

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERIA
ESCUELA DE CIENCIAS Y SISTEMAS
INTELIGENCIA ARTIFICIAL 1, SECCIÓN A

MANUAL TECNICO

PRACTICA 1 – IA1



Jhonathan Daniel Tocay Cotzoyay | 201801268



INTRODUCCIÓN

En la actualidad, el uso de la Inteligencia Artificial se ha vuelto indispensable. Adaptarse a estas innovadoras herramientas y aprovecharlas de manera estratégica es muy importante, ya que ofrecen numerosas ventajas. En este contexto, la implementación de Vision AI se presenta como una solución relevante para el análisis de imágenes en nuestro proyecto.

La aplicación desea implementar Vision AI para el análisis de imágenes, y detectar si son aptas para la institución o no, además de esto la herramienta detecta cuantos rostros existen en dicha imagen y donde se ubican.

Esta herramienta también retorna valores para algunas características como el tipo de contenido, es decir si es contenido adulto, violento, picante, entre otros.



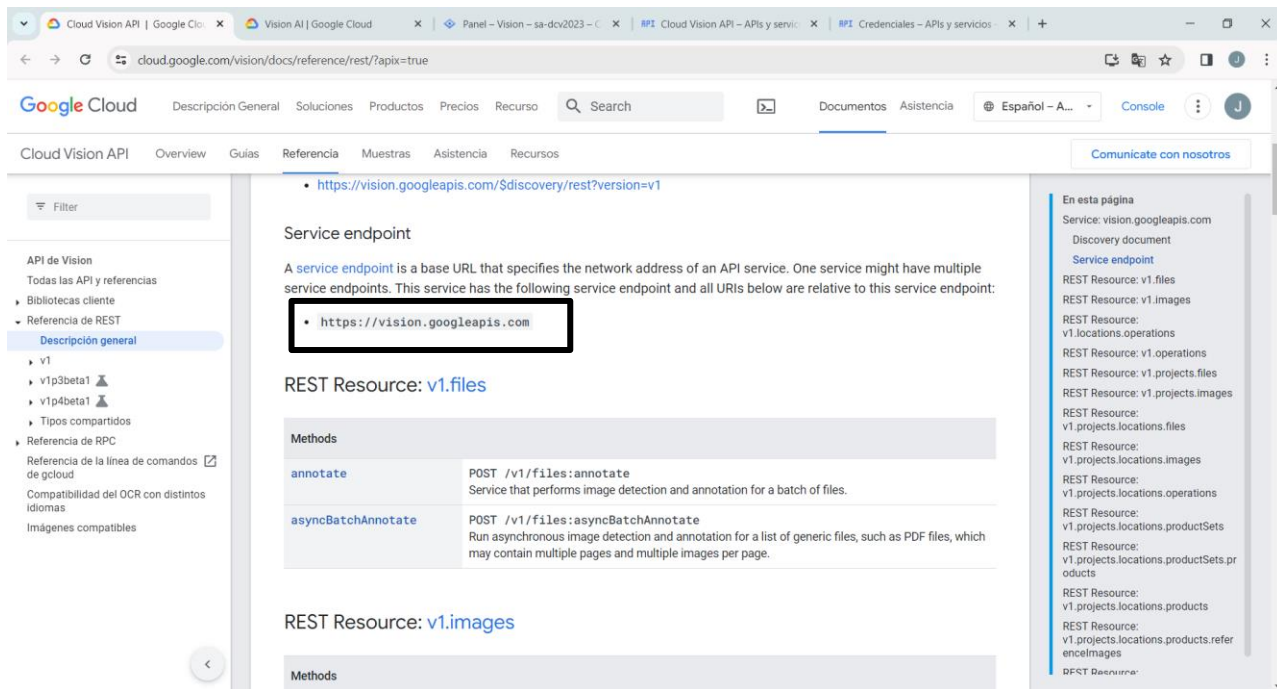
OBJETIVOS

- Implementar herramientas de IA para el analisis de las imágenes, en nuestro caso Vision AI de GCP.
- Proponer soluciones para resolver las necesidades que se solicitan en la practica.
- Implementar tecnologías actuales como Spring Boot y Angular o React, dado a sus características y versatilidad.

Configuración de Vision API

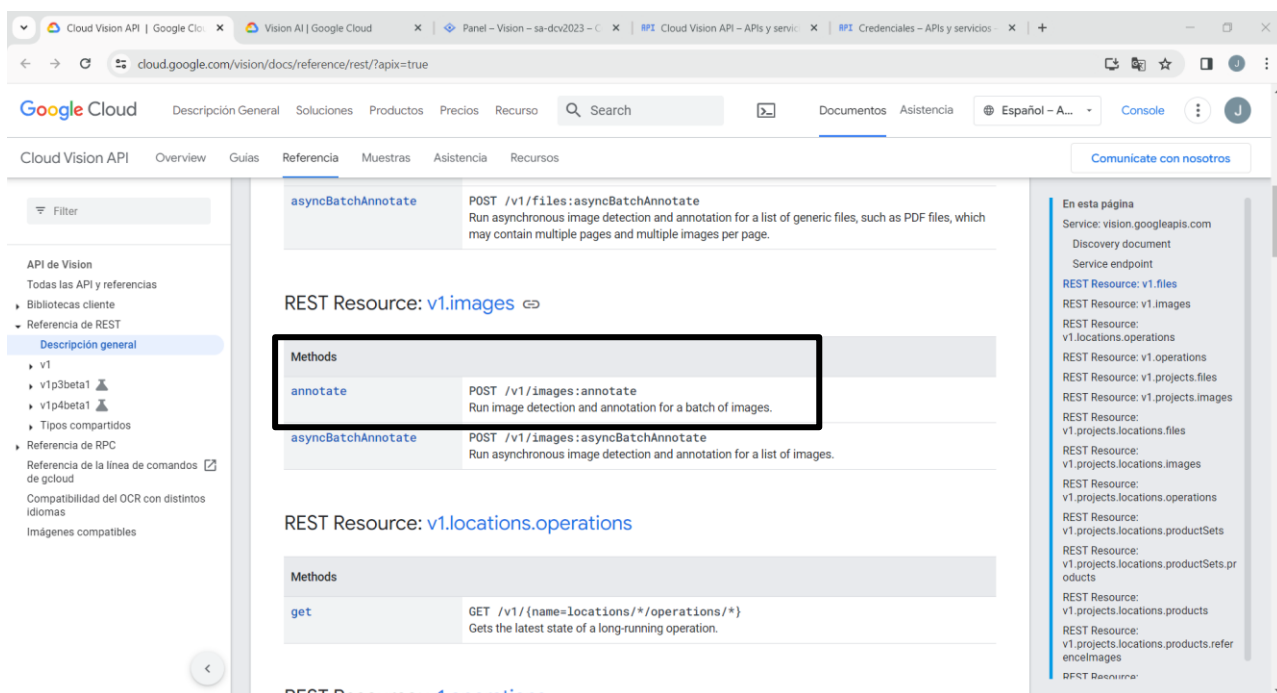
De acuerdo con la documentación de Vision AI el análisis puede realizarse utilizando la API REST el endpoint para el servicio es :

Service endpoint: <https://vision.googleapis.com>



The screenshot shows the Google Cloud Vision API documentation page. The service endpoint is highlighted as <https://vision.googleapis.com>. The REST Resource [v1.files](#) is expanded, showing two methods: **annotate** (POST /v1/files:annotate) and **asyncBatchAnnotate** (POST /v1/files:asyncBatchAnnotate). The **annotate** method is highlighted with a red box. The REST Resource [v1.images](#) is also visible below.

El método que utilizaremos será el de **Annotate**, esta petición es de Tipo **POST** y la ruta es **/v1/images:annotate**



The screenshot shows the Google Cloud Vision API documentation page. The REST Resource [v1.images](#) is expanded, showing two methods: **annotate** (POST /v1/images:annotate) and **asyncBatchAnnotate** (POST /v1/images:asyncBatchAnnotate). The **annotate** method is highlighted with a red box. The REST Resource [v1.locations.operations](#) is also visible below.

Referencia: <https://cloud.google.com/vision/docs/reference/rest/?apix=true>

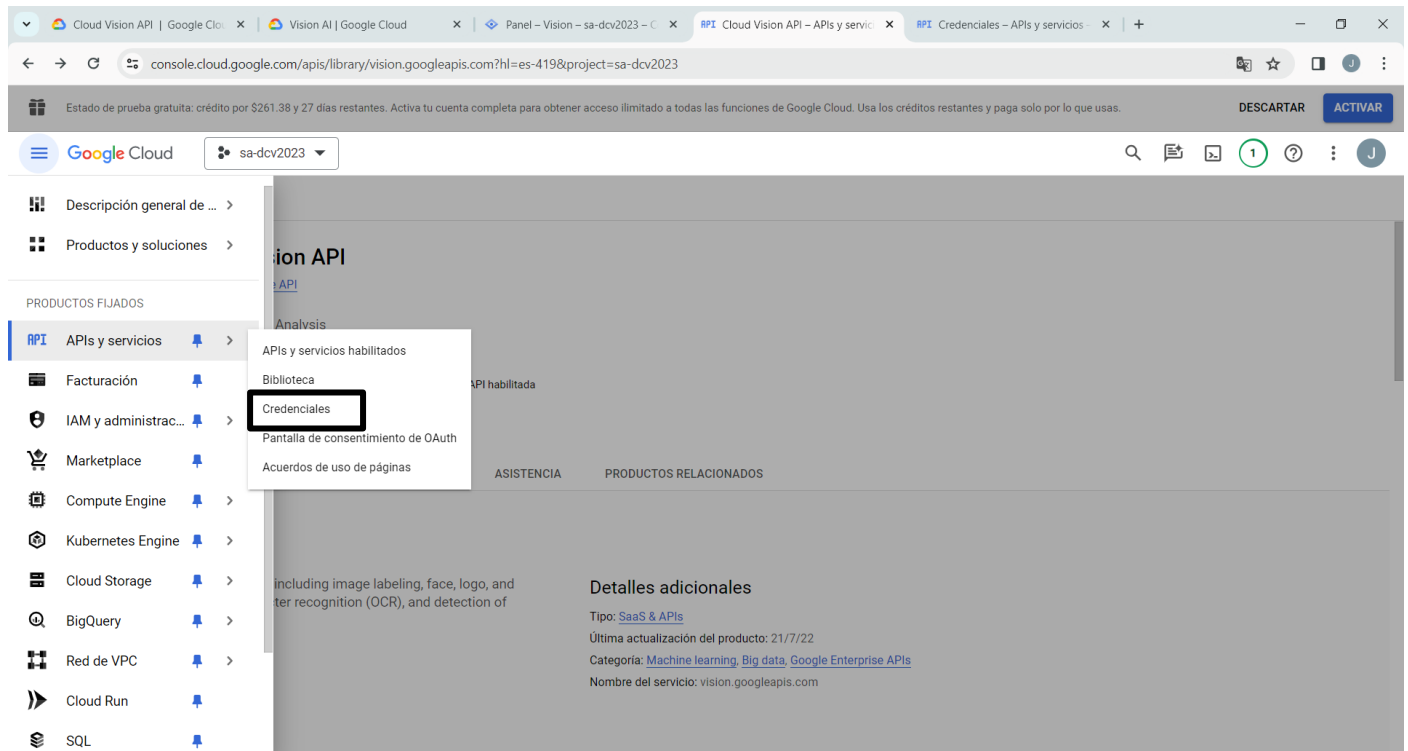
A continuacion debemos de configurar las credenciales con las cuales se nos otorgara permisos para comunicarnos con la API de Vision AI.

Puntos a Destacar

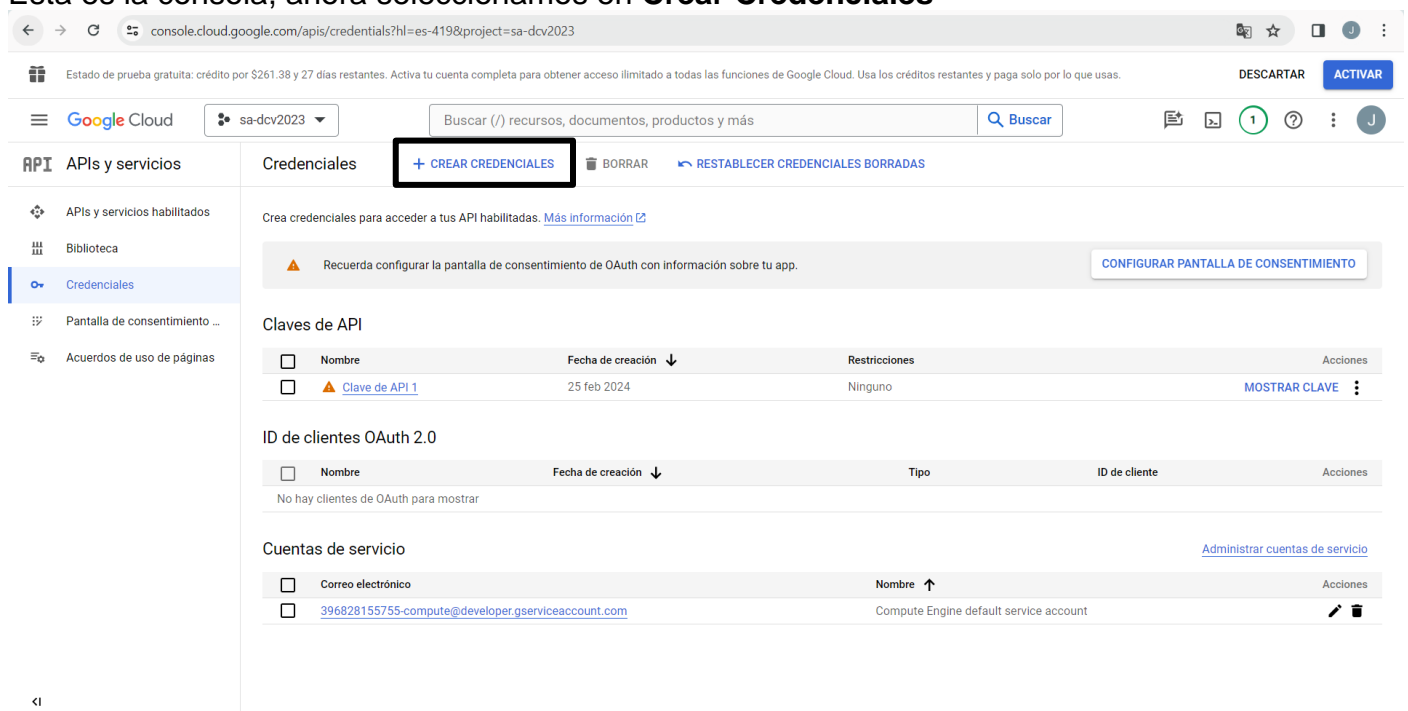
- Debemos de contar con una cuenta de Google Cloud Plataform
- Generar un Proyecto.
- Habilitar la facturacion para que podamos utilizar la Capa Gratuita y los servicios de GCP.

Pasos.

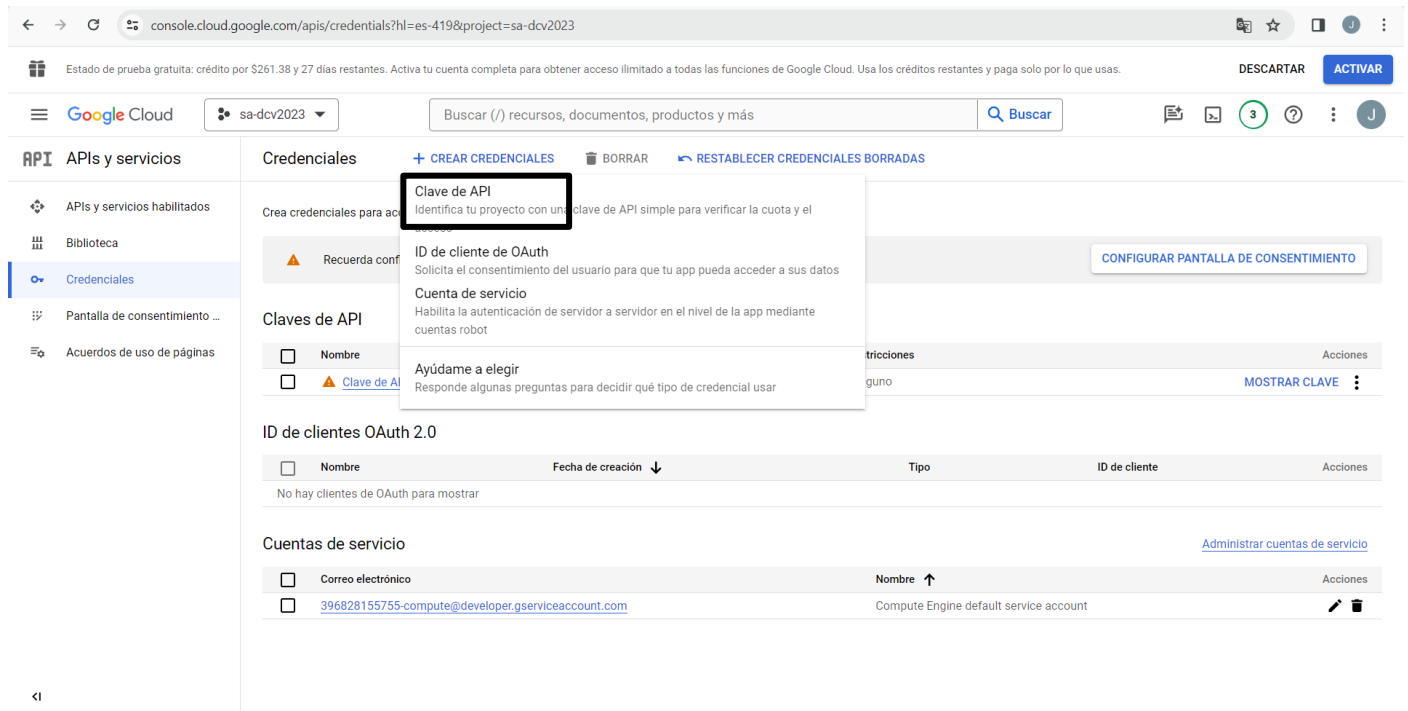
1. Debemos de Ir hacia el menu de la consula
2. Buscar el servicio de APIS y Servicios
3. Seleccionar Credenciales.



Esta es la consola, ahora seleccionamos en **Crear Credenciales**



Seleccionamos la opcion de Clave de API



Estado de prueba gratuita: crédito por \$261.38 y 27 días restantes. Activa tu cuenta completa para obtener acceso ilimitado a todas las funciones de Google Cloud. Usa los créditos restantes y paga solo por lo que usas. [DESCARTAR](#) [ACTIVAR](#)

Google Cloud sa-dcv2023 Buscar (/) recursos, documentos, productos y más

APIs y servicios

- APIs y servicios habilitados
- Biblioteca
- Credenciales**
- Pantalla de consentimiento ...
- Acuerdos de uso de páginas

Credenciales + CREAR CREDENCIALES BORRAR RESTABLECER CREDENCIALES BORRADAS

Crea credenciales para acceder a tus API habilitadas. [Más información](#)

Recuerda configurar la pantalla de consentimiento

Claves de API

- ☐ Nombre
- ☒ **Clave de API**

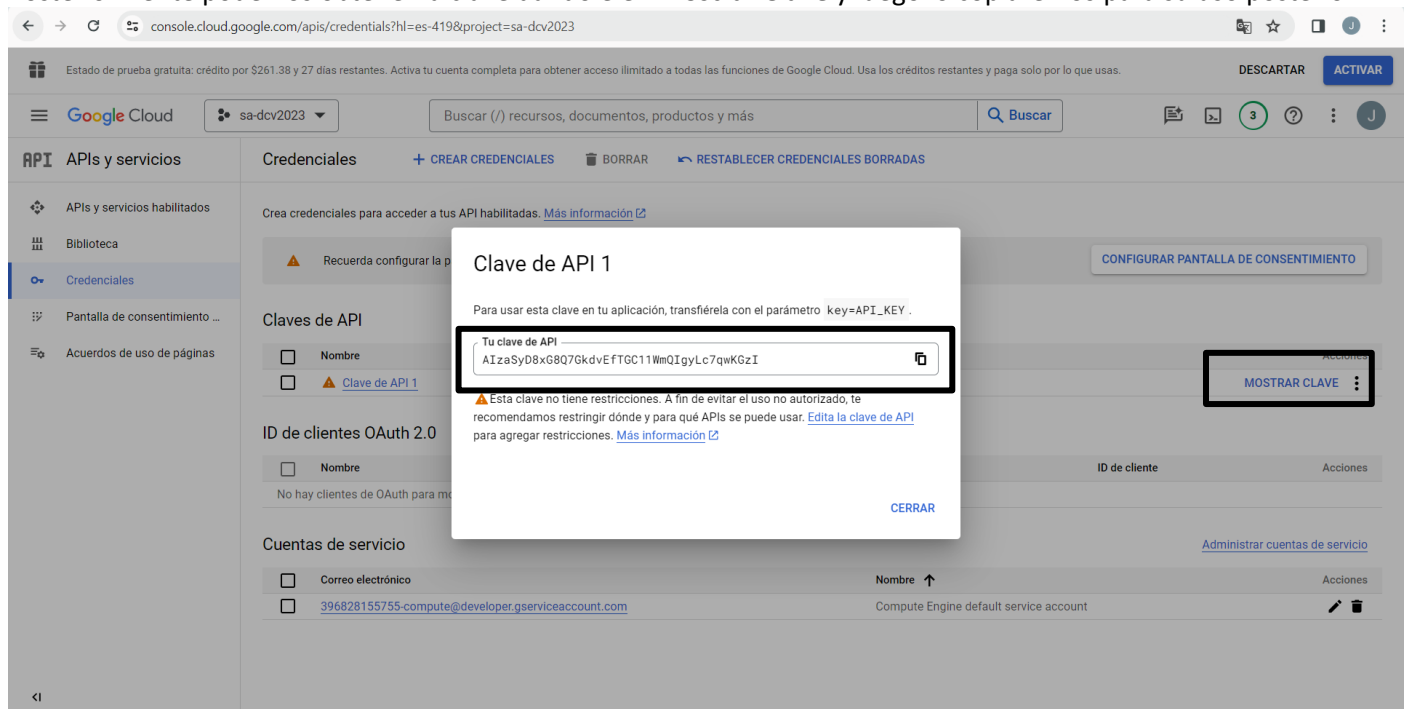
ID de clientes OAuth 2.0

| Nombre | Fecha de creación | Tipo | ID de cliente | Acciones |
|---------------------------------------|-------------------|------|---------------|----------|
| No hay clientes de OAuth para mostrar | | | | |

Cuentas de servicio [Administrar cuentas de servicio](#)

| Nombre | Acciones |
|--|--|
| Correo electrónico | |
| 396828155755-compute@developer.gserviceaccount.com | Compute Engine default service account |

Posteriormente podemos obtener la clave dandole en **Mostrar Clave** y luego lo copiaremos para su uso posterior.



Estado de prueba gratuita: crédito por \$261.38 y 27 días restantes. Activa tu cuenta completa para obtener acceso ilimitado a todas las funciones de Google Cloud. Usa los créditos restantes y paga solo por lo que usas. [DESCARTAR](#) [ACTIVAR](#)

Google Cloud sa-dcv2023 Buscar (/) recursos, documentos, productos y más

APIs y servicios

- APIs y servicios habilitados
- Biblioteca
- Credenciales**
- Pantalla de consentimiento ...
- Acuerdos de uso de páginas

Credenciales + CREAR CREDENCIALES BORRAR RESTABLECER CREDENCIALES BORRADAS

Crea credenciales para acceder a tus API habilitadas. [Más información](#)

Recuerda configurar la pantalla de consentimiento

Claves de API

- ☐ Nombre
- ☒ **Clave de API 1**

ID de clientes OAuth 2.0

| Nombre | Acciones | | | |
|---------------------------------------|----------|--|--|--|
| No hay clientes de OAuth para mostrar | | | | |

Cuentas de servicio [Administrar cuentas de servicio](#)

| Nombre | Acciones |
|--|--|
| Correo electrónico | |
| 396828155755-compute@developer.gserviceaccount.com | Compute Engine default service account |

Clave de API 1

Para usar esta clave en tu aplicación, transfírela con el parámetro `key=API_KEY`.

Tu clave de API
AIzaSyD8xG8Q7GkdVfTGC11WmQIgyLc7qwKGzI

Esta clave no tiene restricciones. A fin de evitar el uso no autorizado, te recomendamos restringir dónde y para qué APIs se puede usar. [Edita la clave de API](#) para agregar restricciones. [Más información](#)

[CERRAR](#)

[MOSTRAR CLAVE](#)

BACKEND

ENDPOINTS

En este proyecto únicamente se utilizó un endpoint:

- **/Analizar**
 - Tipo POST
 - Este endpoint realizara la consulta al servicio de **Vision AI** en GCP por medio del api
 - Retorna la respuesta de Vision AI y concatena la cantidad de Rostros a dicha respuesta.

```
@PostMapping("/Analizar")
public String analizarIMG(@RequestBody Map<String, String> requestBody) {
```

CLASES

La clase que utilizamos en esta aplicación es analizarIMG: este obtiene el json que enviamos desde frontend, para obtener la imagen en base64 y posteriormente consultarlo desde springboot hacia Vision AI, posteriormente analiza cuantos rostros detecta y retorna esta informacion hacia nuestro frontend.

```
@PostMapping("/Analizar")
public String analizarIMG(@RequestBody Map<String, String> requestBody) {
```

Cadena de Conexión

| Variable | Descripcion |
|-----------|---|
| img | Obtiene el valor de la llave base64Image, es decir nuestra imagen en base64. |
| ClaveGCP | Quemamos las credenciales para nuestra API |
| ulrString | Concatena la url que vamos a consultar a la api de Vision AI + la clave de GCP. |

```
try {
    String img = requestBody.get(key:"base64Image");
    // Url + Clave de API's
    String ClaveGCP = "AIzaSyD8xG8Q7GkdvEfTGC11WmQIgyLc7qwKGzI";
    String urlString = "https://vision.googleapis.com/v1/images:annotate?key=" + ClaveGCP;
```

| Variable | Descripcion |
|-------------------|---|
| cuerpoJSON | <p>Construimos el Body, con lo siguiente un request: que sera la lista de parametros que solicita la documentacion para realizar una peticion.</p> <p>Parametros necesarios.</p> <ul style="list-style-type: none"> • Image: imagen <ul style="list-style-type: none"> ◦ Content: cadena de la imagen en base64 • Features: Indica que es lo que necesitamos <ul style="list-style-type: none"> ◦ type: como se llama la etiqueta que deseamos que retorne nuestra la api de Vision AI. |

```
// Crear el cuerpo de la Peticion hacia Vision AI
String cuerpoJSON = "{"
    + "\"requests\":["
    + "{"
    + "\"image\":{\""
    + "\"content\": \"" + img + "\""
    + "},"
    + "\"features\":["
    + "{"
    + "\"type\": \"FACE_DETECTION\""
    + "},"
    + "{"
    + "\"type\": \"SAFE_SEARCH_DETECTION\""
    + "}"
    + "]"
    + "}"
    + "]"
    + "}";
```

Construccion de la Peticion

| Variable | Descripcion |
|------------|--|
| url | Esta variable es de tipo URL, crea un objeto tipo URL para convertir nuestra cadena de conexión/petición. |
| con | <p>Creamos una conexión tipo HttpURLConnection:</p> <ul style="list-style-type: none"> • Agregamos el tipo de metodo que sera nuestra peticion, en nuestro caso un tipo POST • Agregamos los Headers: Content-Type y el valor tipo: application/json |

```
// Crear una conexión HTTP
URL url = new URL(urlString); // Obtenemos la ruta completa y lo convertimos a tipo URL
HttpURLConnection con = (HttpURLConnection) url.openConnection();
// Configurar la solicitud HTTP
con.setRequestMethod(method:"POST"); // tipo de Peticion
con.setRequestProperty(key:"Content-Type", value:"application/json"); // Propiedades de la Petición
con.setDoOutput(dooutput:true);
```


| Variable | Descripcion |
|------------------------|---|
| respuesta | Esta variable es de tipo URL, crea un objeto tipo URL para convertir nuestra cadena de conexión/petición. |
| Respuesta | Parseamos a json la respuesta que devuelve la petición hacia Vision AI |
| Cara | Obtiene la lista del atributo faceAnnotations |
| cantidadRostros | Obtiene el tamaño de la lista de Caras. |

```
try (BufferedReader bufferRead = new BufferedReader(new InputStreamReader(con.getInputStream(), charsetName:"utf-8"))) {
    StringBuilder respuesta = new StringBuilder();
    String linea;
    while ((linea = bufferRead.readLine()) != null) {
        respuesta.append(linea.trim());
    }

    // Parsear la respuesta a formato JSON
    JsonObject Respuesta = JsonParser.parseString(respuesta.toString()).getAsJsonObject();
    // en la variable cara obtenemos el atributo en faceAnnotations
    JSONArray caras = Respuesta.getAsJsonArray(memberName:"responses").get(i:0).getAsJsonObject().getAsJsonArray(memberName:"faceAnnotations");
    int cantidadRostros = caras.size(); // Obtenemos el tamaño la lista de json de face Annotations
    Respuesta.addProperty(property:"cantidadRostros", cantidadRostros); // Agregar la cantidad de rostros a la respuesta JSON
    return Respuesta.toString(); // Devolver la respuesta modificada
}
```

CORS

```
@RestController
@CrossOrigin(origins = "**")
```

Utilizamos CorsOrigin para llamar a la librería, y dentro le enviamos el parametro origins = "*" lo que indica que permite el trafico de cualquier host.

FRONTEND

LIBRERIAS

- **AXIOS:** Esta librería se utiliza para realizar peticiones hacia nuestro backend
- **SWEETALERT2:** Librería que da estilo a nuestros Alerts.
- **BOOTSTRAP:** Librería que tiene definido estilos de css e implementarlo hacia nuestro html.
- **REACTSTRAP:** Librería que utiliza componentes pre-diseñados para su posterior uso.
- **MUI/MATERIAL:** En este caso utilizamos MUI unicamente para llamar un icono.

COMPONENETES UTILIZADOS

| | |
|---|--|
| useState | Utilizamos use state para crear variables que detecten el cambio de estado y se actualicen en tiempo real. |
| <pre>const [previewImage, setPreviewImage] = useState(null); const [selectedPanel, setSelectedPanel] = useState(null); const [faceDetectionMock, setfaceDetectionMock] = useState(null); const [tipoContenido, setTipoContenido] = useState(null); // Tipo de Contenido const [cantCaras, setCaras] = useState(0); const [varAdulto, setAdulto] = useState(0); const [varMedical, setMedical] = useState(0); const [varRacy, setRacy] = useState(0); const [varSpoof, setSpoof] = useState(0); const [varViolence, setViolence] = useState(0); // difuminado const [blurAmount, setBlurAmount] = useState(0); // valor de Mensajes y Color const [mensaje, setMensaje] = useState(""); const [color, setColor] = useState("white"); // Flags const [flag, setFlag] = useState(false); const [flagOpciones, setFlagOpciones] = useState(true);</pre> | |
| Rect-bootstrap | |
| Navbar | Es un componente que se ubica en el header del html |
| ProgressBar | Se utiliza para definir un porcentaje para nuestros atributos, en nuestro caso: violencia, adulto, etc. |
| Container | Es un componente propio de react el cual funciona como un div el cual contendra otros componentes dentro. |
| Card | Se utiliza como tipo de paneles. |
| Row | dentro del container definimos Rows, estos componentes crea filas. |
| Col | Dentro del container definimos Cols, estos componentes son columnas. |

| MUI - icons-material | |
|----------------------|---|
| VisibilityIcon | Creamos un componente que tenga un icono de visualizacion |

Funciones

convertiraBase64

esta funcion tiene 3 propositos

1. Verificar el tipo de Archivo
2. Convertir a base64 la imagen y almacenarlo en nuestro useState Post para alacenar el valor y posteriormente enviarlo en la peticion.
3. Visualizar la imagen.

```
const convertiraBase64 = (archivos) => {
  const allowedExtensions = ['png', 'jpg'];
  Array.from(archivos).forEach((archivo) => {
    const fileExtension = archivo.name.split('.').pop().toLowerCase();
    if (allowedExtensions.includes(fileExtension)) {
      setSelectedPanel(0);
      setFlagOpciones(true);
      var reader = new FileReader();
      reader.readAsDataURL(archivo);
      reader.onload = function () {
        var aux = [];
        var base64 = reader.result;
        console.log(base64);
        aux = base64.split(',');
        setPost({...post, foto: aux[1]}) // Guardamos el dato en UseState para posteriormente enviarlo.
        setPreviewImage(base64); // Guardar la imagen base64 en el estado para previsualización
      };
    } else {
      console.log('Solo se permiten archivos .png y .jpg');
      setFileInputKey((prevKey) => prevKey + 1);
      sweetAlert.fire({
        title: "Error",
        text: "Solo se permiten archivos .png y .jpg",
        icon: "error"
      });
    }
  });
};
```

AnalizarImagen

Esta función realiza la petición hacia nuestro backend, enviándole en un JSON body con nuestra imagen en base64, interpretamos la respuesta y almacenamos los parámetros en useStates, uno donde tenga la información de faceAnotations, la cantidad de caras y también el tipo de contenido.

```
const AnalizarImagen = () =>{
  setFlagOpciones(false)
  let body = {
    base64Image: post.foto
  }

  axios.post('http://localhost:8080/Analizar', body).then(response =>{
    //console.log(response.data)
    //console.log(response.data.responses[0])
    setFaceDetectionMock(response.data.responses[0])
    console.log(response.data.responses[0].safeSearchAnnotation)
    setTipoContenido(response.data.responses[0].safeSearchAnnotation)
    setCaras(response.data.cantidadRostros)
    //console.log(response.data.cantidadRostros)
    sweetAlert.fire({
      title: "Correcto",
      text: "Análisis Completo",
      icon: "success"
    });
  }).catch(error =>{
    sweetAlert.fire({
      title: "Error",
      text: "Ha ocurrido un error en la petición",
      icon: "error"
    });
    console.log(error.message);
  })
  //console.log(body)
};
```

showPanel

Esta función tiene 2 funciones:

1. Detecta que panel deseamos observar: Cara o Tipo Contenido.
2. Realiza el el calculo de los parametros: adult, spoof, medical, violence y racy.

```
const showPanel = (panelNumber) => {
  setSelectedPanel(panelNumber);

  setAdulto(calculo(tipoContenido.adult))
  setSpoof(calculo(tipoContenido.spoof))
  setMedical(calculo(tipoContenido.medical))
  setViolence(calculo(tipoContenido.violence))
  setRacy(calculo(tipoContenido.racy))
  caluloBlur();
};
```

calculoBlur

Realiza los cálculos para la censura de la imagen, también dependiendo de la sumatoria envía texto a nuestro useState para Mensajes y también para el color del texto.

```
const calculoBlur = ()=>{
  if (varViolence > 59){
    console.log("filtro por contenido de violencia")
    //setMensaje("filtro por contenido de violencia")
    setBlurAmount(20);
  }

  if (varRacy > 50){
    console.log("filtro por contenido picante")
    //setMensaje("filtro por contenido picante")
    setBlurAmount(20);
  }

  if (varAdulto > 40){
    console.log("filtro por contenido picante")
    //setMensaje("filtro por contenido picante")
    setBlurAmount(20);
  }

  var sumatoria = varViolence + varRacy + varAdulto;
  if(sumatoria > 45){
    setColor("red");
    setMensaje("Imagen no apta para la institución");
    setFlag(false);
    setBlurAmount(20);
  }else{
    setColor("green");
    setMensaje("Imagen valida");
    setBlurAmount(0);
    setFlag(true);
  }
}
```