PRÁCTICA 1

MANUAL TÉCNICO

PROGRAMA REALIZADO POR EL ALUMNO ALLEN GIANKARLO ROMÁN VÁSQUEZ CARNET 202004745 PARA EL CURSO INTELIGENCIA ARTIFICIAL 1 DE LA ESCUELA DE CIENCIAS Y SISTEMAS DE LA FACULTAD DE INGENIERÍA DE LA UNIVERSIDAD SAN CARLOS DE GUATEMALA

CONTENIDO

Objetivos

Objetivo General

- Desarrollar un sistema de análisis de imágenes para la Facultad e Ingeniería que garantice la seguridad de imágenes cargadas al portal estudiantil, aplicando filtros para evaluarlas en tiempo real.

Objetivos Específicos

- Crear un entorno web interactivo que permita cargar imágenes y visualizarlas dentro del sistema.
- Desarrollar dos aspectos de visualización: uno para mostrar información sobre los rostros identificados y otro para detallar el tipo de contenido.
- Establecer un sistema de filtros para asegurar la seguridad de las imágenes.

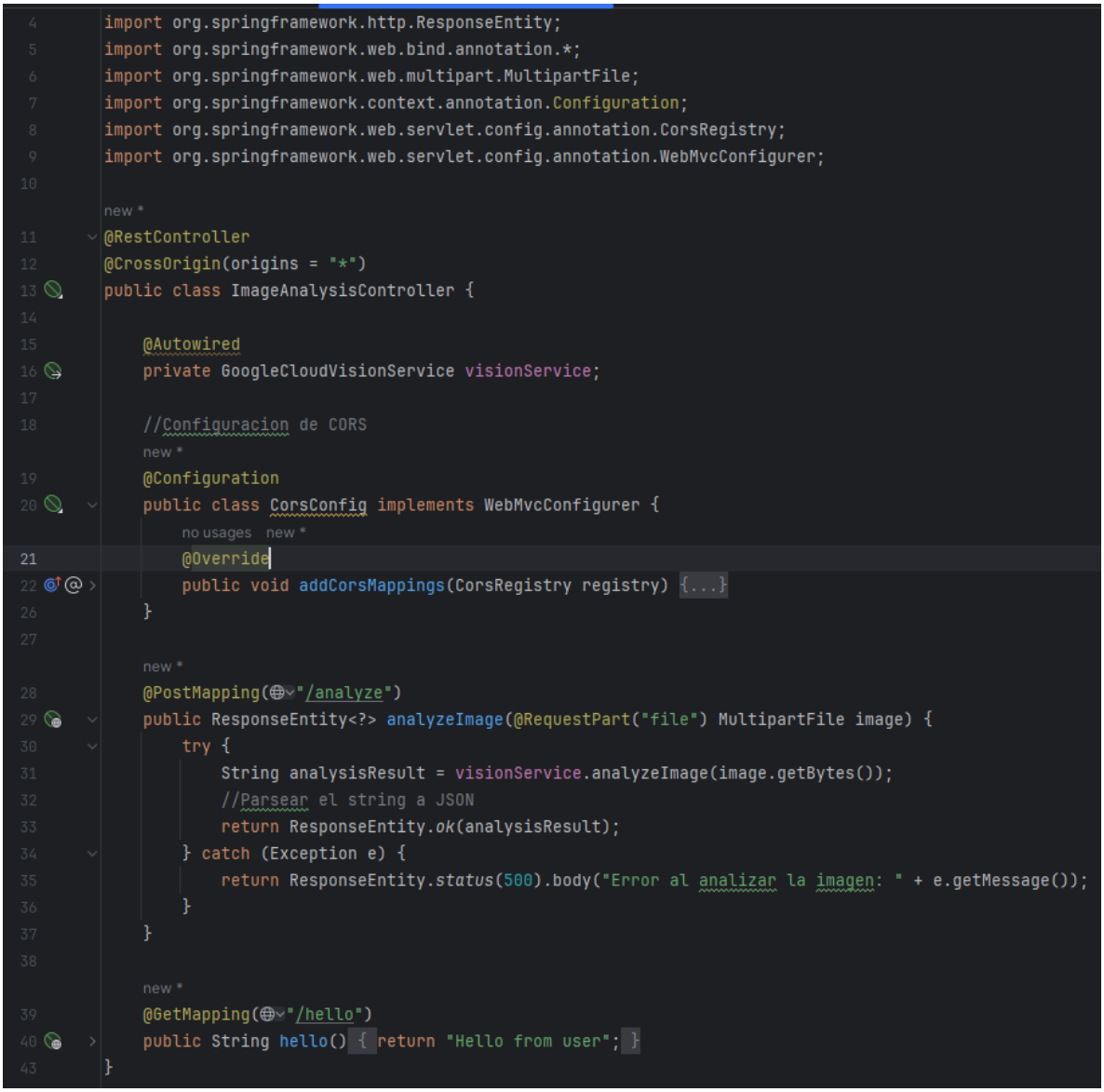
Descripción de las clases (JAVA)

Ia1P1202004745Application.java

```java
package org.example.ia1_p1_202004745;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

new *
@SpringBootApplication
public class Ia1P1202004745Application {

    new *
    public static void main(String[] args) {
        SpringApplication.run(Ia1P1202004745Application.class, args);
    }

}
```

Configuración principal de una aplicación Spring Boot. La anotación @SpringBootApplication indica que esta clase es la configuración de la aplicación Spring Boot. El método main() inicia la aplicación Spring Boot al llamar a SpringApplication.run() con la clase principal.

ImageAnalysisController.java



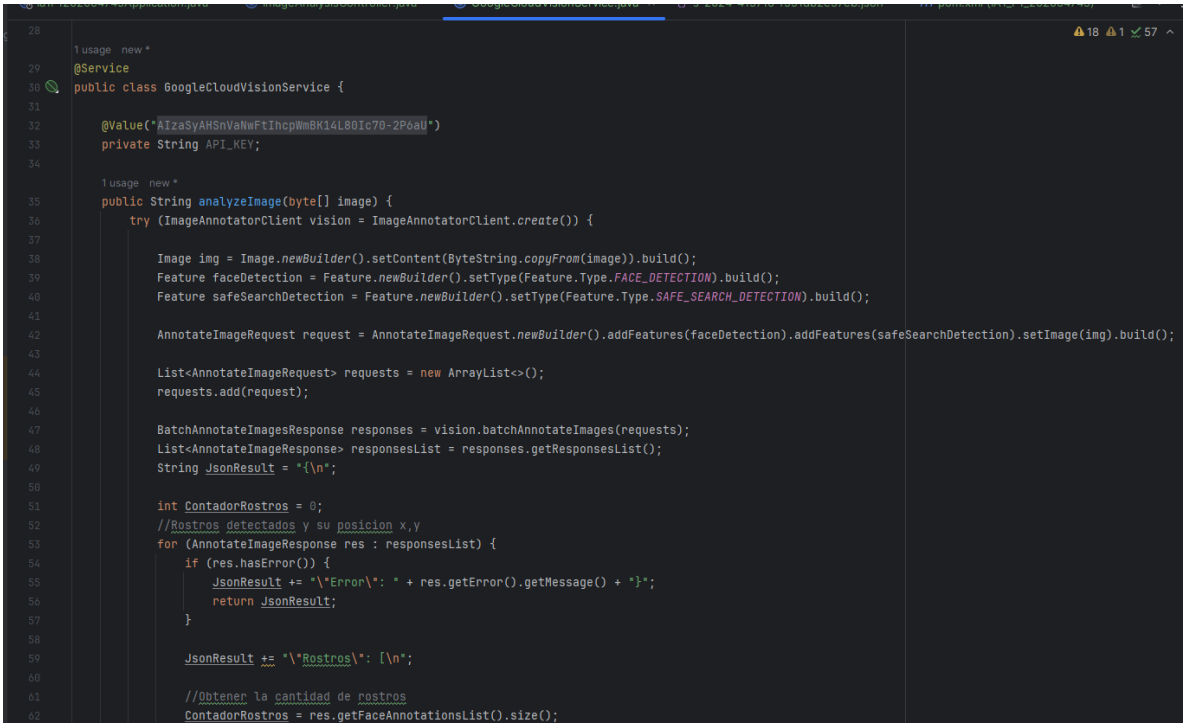
```java
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

new *
@RestController
@CrossOrigin(origins = "*")
public class ImageAnalysisController {

    @Autowired
    private GoogleCloudVisionService visionService;

    //Configuracion de CORS
    new *
    @Configuration
    public class CorsConfig implements WebMvcConfigurer {
        no usages  new *
        @Override
        public void addCorsMappings(CorsRegistry registry) {...}
    }

    new *
    @PostMapping("/analyze")
    public ResponseEntity<?> analyzeImage(@RequestPart("file") MultipartFile image) {
        try {
            String analysisResult = visionService.analyzeImage(image.getBytes());
            //Parsear el string a JSON
            return ResponseEntity.ok(analysisResult);
        } catch (Exception e) {
            return ResponseEntity.status(500).body("Error al analizar la imagen: " + e.getMessage());
        }
    }

    new *
    @GetMapping("/hello")
    public String hello() { return "Hello from user"; }
}
```

Define un controlador REST que gestiona las solicitudes relacionadas con el análisis de imágenes. El controlador utiliza Spring Framework y CORS para habilitar el intercambio de recursos entre diferentes orígenes.

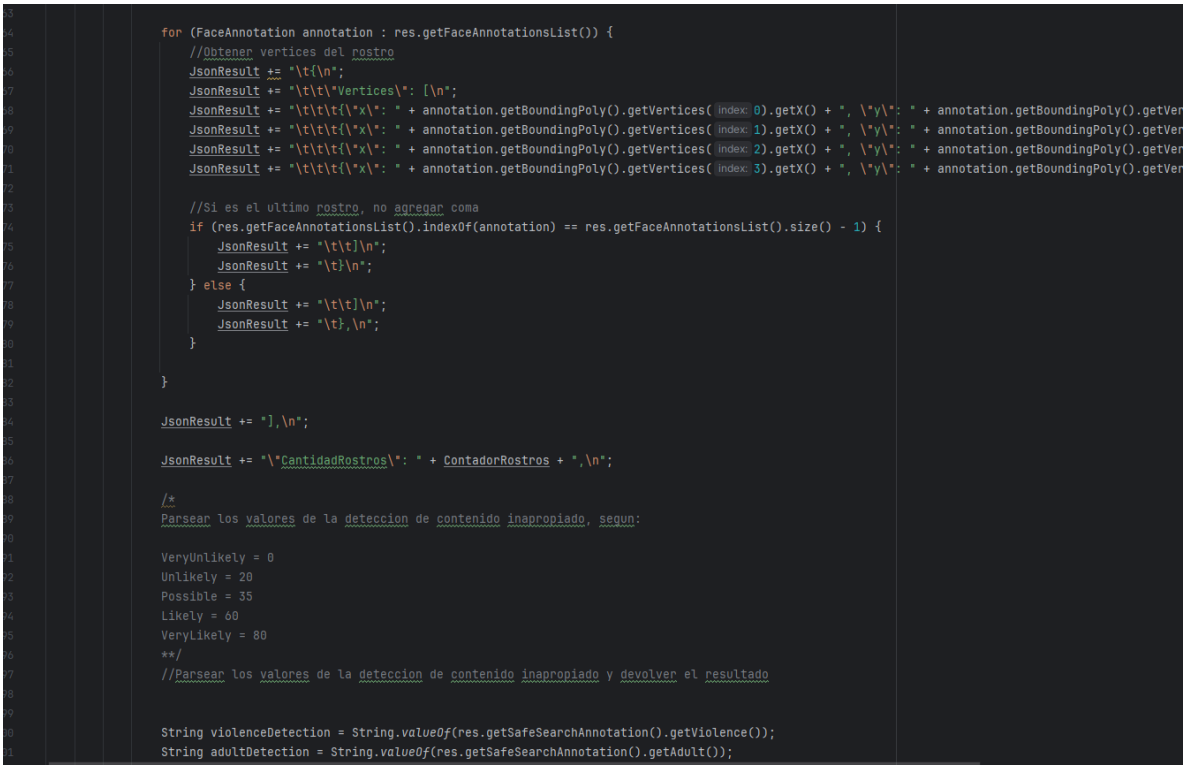
El primer endpoint @PostMapping("/analyze") recibe una imagen multipart y utiliza el servicio visionService para analizarla con Google Cloud Vision API. Luego, devuelve la respuesta del análisis como un JSON.

El segundo endpoint @GetMapping("/hello") simplemente devuelve un saludo "Hello from user" cuando se accede a través de una solicitud GET y fue utilizado para verificar que esté levantado el servidor.

GoogleCloudVisionService.java



```java
1 usage  new *
@Service
public class GoogleCloudVisionService {

    @Value("AIzaSyAHSnVaNwFtIhcpWmBK14L80Ic70-2P6aU")
    private String API_KEY;

    1 usage  new *
    public String analyzeImage(byte[] image) {
        try (ImageAnnotatorClient vision = ImageAnnotatorClient.create()) {

            Image img = Image.newBuilder().setContent(ByteString.copyFrom(image)).build();
            Feature faceDetection = Feature.newBuilder().setType(Feature.Type.FACE_DETECTION).build();
            Feature safeSearchDetection = Feature.newBuilder().setType(Feature.Type.SAFE_SEARCH_DETECTION).build();

            AnnotateImageRequest request = AnnotateImageRequest.newBuilder().addFeatures(faceDetection).addFeatures(safeSearchDetection).setImage(img).build();

            List<AnnotateImageRequest> requests = new ArrayList<>();
            requests.add(request);

            BatchAnnotateImagesResponse responses = vision.batchAnnotateImages(requests);
            List<AnnotateImageResponse> responsesList = responses.getResponsesList();
            String JsonResult = "{\n";

            int ContadorRostros = 0;
            //Rostros detectados y su posicion x,y
            for (AnnotateImageResponse res : responsesList) {
                if (res.hasError()) {
                    JsonResult += "\"Error\": " + res.getError().getMessage() + "}";
                    return JsonResult;
                }

                JsonResult += "\"Rostros\": [\n";

                //Obtener la cantidad de rostros
                ContadorRostros = res.getFaceAnnotationsList().size();
```



```java
            for (FaceAnnotation annotation : res.getFaceAnnotationsList()) {
                //Obtener vertices del rostro
                JsonResult += "\t{\n";
                JsonResult += "\t\t\"Vertices\": [\n";
                JsonResult += "\t\t\t{\"x\": " + annotation.getBoundingPoly().getVertices( index: 0).getX() + ", \"y\": " + annotation.getBoundingPoly().getVer
                JsonResult += "\t\t\t{\"x\": " + annotation.getBoundingPoly().getVertices( index: 1).getX() + ", \"y\": " + annotation.getBoundingPoly().getVer
                JsonResult += "\t\t\t{\"x\": " + annotation.getBoundingPoly().getVertices( index: 2).getX() + ", \"y\": " + annotation.getBoundingPoly().getVer
                JsonResult += "\t\t\t{\"x\": " + annotation.getBoundingPoly().getVertices( index: 3).getX() + ", \"y\": " + annotation.getBoundingPoly().getVer

                //Si es el ultimo rostro, no agregar coma
                if (res.getFaceAnnotationsList().indexOf(annotation) == res.getFaceAnnotationsList().size() - 1) {
                    JsonResult += "\t\t]\n";
                    JsonResult += "\t}\n";
                } else {
                    JsonResult += "\t\t]\n";
                    JsonResult += "\t},\n";
                }

            }

            JsonResult += "],\n";

            JsonResult += "\"CantidadRostros\": " + ContadorRostros + ",\n";

            /*
            Parsear los valores de la deteccion de contenido inapropiado, segun:

            VeryUnlikely = 0
            Unlikely = 20
            Possible = 35
            Likely = 60
            VeryLikely = 80
            **/
            //Parsear los valores de la deteccion de contenido inapropiado y devolver el resultado


            String violenceDetection = String.valueOf(res.getSafeSearchAnnotation().getViolence());
            String adultDetection = String.valueOf(res.getSafeSearchAnnotation().getAdult());
```

```java
String violenceDetection = String.valueOf(res.getSafeSearchAnnotation().getViolence());
String adultDetection = String.valueOf(res.getSafeSearchAnnotation().getAdult());
String spoofDetection = String.valueOf(res.getSafeSearchAnnotation().getSpoof());
String medicalDetection = String.valueOf(res.getSafeSearchAnnotation().getMedical());
String racyDetection = String.valueOf(res.getSafeSearchAnnotation().getRacy());

//Agregar los string a una lista y correrla para parsear los valores
List<String> detections = new ArrayList<>();
List<Integer> detectionsParse = new ArrayList<>();
detections.add(violenceDetection);
detections.add(adultDetection);
detections.add(spoofDetection);
detections.add(medicalDetection);
detections.add(racyDetection);

//Iterar la lista y parsear los valores
for (String detection : detections) {
    if (detection.equals("VERY_UNLIKELY")) {
        detectionsParse.add(0);
    } else if (detection.equals("UNLIKELY")) {
        detectionsParse.add(20);
    } else if (detection.equals("POSSIBLE")) {
        detectionsParse.add(35);
    } else if (detection.equals("LIKELY")) {
        detectionsParse.add(60);
    } else if (detection.equals("VERY_LIKELY")) {
        detectionsParse.add(80);
    }
}
```

```java
//Agregar los valores parseados a un string
JsonResult += "\"Violencia\": " + detectionsParse.get(0) + ",\n";
JsonResult += "\"Adulto\": " + detectionsParse.get(1) + ",\n";
JsonResult += "\"Spoof\": " + detectionsParse.get(2) + ",\n";
JsonResult += "\"Medico\": " + detectionsParse.get(3) + ",\n";
JsonResult += "\"Racy\": " + detectionsParse.get(4) + ",\n";

//Sumar valores violencia, adulto y picante
int suma = detectionsParse.get(0) + detectionsParse.get(1) + detectionsParse.get(4);

//Si la suma es mayor a 45, devolver que la imagen es inapropiada
if (suma > 45) {
    JsonResult += "\"Resultado\": \"Imagen inapropiada\"\n";
} else {
    JsonResult += "\"Resultado\": \"Imagen apropiada\"\n";
}

JsonResult += "}";

}
return JsonResult;
} catch (Exception  e) {
return "Error al procesar la imagen" + e.getMessage();
}
}
}
```

Esta clase GoogleCloudVisionService, utiliza la biblioteca de Google Cloud Vision API para analizar imágenes. Convierte la imagen a bytes, realiza solicitudes a la API de visión para detectar rostros y evaluar la seguridad de la imagen. Luego, formatea los resultados en un JSON que incluye la cantidad de rostros detectados, la evaluación de seguridad, incluyendo la detección de contenido inapropiado como violencia, contenido adulto y contenido picante. Finalmente, determina si la imagen es adecuada o inapropiada para la Facultad de Ingeniería.

Descripción de los componente y métodos (REACT)

app.jsx

```jsx
import './App.css'
import React, { useState } from 'react';
import Blur from 'react-blur';
import axios from 'axios';
import { FaCheck, FaTimes } from 'react-icons/fa';

function App() {
  const [image, setImage] = useState(null);
  const [faces, setFaces] = useState([]);
  const [isAppropriate, setIsAppropriate] = useState(true);
  const [loading, setLoading] = useState(false);
  const [file, setFile] = useState(null);
  const [imageInfo, setImageInfo] = useState({
    Violencia: 0,
    Adulto: 0,
    Spoof: 0,
    Medico: 0,
    Racy: 0,
    CantidadRostros: 0
  });

  const handleImageUpload = (e) => {
    const carga = e.target.files[0];
    if (carga) {
      setImage(URL.createObjectURL(carga));
    }
    setFile(carga);
    setFaces([]);
    setIsAppropriate(true);
  };

  const handleSendRequest = async () => {
    setLoading(true);
    if (file) {
      const formData = new FormData();
      formData.append('file', file);
      try {
        const response = await axios.post('http://localhost:8080/analyze', formData, {
          headers: {
            'Content-Type': 'multipart/form-data'
          }
        });
        const { Rostros, Resultado, ...info } = response.data;
        setFaces(Rostros);
        setIsAppropriate(Resultado === 'Imagen apropiada');
        setImageInfo(info);
      } catch (error) {
        console.error('Error:', error);
        // Aquí puedes mostrar un mensaje de error al usuario
      } finally {
        setLoading(false);
      }
    }
  };

  const renderValidIcon = () => {
    return isAppropriate ? <FaCheck style={{ color: 'green', marginRight: '5px' }} /> : <FaTimes style={{ color: 'red', marginRight: '5px' }} />;
  };
```

useState: Se utiliza el hook useState para manejar el estado de la aplicación. Por ejemplo, se utiliza para almacenar la imagen cargada, la información de los rostros detectados, si la imagen es apropiada o no, etc.

handleImageUpload: Este método se activa cuando un usuario carga una imagen. Obtiene el archivo de la carga, actualiza el estado de la imagen y restablece los resultados del análisis de imagen previos.

handleSendRequest: Se activa cuando un usuario hace clic en el botón "Enviar". Envía la imagen al servidor para su análisis utilizando axios para realizar una solicitud POST a la ruta '/analyze'. Luego, actualiza el estado de la aplicación con los resultados del análisis.

renderValidIcon: Renderiza un icono verde de verificación (FaCheck) si la imagen es apropiada y un icono rojo de "X" (FaTimes) si la imagen no es apropiada.

```
  const renderProgressBar = (percentage) => {
    const barColor = percentage > 50 ? 'red' : 'green';

    return (
      <div style={{ backgroundColor: '#ccc', height: '20px', width: '100%', borderRadius: '5px', marginTop: '5px' }}>
        <div style={{ backgroundColor: barColor, height: '100%', width: `${percentage}%`, borderRadius: '5px' }}></div>
      </div>
    );
  };

  return (
    <div>
      <h1>Detección de Rostros</h1>
      <input type="file" accept="image/*" onChange={handleImageUpload} />
      {image && (
        <div style={{ display: 'flex' }}>
          <div style={{ position: 'relative' }}>
            {isAppropriate ? (
              <img src={image} alt="Uploaded" />
            ) : (
              <img src={image} alt="Uploaded" style={{ filter: "blur(8px)" }}/>

            )}
            {faces.map((face, index) => (
              <div
                key={index}
                style={{
                  position: 'absolute',
                  border: '2px solid #00ff00',
                  left: face.Vertices[0].x,
                  top: face.Vertices[0].y,
                  width: face.Vertices[1].x - face.Vertices[0].x,
                  height: face.Vertices[2].y - face.Vertices[1].y
                }}
              ></div>
            ))}
          </div>
          <div style={{ marginLeft: '20px' }}>
            <div><h2>{renderValidIcon()}Imagen {isAppropriate ? 'válida' : 'no válida'}</h2></div>
            <div><h3>Cantidad de rostros: {imageInfo.CantidadRostros}</h3></div>
            <div><h2>Información de la imagen</h2></div>
            <div><h4>Violencia: {imageInfo.Violencia}% {renderProgressBar(imageInfo.Violencia)}</h4></div>
            <div><h4>Adulto: {imageInfo.Adulto}% {renderProgressBar(imageInfo.Adulto)}</h4></div>
            <div><h4>Parodia: {imageInfo.Spoof}% {renderProgressBar(imageInfo.Spoof)}</h4></div>
            <div><h4>Médico: {imageInfo.Medico}% {renderProgressBar(imageInfo.Medico)}</h4></div>
            <div><h4>Caliente: {imageInfo.Racy}% {renderProgressBar(imageInfo.Racy)}</h4></div>
          </div>
        </div>
      )}
      <button onClick={handleSendRequest} disabled={!image || loading}>
        {loading ? 'Cargando...' : 'Enviar'}
      </button>
    </div>
  );
}

export default App;
```

renderProgressBar: Renderiza una barra de progreso que indica el porcentaje de detección de diferentes tipos de contenido inapropiado en la imagen, como violencia, contenido adulto, etc.

Imagenes y estilos: Se utilizan etiquetas <img> para mostrar la imagen cargada y se aplican estilos CSS para aplicar el filtro de desenfoque (blur) a la imagen si no es apropiada y para resaltar los rostros detectados con un borde verde.