3;  
    border: none;  
    color: black;  
    text-align: center;  
    display: inline-block;  
    font-size: 16px;  
    transition-duration: 0.4s;  
    cursor: pointer;  
    padding: 10px 24px;  
    border-radius: 12px;  
    font-weight: bold;  
}
```

Cuando el cursor este sobre el botón, el color de fondo se oscurece y el color del texto se vuelve blanco.

```
.boton-estilo:hover {  
    background-color: #34a489;  
    color: white;  
}
```

Es momento de darle estilo a las gráficas. Ponemos relleno arriba y abajo, aplicamos FlexBox, centramos el contenido de forma horizontal y vertical y definimos la dirección en vertical.

```
.div_Grafica{  
    padding-bottom: 30px;  
    padding-top: 25px;  
    display: flex;  
    justify-content: center;  
    align-items: center;  
    flex-direction: column;  
}
```

Para las etiquetas “label” de las gráficas, las definimos como elementos de bloque, centramos el texto, ponemos relleno arriba, inclinamos el texto, cambiamos la fuente del texto, cambiamos el tamaño del texto y definimos la anchura al 70% del tamaño del contenedor padre.

```
.div_Grafica label{  
    display: block;  
    text-align: center;  
    padding-top: 20px;  
    font-style: oblique;  
    font-family: 'Times New Roman', Times, serif;  
    font-size: 14px;  
    width: 70%;  
}
```

La clase “miCasilla” se aplica a la etiqueta “input” que corresponde a la casilla en la página “Modelos” donde se ingresa el número de habitantes. Redondeamos las esquinas, cambiamos el tamaño del texto, definimos la altura y cambiamos la fuente del texto.

```
.miCasilla{  
    border-radius: 5px;  
    font-size: 16px;  
    height: 26px;  
    font-family: 'Times New Roman', Times, serif;  
}
```

La clase “icono” se aplica a la etiqueta “img” en el menú. Cambiamos la anchura de la imagen, dejamos la altura automática y agregamos un margen a la derecha.

```
.icono{  
    width: 18px;  
    height: auto;  
    margin-right: 5px;  
}
```

La clase “deslizador” pertenece a una etiqueta “input” de tipo “range”, de esta forma creamos la línea de tiempo en la página “Modelos”. Definimos el ancho en 90% del contenedor padre.

```
.deslizador{  
    width: 90%;  
}
```

La clase “etiquetasAnios” se aplica a un contenedor con las etiquetas “label” con el texto de los años de la línea de tiempo. La posición se define como relativa.

```
.etiquetasAnios{  
    position: relative;  
}
```

La clase “anio” corresponde a las etiquetas “label” mencionadas hace un momento. Se define la posición como absoluta, se cambia el tamaño del texto y se pone en negritas.

```
.anio{  
    position: absolute;  
    font-size: 12px;  
    font-weight: bold;  
}
```

Las siguientes pseudo-classes tienen el propósito de modificar la posición de los años mostrados en la línea de tiempo, aumentando progresivamente el espacio del lado izquierdo.

```
.anio:nth-child(1){  
    left: 15%;  
}  
.anio:nth-child(2){  
    left: 18%;  
}  
.anio:nth-child(3){  
    left: 22%;  
}  
.anio:nth-child(4){  
    left: 28%;  
}  
.anio:nth-child(5){  
    left: 37%;  
}  
.anio:nth-child(6){  
    left: 50%;  
}  
.anio:nth-child(7){  
    left: 59.5%;  
}  
.anio:nth-child(8){  
    left: 66.5%;  
}  
.anio:nth-child(9){  
    left: 70.5%;  
}  
.anio:nth-child(10){  
    left: 74.5%;  
}  
.anio:nth-child(11){  
    left: 81%;  
}  
.anio:nth-child(12){  
    left: 90%;  
}
```

La clase “container-pictures” da estilo a un contenedor con las imágenes e información de los desarrolladores. Damos un margen superior, definimos el ancho al 100% del contenedor padre y la altura automática.

```
.container-pictures{  
    margin-top: 40px;  
    width: 100%;  
    height: auto;  
}
```

La clase “about-me” da estilo a dos contenedores dentro del anterior, uno con las imágenes de los desarrolladores y otro con la información. Definimos el ancho al 100% del contenedor padre, la altura automática, ocultamos un posible desbordamiento y aplicamos FlexBox.

```
.about-me{  
    width: 100%;  
    height: auto;  
    overflow: hidden;  
    display: flex;  
}
```

La clase “picture” aplica un estilo más detallado a las imágenes de los desarrolladores. Definimos el ancho en 50% del contenedor padre, redondeamos las imágenes, aplicamos un borde negro y redefinimos el ancho al 40%.

```
.picture{  
    width: 50%;  
}  
.picture img{  
    border-radius: 50%;  
    border: 2px solid black;  
    width: 40%;  
}
```

La clase “description” aplica un estilo más detallado a la información de los desarrolladores. Define el ancho en 70% del contenedor padre, cambiamos el tamaño del texto, ponemos el texto en cursiva y lo centramos.

```
.description{  
  width: 70%;  
  font-size: 16px;  
  font-family:cursive;  
}  
  
.description p{  
  text-align: center;  
}
```