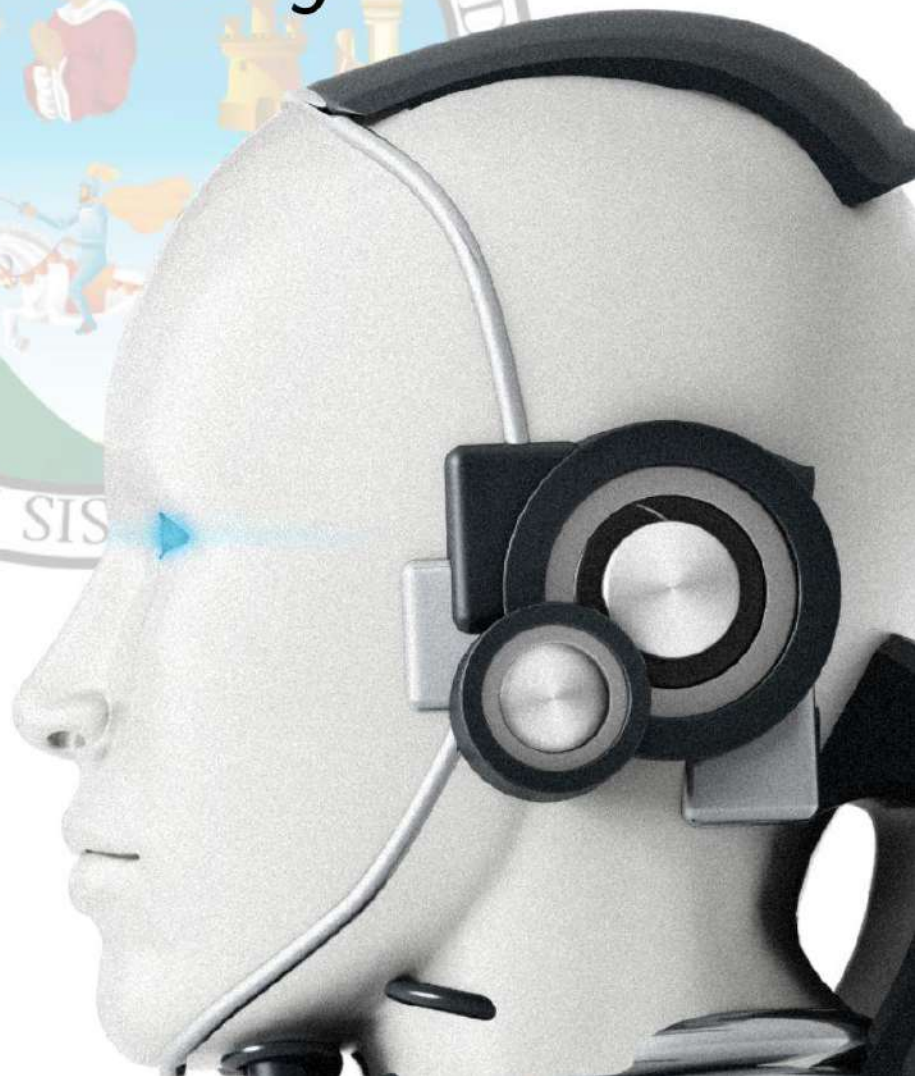
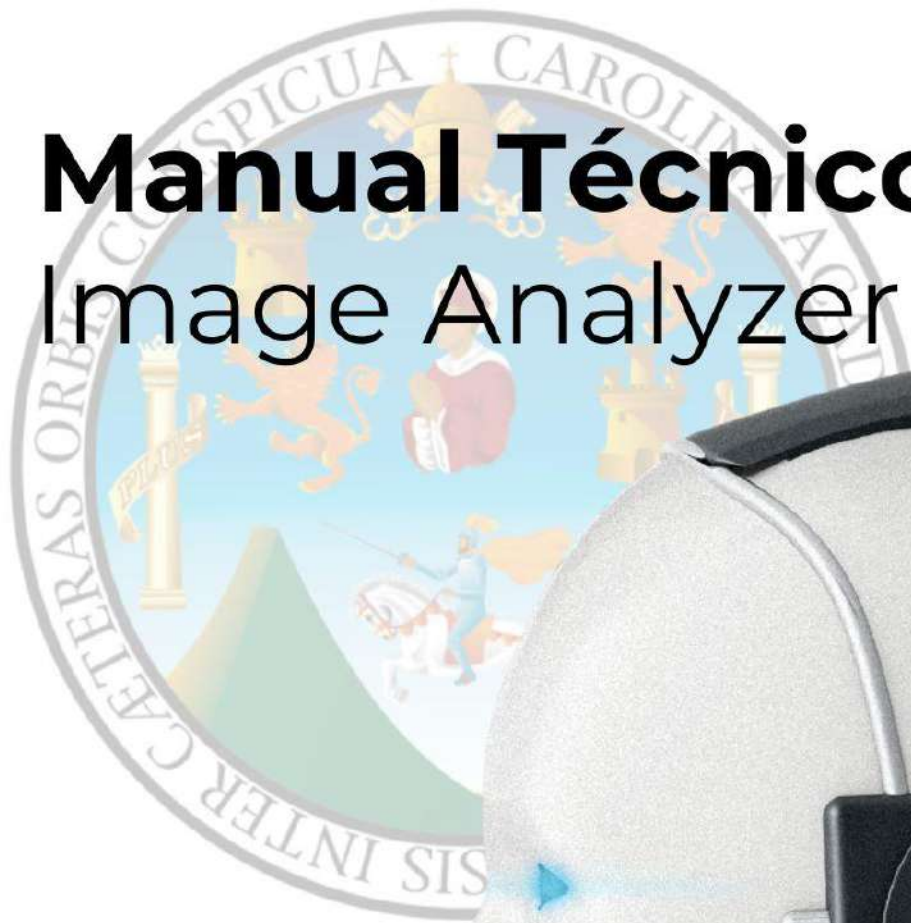


PRACTICA 1

# IA1



## Manual Técnico Image Analyzer



Rodrigo Alejandro Hernández de León

201900042

## Índice

Objetivos .....	3
Lógica de la Aplicación.....	4
Servidor .....	4
Cliente .....	8

## Objetivos

### General:

Proporcionar la documentación adecuada para el conocimiento de la implementación de la aplicación desde el código fuente y los métodos utilizados por el lado del servidor hasta los componentes utilizados desde el lado del cliente para garantizar el buen funcionamiento y también el desarrollador que lo lea pueda comprender y si desea desarrollar y/o agregar cambios a la aplicación.

### Específicos:

- Conocer el uso de la aplicación de Java con Spring Boot para llamar a la API de Google Cloud Vision y así enviar la data necesaria de lado del cliente. Con esto conocer la transformación de los datos recibidos de un análisis de una inteligencia artificial.
- Conocer la implementación del lado del Cliente para cargar imágenes y así mismo también interpretar el análisis recibido de la data del lado del Servidor para que el desarrollador comprenda como fueron expuestos los datos a los componentes.


# Lógica de la Aplicación

## Servidor

El servidor cuenta con el siguiente Endpoint importante para su flujo:

Dirección	Tipo de Método
/analizar	POST

El cual su body debe de ser de tipo *multipart/form-data* y teniendo lo siguiente:

Key	Value
<input checked="" type="checkbox"/> file	File  zombies-walking-dead.jpg 

Donde el Value es un Objeto de tipo File.

Y su respuesta es la siguiente:

```
{
  "caras": [
    {
      "cuadro": [
        {
          "x": 700,
          "y": 357
        },
        {
          "x": 756,
          "y": 357
        },
        {
          "x": 756,
          "y": 412
        },
        {
          "x": 700,
          "y": 412
        }
      ]
    }
  ],
  "cantidad_caras": 6,
  "contenido": {
    "adulto": 0,
    "parodia": 20,
    "medico": 0,
    "violencia": 60,
    "picante": 20
  }
}
```

Donde:

- Caras: Es un arreglo de tipo cuadro.
- Cuadro: Es un arreglo de coordenadas que contienen la posición x y y de los puntos del cuadrado de la cara.
- Cantidad\_caras: Es la cantidad de caras detectadas.
- Contenido: Contiene los porcentajes detectados del SAFE\_SEARCH\_DETECTION.

En java se usaron las distintas clases:

- Coordenadas: Para el almacenamiento de la posición en x y y de los puntos del cuadrante.

```
public class Coordenadas {  
    private int x;  
    private int y;  
  
    public Coordenadas(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

- Cara: Para el almacenamiento de un arreglo de 4 coordenadas.

```
public class Cara {  
    private LinkedList<Coordenadas> cuadro;  
  
    public Cara(LinkedList<Coordenadas> cuadro) {  
        this.cuadro = cuadro;  
    }  
  
    public LinkedList<Coordenadas> getCuadro() {  
        return cuadro;  
    }  
  
    public void setCuadro(LinkedList<Coordenadas> cuadro) {  
        this.cuadro = cuadro;  
    }  
}
```

- Contenido: Para el almacenamiento del porcentaje de las etiquetas de adulto, parodia, médico, violencia y picante.

```
public class Contenido {  
    private int adulto;  
    private int parodia;  
    private int medico;  
    private int violencia;  
    private int picante;  
  
    public Contenido() {  
    }  
}
```

- ResponseDataObject: Es para enviar todo el contenido de la respuesta del servidor como el arreglo de la clase Cara, la cantidad de caras y la clase Contenido.

```
public class ResponseDataObject {  
    private List<Cara> caras;  
    private int cantidad_caras;  
    private Contenido contenido;  
  
    public ResponseDataObject() {  
    }  
}
```

- ApiApplication: Esta clase contiene el método de la lógica del POST del endpoint analizar, donde también se hacen las peticiones a Google Cloud Vision y retorna el json de respuesta.

```

@RestController
@CrossOrigin(origins = "**")
@RequestMapping("/analizar")
public class VisionController {

    @PostMapping
    public ResponseEntity analyzeImage(@RequestParam("file") MultipartFile file) throws Exception {
        //Contenido
    }
}

```

Dentro de este método se hizo lo siguiente:

Se convierte lo ingresado en un objeto Image

```

//Se convierte la imagen de entrada en un byte de caracteres y pasa a
ser un objeto Image
ByteArray imgBytes = ByteArray.copyFrom(file.getBytes());
Image img = Image.newBuilder().setContent(imgBytes).build();

```

Se crean 2 Listas que serán nuestros requests

```

List<AnnotateImageRequest> requests1 = new ArrayList<>();
List<AnnotateImageRequest> requests2 = new ArrayList<>();

```

Donde el request 1 es para la detección de rostros y el request2 es para la detección de búsqueda segura.

```

Feature feat1 = Feature.newBuilder().setType(Type.FACE_DETECTION).build();
Feature feat2 = Feature.newBuilder().setType(Type.SAFE_SEARCH_DETECTION).build();
AnnotateImageRequest request1 = AnnotateImageRequest.newBuilder().addFeatures(feat1).setImage(img).build();
AnnotateImageRequest request2 = AnnotateImageRequest.newBuilder().addFeatures(feat2).setImage(img).build();
requests1.add(request1);
requests2.add(request2);

```

Luego pasa a la lógica de detección de caras donde obtenemos la data y la vamos agregando a la clase de ResponseDataObject:

```

BatchAnnotateImagesResponse response1 = vision.batchAnnotateImages(requests1);
List<AnnotateImageResponse> responses1 = response1.getResponsesList();
LinkedList<Cara> caras = new LinkedList<>();
ResponseDataObject fd = new ResponseDataObject();
for (AnnotateImageResponse res : responses1) {
    if (res.hasError()) {
        System.out.format("Error: %s\n", res.getError().getMessage());
        return new ResponseDataObject(); // Devuelve una lista vacía si hay un error
    }
    for (FaceAnnotation annotation : res.getFaceAnnotationsList()) {
        LinkedList<Coordenadas> coordenadas = new LinkedList<>();
        for (Vertex v : annotation.getFdBoundingPoly().getVerticesList()) {
            Coordenadas c = new Coordenadas(v.getX(), v.getY());
            coordenadas.add(c);
        }
        Cara cara = new Cara(coordenadas);
        caras.add(cara);
    }
}
fd.setCantidad_caras(caras.size());
fd.setCaras(caras);

```

Luego pasa a transformar la data de la búsqueda segura y la guarda en el ResponseDataObject:

```
BatchAnnotateImagesResponse response2 = vision.batchAnnotateImages(requests2);
List<AnnotateImageResponse> responses2 = response2.getResponsesList();
for (AnnotateImageResponse res : responses2) {
    if (res.hasError()) {
        System.out.format("Error: %s\n", res.getError().getMessage());
        return new ResponseDataObject(); // Devuelve una lista vacía si hay un error
    }

    SafeSearchAnnotation annotation = res.getSafeSearchAnnotation();
    Contenido c = new Contenido();
    System.out.println(annotation);
    switch (annotation.getAdult().toString()){
        case "VERY_UNLIKELY":
            c.setAdulto(0);
            break;
        case "UNLIKELY":
            c.setAdulto(20);
            break;
        case "POSSIBLE":
            c.setAdulto(35);
            break;
        case "LIKELY":
            c.setAdulto(60);
            break;
        case "VERY_LIKELY":
            c.setAdulto(80);
            break;
    }
    System.out.println(annotation.getAdult().toString());
    switch (annotation.getMedical().toString()){
        case "VERY_UNLIKELY":
            c.setMedico(0);
            break;
        case "UNLIKELY":
            c.setMedico(20);
            break;
        case "POSSIBLE":
            c.setMedico(35);
            break;
        case "LIKELY":
            c.setMedico(60);
            break;
        case "VERY_LIKELY":
            c.setMedico(80);
            break;
    }
    System.out.println(annotation.getMedical().toString());
    switch (annotation.getViolence().toString()){
        case "VERY_UNLIKELY":
            c.setViolencia(0);
            break;
        case "UNLIKELY":
            c.setViolencia(20);
            break;
        case "POSSIBLE":
            c.setViolencia(35);
            break;
        case "LIKELY":
            c.setViolencia(60);
            break;
        case "VERY_LIKELY":
            c.setViolencia(80);
            break;
    }
    System.out.println(annotation.getViolence().toString());
    switch (annotation.getSpoof().toString()){
        case "VERY_UNLIKELY":
            c.setParodia(0);
            break;
        case "UNLIKELY":
            c.setParodia(20);
            break;
        case "POSSIBLE":
            c.setParodia(35);
            break;
        case "LIKELY":
            c.setParodia(60);
            break;
        case "VERY_LIKELY":
            c.setParodia(80);
            break;
    }
    System.out.println(annotation.getSpoof().toString());
    switch (annotation.getRacy().toString()){
        case "VERY_UNLIKELY":
            c.setPicante(0);
            break;
        case "UNLIKELY":
            c.setPicante(20);
            break;
        case "POSSIBLE":
            c.setPicante(35);
            break;
        case "LIKELY":
            c.setPicante(60);
            break;
        case "VERY_LIKELY":
            c.setPicante(80);
            break;
    }
    System.out.println(annotation.getRacy().toString());
    fd.setContenido(c);
}
```



Por último, se retorna la clase para ser retornada en la solicitud:

```
return fd;
```

## Cliente

Para el lado del cliente se realizó con la librería de React con JavaScript, donde se utilizaron los siguientes archivos:

- App:

Donde contiene el componente general de la aplicación y cuenta con lo siguiente:

Navbar: Donde va la parte superior de la lógica del cliente:

```
<nav class="bg-gray-800 border-gray-200">
  <div class="max-w-screen-xl flex flex-wrap items-center justify-between mx-auto p-4">
    <div class="flex items-center space-x-3 rtl:space-x-reverse">
      <span class="self-center text-xl font-semibold whitespace-nowrap text-white">
        Práctica 1
      </span>
    </div>
    <div class="flex items-center space-x-3 rtl:space-x-reverse flex-grow justify-center">
      <span class="self-center text-2xl font-semibold whitespace-nowrap text-white">
        Image-Analyzer
      </span>
    </div>
    <div class="flex items-center space-x-3 rtl:space-x-reverse">
      <span class="self-center text-xl font-semibold whitespace-nowrap text-white">
        Inteligencia Artificial 1
      </span>
    </div>
  </div>
</nav>
```

Posteriormente cuenta con 2 divs donde 1 cuenta con la información de la imagen y la carga de la imagen junto con el botón de analizar imagen. Y el otro div para mostrar la imagen. Contiene los siguientes métodos:

Cambia la forma de la imagen con el difuminado y face detection:

```
useEffect(() => {
  if (imageSrc) {
    const canvas = canvasRef.current;
    const ctx = canvas.getContext('2d');
    const img = new Image();
    img.src = imageSrc;
    img.onload = () => {
      canvas.width = img.width;
      canvas.height = img.height;
      ctx.drawImage(img, 0, 0);
      if (blur) {
        ctx.filter = "blur(5px)";
        ctx.drawImage(canvas, 0, 0);
        ctx.filter = "none";
      }
      caras.forEach(cara => {
        const cuadro = cara.cuadro;
        ctx.strokeStyle = 'green';
        ctx.lineWidth = 2;
        ctx.beginPath();
        ctx.moveTo(cuadro[0].x, cuadro[0].y);
        for (let i = 1; i < 4; i++) {
          ctx.lineTo(cuadro[i].x, cuadro[i].y);
        }
        ctx.closePath();
        ctx.stroke();
      });
    };
  }
}, [imageSrc, blur, caras]);
```



Analizar imagen para enviar la solicitud de la imagen y procesar los datos recibidos:

```
const AnalizarImagen = () => {
  if(imagen == null){
    toast.error('No has seleccionado una imagen');
    return;
  }
  const fd = new FormData();
  fd.append('file', imagen);
  Service.analizar(fd)
  .then((response) => {
    console.log(response);
    const data = response.data;
    setRostros(data.cantidad_caras);
    const contenido = data.contenido;
    setAdulto(contenido.adulto);
    setParodia(contenido.parodia);
    setMedico(contenido.medico);
    setViolencia(contenido.violencia);
    setPicante(contenido.picante);
    if(contenido.violencia > 59){
      setBlur(true);
    }else if(contenido.picante > 50){
      setBlur(true);
    }else if(contenido.adulto > 40){
      setBlur(true);
    }else{
      setBlur(false);
    }
    const sumatoria = contenido.violencia + contenido.picante + contenido.adulto;
    if(sumatoria > 45){
      setMensaje(true);
      setAceptado(false);
    }else{
      setMensaje(true);
      setAceptado(true);
    }
    const caras = data.caras;
    setCaras(caras);
  })
  .catch((error) => {
    console.log(error);
  });
}
```

FileToBase64: para pasar la imagen a base64 para mostrarla

```
function fileToBase64(file) {
  return new Promise((resolve, reject) => {
    const reader = new FileReader();
    reader.readAsDataURL(file);
    reader.onload = () => resolve(reader.result);
    reader.onerror = error => reject(error);
  });
}
```

handleFile: Muestra la imagen y resetea los datos cuando se carga una nueva imagen

```

const handleFile = (file) => {
  const allowedTypes = ['image/png', 'image/jpeg'];
  if(!file){
    toast.error('Hubo un error al cargar la imagen');
    return;
  };
  if (allowedTypes.includes(file.type)) {
    setNameFile(file.name);
    setImagen(file);
    console.log('Archivo cargado:', file);
    setBlur(false);
    setCaras([]);
    setAdulto(0);
    setParodia(0);
    setMedico(0);
    setViolencia(0);
    setPicante(0);
    setNostrao(0);
    setMensaje(false);
    fileToBase64(file)
      .then((result) => {
        setImageSrc(result);
      })
  } else {
    toast.error('El archivo seleccionado no es válido. Por favor, selecciona un archivo de tipo .png o .jpg.', {
      style: {
        borderRadius: '18px',
        background: '#333',
        color: 'fff',
      },
    });
  }
}

```

- Service: Se encarga de armar un tipo Singleton para que sea más fácil de llamar a los métodos de petición al servidor

```

import * as petition from './Connection/comapi';

export default {
  ...petition
}

```

- Comapi: Se encarga de la comunicación con el servidor a conectarse con los endpoints.

```

import axios from 'axios';

const instance = axios.create({
  baseURL: 'http://localhost:4000/'
});

export const analizar = async (fd) => {
  console.log([...fd]);
  const response = await instance.post('/analizar', fd, {
    headers: {
      'Content-Type': 'multipart/form-data'
    }
  });
  return response;
}

```