Nueva Guatemala de la Asunción

Universidad San Carlos de Guatemala

Inteligencia Artificial

Ciclo 2024

Manual Técnico

Estuardo Gabriel Son Mux

202003894

Objetivos

- Elaborar un software para la comprensión del funcionamiento de las librerías de Google para el análisis de imágenes.
- Poner en practica los conocimientos adquiridos en inteligencia artificial por medio del uso de Cloud Vision API.
- Utilizar los frameworks Spring boot y Angular para la creación de una aplicación el manejo el manejo de respuestas del servicio en un entorno web.

Backend

mainController.java

En este archivo se configuran los endpoints a los cuales se enviaran las solicitudes rest para el análisis de los archivos jpg y png.

Inicio de la clase con la etiqueta @RestController que indica que sera utilizada como controlador de endponts.

```
@RestController
public class mainController {
```

Enpoint “/”

Creación del endpoint “/” de tipo GET definido por la etiqueta @GetMapping. Cuenta con la etiqueta @CrossOrigin(origins = "*") para permitir el acceso desde cualquier punto. Así mismo, retorna un valor string que contiene el nombre y carnet del creador.

```
@CrossOrigin(origins = "*")
	@GetMapping("/")
	public String hello() {
		return "Estuardo Gabriel Son Mux - 202003894";
	}
```

Endpoint “/scan”

Creación del endpoint “/scan” de tipo POST definido por la etiqueta @PostMapping. Cuenta con la etiqueta @CrossOrigin(origins = "*") para permitir el acceso desde cualquier punto. Esta función recibe por paramento un archivo (File type MultipartFile) el cual será analizado.

```
@CrossOrigin(origins = "*")
	@PostMapping("/scan")
	public reponse scan(@RequestParam("file") MultipartFile file) throws
IOException {
		try (ImageAnnotatorClient vision = ImageAnnotatorClient.create()) {
```

Crear lista de solicitudes

```
List<AnnotateImageRequest> requests = new ArrayList<>();
```

Crear caracteristicas a buscar

```
Feature feat = Feature.newBuilder().setType(Type.SAFE_SEARCH_DETECTION).build();
                    Feature featFace =
Feature.newBuilder().setType(Type.FACE_DETECTION).build();
```

Parsear la imagen

```
ByteString imgBytes = ByteString.copyFrom(file.getBytes());
Image img = Image.newBuilder().setContent(imgBytes).build();
```

Unir las configuraciones de busqueda para crear las solicitudes

```
AnnotateImageRequest request =
AnnotateImageRequest.newBuilder().addFeatures(feat).setImage(img).build();
AnnotateImageRequest requestFace =
AnnotateImageRequest.newBuilder().addFeatures(featFace).setImage(img).build();
```

Agregar las solicitudes a la lista

```
requests.add(request);
requests.add(requestFace);
```

Almacenar las respuestas

```
BatchAnnotateImagesResponse response = vision.batchAnnotateImages(requests);
List<AnnotateImageResponse> responses = response.getResponsesList();
```

Separar las respuestas en anotaciones diferentes

```
AnnotateImageResponse AIRtags = responses.get(0);
AnnotateImageResponse AIRfaces = responses.get(1);
```

Procesar la respuesta de Security Search Detection y almacenarlas en una lista.

```
List<tag> tags = new ArrayList<>();

                if (AIRtags.hasError()) {
                    System.out.format("Error: %s%n",
AIRtags.getError().getMessage());
                }else {
                   SafeSearchAnnotation annotation =
AIRtags.getSafeSearchAnnotation();

                   tags.add(new tag("Adulto", annotation.getAdultValue()));
                         tags.add(new tag("Parodia",
annotation.getSpoofValue()));
                         tags.add(new tag("Medico",
annotation.getMedicalValue()));
                         tags.add(new tag("Caliente",
annotation.getRacyValue()));
```

```
                        tags.add(new tag("Violencia",
annotation.getViolenceValue()));
                }
```

Procesar la respuesta de Face Detection y almacenar la cantidad de resultados en una variable.

```
int rostros = 0;

                if (AIRfaces.hasError()) {
                   System.out.format("Error: %s%n",
AIRfaces.getError().getMessage());
                }else {
                   List<FaceAnnotation> annotation =
AIRfaces.getFaceAnnotationsList();

                   rostros = annotation.size();
                }

                return new reponse(200, tags, rostros);
```

Tag.java

Esta clase se utiliza para la creación de los objetos tipo tag que contendrán el nombre y porcentaje de las respuestas dadas por el servicio de cloud visión.

```
package com.backend.demo.models;

public class tag {
      public String nombre;
      public float porcentaje;

      public tag(String nombre, float porcentaje) {
            super();
            this.nombre = nombre;
            this.porcentaje = this.getPorcentaje(porcentaje);
      }
```

Función para que retorna un porcentaje según una escala de 0 a 5.

```
      private float getPorcentaje(float escala) {
            if (escala > 1) {
                  return (escala - 1) * 25;
            }
            return 0;
      }
}
```

Response.java

Clase que sirve para estandarizar las respuestas devueltas por el servidor conteniendo el mensaje, código, lista de tags y cantidad de rostros obtenidos en el análisis de una imagen.

```
package com.backend.demo.models;
```

```java
import java.util.List;

public class reponse {
	public int code;
	public List<tag> tags;
	public String msg;
	public int rostros;

	public reponse(int code, List<tag> tags, int rostros) {
		super();
		this.code = code;
		this.tags = tags;
		this.rostros = rostros;
		this.msg = "Correcto";
	}

	public reponse(int code, String msg) {
		super();
		this.code = code;
		this.msg = msg;
	}
}
```

Application.properties

Configuración de opciones para el funcionamiento de la aplicación en spring boot.

```
spring.servlet.multipart.enabled=
spring.servlet.multipart.max-file-size=
spring.servlet.multipart.max-request-size=

spring.cloud.gcp.sql.credentials.location=
```

Frontend

Visor.component

Html

En este apartado se realiza la estructura que contiene el apartado para la carga de la imagen y su visualización

```
<div id="contenedor" class="p-3">
  <div class="row mx-0 justify-content-evenly p-2">
    <div class="col col-6">
      <input type="file" class="form-control"
(change)="onFileChange($event)" accept=".jpg, .png" />
    </div>
    <button
      class="btn btn-primary col col-4 align-items-center"
      (click)="onUpload($event)"
```

```
    >
      UPLOAD
    </button>
  </div>
  <br />
  <div class="row mx-0 justify-content-evenly">
    <div class="col col-8">
      <div class="d-flex justify-content-center">
        <img [src]="url" [style]="estilo" alt="" class="img-fluid" />
      </div>
    </div>
  </div>
  <div class="d-flex justify-content-center mt-2"
[style.color]="validacion.color">
    <h4>{{ validacion.text }}</h4>
  </div>
</div>
```

Typescript

Configuraciones iniciales e importe de la librería ReactiveFormsModule.

```
@Component({
  selector: "app-visor",
  standalone: true,
  imports: [ReactiveFormsModule],
  templateUrl: "./visor.component.html",
  styleUrl: "./visor.component.css",
})
```

Creación de la clase VisorComponent el donde se con las variables globales file que contendrá el archivo jpg o png, url que contendrá la direccion de la imagen, validación el cual es un diccionario para almacenar información acerca de resultados de consultas y estilo que contiene información para css.

```
export class VisorComponent {
  file: Blob | null = null;
  url = "";
  validacion = {
    text:"",
    color:""
  };

  estilo = {
    filter: `blur(0px)`
  }
```

Creación del constructor que utilizara el servicio BackendService.

```
  constructor(private backendService: BackendService) {}
  ngOnInit(): void {}
```

Crear la función onUpload que recibe el evento de un botón, al accionarse el botón enviara la imagen ingresada en un input type file para analizarlo, al recibir una respuesta aplicara ciertas modificaciones a varibles de la clase o variables que se encuentren en el servicio backendService.

```
  onUpload(event: any) {
    event.preventDefault();

    let formData = new FormData();

    if (this.file) {
      formData.append("file", this.file);
      this.backendService.postURL("/scan", formData).subscribe(
        (res:any) => {
          console.log(res);
          this.backendService.setTags(res.tags);
          this.backendService.setCont(res.rostros);

          if(res.tags[0].porcentaje >= 40 || res.tags[3].porcentaje >= 50 ||
res.tags[4].porcentaje >= 59) {
            this.estilo = {
              filter: `blur(10px)`
            }
          }

          let sum = res.tags[0].porcentaje + res.tags[3].porcentaje +
res.tags[4].porcentaje;
          if (sum >= 45) {
            this.validacion = {
              text:"Imagen no apta para la institución",
              color:"red"
            };
          } else {
            this.validacion = {
              text:"Imagen valida",
              color:"green"
            };
          }
        },
```

```
        (err) => {
          console.log(err);
        }
      );
    }
  }
```

Creación de la función onFileChange que recive un evento, el evento es proporcionado al realizar la carga de una imagen dentro de un input type file, esto ocasionara que se muestre la imagen y se retorne a los valores de inicio de las variables de la clase.

```
  onFileChange(event: any) {
    this.url = URL.createObjectURL(event.target.files[0]);
    this.file = event.target.files[0];
    this.validacion = {
      text:"",
      color:""
    };
    this.estilo = {
      filter: `blur(0px)`
    }
  }
}
```

Etiquetas.component

Html

En este apartado se crea la estructura que mostrara las etiquetas y porcentajes respectivos obtenidos de la consulta al api de Google cloud visión.

```
<div id="contenedor">
  <h4>Cantidad de Rostros: {{ backendService.cont }}</h4>
</div>
<div id="contenedor">
  <h4>Etiquetas</h4>
  <div class="row mx-0 justify-content-evenly">
      @for (tag of backendService.tags; track $index) {
        <div class="row mx-0 justify-content-evenly">
          <div class="col col-2">{{ tag.nombre }}</div>
          <div class="col col-10" style="margin-top: 5px;">
            <div class="progress">
              <div [class]="getColor(tag.porcentaje)" role="progressbar"
[style.width.%]="tag.porcentaje"  aria-valuemin="0" aria-valuemax="100">{{
tag.porcentaje }}%</div>
```

```
            </div>
          </div>
        </div>
      } @empty {
        <p>No hay etiquetas!!!!</p>
      }
  </div>
</div>
```

Typescript

Creación de la clase EtiquetasComponent que contiene las configuraciones para la visualización de las barras de proceso así como la función getColor que recibe un valor numérico que indica el color con el cual se presentara la barra de porcentaje.

```
export class EtiquetasComponent {

  constructor(public backendService: BackendService) {
  }

  getColor(value: number) {
    if (value >= 100) {
      return 'progress-bar bg-danger';
    }else if (value >= 75) {
      return 'progress-bar bg-warning';
    }else if (value >= 50) {
      return 'progress-bar bg-success';
    }else if (value >= 25) {
      return 'progress-bar bg-info';
    }
    return 'progress-bar bg-info';
  }
}
```

Backend.service

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class BackendService {

  url = 'http://localhost:8080';
  tags:Array<any> = [];
```

```
  cont = 0;

  constructor(private http:HttpClient) { }

  public postURL(endpoint: string, body: any, options?: any) {
    return this.http.post(this.url+endpoint, body);
  }

  public setTags(tags: Array<any>) {
    this.tags = tags;
  }

  public setCont(cont: number) {
    this.cont = cont;
  }
}
```