Universidad de San Carlos de Guatemala

Inteligencia Artificial I

Escuela de Sistemas

Sección: “A”.

Catedrático: Ing. Luis Fernando Espino Barrios

Auxiliar: Erick Eden Sandoval Ramirez

Manual Tecnico

Nombre:
Brandon Oswaldo Yax Campos

Carnet:
201800534

24 de febrero de 2024

Objetivos:

Objetivos Generales:

Permitir la exploración y familiarización de diversas librerías utilizadas en el análisis de imágenes, fomentando así su comprensión de las tecnologías relevantes en este campo.

Objetivos Específicos:

- Comprender en profundidad el funcionamiento y la aplicación de la librería de Google para el análisis de imágenes.
- Desarrollar habilidades prácticas al utilizar las funcionalidades proporcionadas por la librería de Google.
- Aprender a manejar de manera efectiva las respuestas generadas por los servicios de análisis de imágenes, integrándose en un entorno web para su visualización y uso práctico.
- Aprender sobre el uso de Java como backend a través del framework Spring Boot.

Descripción de las clases en Backend Java:

Para el desarrollo del backend fue necesario la creación de 3 clases fundamentales:

- **BackendAplication:** Esta clase actúa como el punto de entrada principal del proyecto. Está anotada con @SpringBootApplication, lo que la convierte en la clase principal de Spring Boot. Su función principal es iniciar la aplicación Spring Boot.

```
package com.usac.Backend;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@SpringBootApplication
public class BackendApplication {

   public static void main(String[] args) {
      SpringApplication.run(BackendApplication.class, args);
   }
}
```

- **CorsConfiguration:** Esta clase está diseñada para configurar los cors, permitiendo la conexión entre React y Spring Boot. Está anotada con @Configuration y @EnableWebMvc, e implementa la interfaz WebMvcConfigurer para proporcionar métodos de configuración de CORS.

```
package com.usac.Backend;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
@EnableWebMvc
public class CorsConfig implements WebMvcConfigurer {

   @Override
   public void addCorsMappings(CorsRegistry registry) {
       registry.addMapping("/**")
               .allowedOrigins("*")
```

```
                .allowedMethods("*")
                .allowedHeaders("*");
    }
}
```

- **ControladorRest:** Es la encargada de gestionar todos los endpoints utilizados en el backend de la aplicación. En esta clase se definen los métodos para manejar las solicitudes HTTP entrantes y proporcionar las respuestas correspondientes.

```
package com.usac.Backend;

import lombok.extern.slf4j.Slf4j;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfigura
tion;

import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.multipart.MultipartFile;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Base64;
import java.util.List;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.client.RestTemplate;
import org.springframework.http.HttpHeaders;



@RestController
@Slf4j
@SpringBootApplication(exclude = {SecurityAutoConfiguration.class})
public class ControladorRest {
    @GetMapping("/")
    public String Comienzo() {
```

```
        log.info("Peticion GET - Inicio");
        log.debug("Información Extra: ");
        return "Hola Mundo";
    }
    @GetMapping("/1")
    public String Comienzo1() {
        log.info("Peticion GET - Inicio");
        log.debug("Información Extra: ");
        return "Hola Mundo";
    }

    @PostMapping("/analisisImage")
    @CrossOrigin(origins = "http://localhost:4000")
       public  List<String>  analizarImagen(MultipartFile  imageFile)  throws
IOException {
        List<String> results = new ArrayList<>();
        if (imageFile == null) {
            results.add("Se envió un archivo nulo");
            return results;
        }
        byte[] imageB = imageFile.getBytes();
        String imageB64 = Base64.getEncoder().encodeToString(imageB);
         String keyV = "AIzaSyAz3G7fw2ZYe9rTBtPusiEZshRrP4P02JU"; // Agrega tu
clave API aquí
        String url = "https://vision.googleapis.com/v1/images:annotate?key=" +
keyV;
        /*
        * LABEL_DETECTION (detección de etiquetas) es una función de la API de
Google Vision que identifica objetos, lugares, actividades, productos y otros
elementos significativos en una imagen. Básicamente, proporciona una lista de
etiquetas que describen el contenido de la imagen.
             Por ejemplo, si tienes una imagen de una playa, el LABEL_DETECTION
podría devolver etiquetas como "playa", "mar", "arena", "cielo", "personas",
etc.
        * TEXT_DETECTION: Para detectar texto en la imagen.
        * FACE_DETECTION: Para detectar rostros en la imagen.
        * LANDMARK_DETECTION: Para detectar puntos de referencia en la imagen.
        * LOGO_DETECTION: Para detectar logotipos en la imagen.
        * DOCUMENT_TEXT_DETECTION: Para detectar texto en documentos dentro de
la imagen.
```

```
         * IMAGE_PROPERTIES: Para obtener propiedades de la imagen, como
colores dominantes.
         * SAFE_SEARCH_DETECTION: Para detectar contenido inapropiado en la
imagen.
       */
       String jsonBody = "{"
               + "\"requests\":[{"
               + "\"image\":{\"content\":\"" + imageB64 + "\"" + "},"
               + "\"features\":["
               + "{\"type\":\"FACE_DETECTION\"},"
               + "{\"type\":\"SAFE_SEARCH_DETECTION\"}"
               + "]}]"
               + "}";

       HttpHeaders headersp = new HttpHeaders();
       headersp.setContentType(MediaType.APPLICATION_JSON);
       HttpEntity<String> rEntity = new HttpEntity<>(jsonBody, headersp);
       RestTemplate restTemplate = new RestTemplate();
                               ResponseEntity<String>    responseEntity    =
restTemplate.postForEntity(url, rEntity, String.class);
       HttpStatus statusCode = (HttpStatus) responseEntity.getStatusCode();
       if (statusCode == HttpStatus.OK) {
           String responseBody = responseEntity.getBody();
           System.out.println("Respuesta del servidor: " + responseBody);
           results.add(responseBody);
       } else {
            System.err.println("La solicitud no se pudo completar. Código de
estado: " + statusCode);
       }
       return results;
   }
}
```

Descripción de los componente y métodos (Frontend):

Componentes:

- **Componente Homepage:**

  Este componente representa la página principal de la aplicación. Utiliza varios componentes de Material-UI, como Grid, Typography, Table, TableContainer, TableRow, TableCell, Paper, Alert, etc. Utiliza estados para manejar la visibilidad de

las alertas, la información de la imagen, el número de caras detectadas y el tipo de contenido que representa la imagen. Define funciones handleCargarClick y handleProcesarClick para cargar y procesar una imagen, respectivamente y utiliza useEffect para realizar acciones basadas en cambios en el estado de “percentage”, como mostrar alertas y aplicar efectos de difuminado a la imagen.

Renderiza un formulario con un campo de carga de archivos, botones para cargar y procesar la imagen, y una sección para mostrar la imagen, la información de la imagen y el porcentaje de análisis y muestra alertas dependiendo de los resultados del análisis de la imagen.

- **Componente TableComponent:**

  Este componente renderiza una tabla para mostrar la información de la imagen y el porcentaje de análisis. Recibe dos props: Data para la información de la imagen y Percentage para el porcentaje de análisis. Mapea los datos recibidos para renderizar las filas de la tabla.

Componentes y Librerías Utilizadas:

- useState: Hook de React para manejar estados locales en componentes de función.
- useEffect: Hook de React para ejecutar efectos secundarios en componentes funcionales.
- useRef: Hook de React para crear referencias a elementos del DOM.
- axios: Librería para hacer peticiones HTTP desde el navegador o Node.js.
- Componentes de Material-UI: Proporcionan una interfaz de usuario consistente y estilizada para la aplicación.

Métodos:

- **handleCargarClick:** Este método se activa cuando el usuario hace clic en el botón "Cargar". Obtiene el archivo de imagen seleccionado por el usuario utilizando la referencia fileInputRef y lee el contenido del archivo de imagen seleccionado y lo muestra en la interfaz de usuario.
- **handleProcesarClick:** Este método se activa cuando el usuario hace clic en el botón "Procesar". Envía una petición POST al servidor backend utilizando la librería axios, adjuntando el archivo de imagen como un objeto FormData y maneja la respuesta del servidor, parsea los datos recibidos y actualiza el estado local percentage con el porcentaje de análisis de la imagen.

- **useEffect:** Este hook se utiliza para realizar acciones basadas en cambios en el estado de percentage. Se activa cada vez que cambia el estado de “percentage” y en función del porcentaje de análisis de la imagen, este método aplica efectos visuales a la imagen, como el difuminado, y muestra alertas según ciertas condiciones.

Descripción de los componente y métodos (Backend):

Métodos:

Clase BackendApplication:

- Método main: Método principal que inicia la aplicación Spring Boot.

Clase CorsConfig:

- Método addCorsMappings: Configura las reglas CORS para permitir la conexión entre React y Spring Boot.

Clase ControladorRest:

- Método Comienzo: Endpoint que devuelve un saludo "Hola Mundo" al acceder a la página de inicio.
- Método analizarImagen: Endpoint para analizar una imagen. Recibe un archivo de imagen, lo convierte a base64 y lo envía a Google Vision para obtener información sobre los rostros detectados y el contenido seguro de la imagen.

Componentes y Librerías Utilizadas:

- **@SpringBootApplication:** Anotación que indica que la clase es una aplicación Spring Boot.
- **@Configuration:** Anotación que indica que la clase contiene métodos de configuración.
- **@EnableWebMvc:** Anotación que habilita la configuración de MVC en Spring.
- **@RestController:** Anotación que combina @Controller y @ResponseBody, lo que significa que cada método devolverá un objeto serializado directamente en la respuesta HTTP.
- **@GetMapping:** Anotación para mapear solicitudes HTTP GET a métodos de manejo específicos.
- **@PostMapping:** Anotación para mapear solicitudes HTTP POST a métodos de manejo específicos.
- **@CrossOrigin:** Anotación para configurar los orígenes CORS permitidos para un controlador específico.

- **MultipartFile:** Clase que representa un archivo recibido en una solicitud multipart.
- **Base64:** Clase para codificar y decodificar datos en base64.
- RestTemplate: Clase que proporciona un cliente HTTP fácil de usar para realizar solicitudes HTTP.