BlindView

Soporte visual para personas invidentes

1. Descripción de la aplicación

La aplicación está destinada a dispositivos Android, y utilizará la cámara del dispositivo para captar imágenes, procesarlas mediante inteligencia artificial y extraer conclusiones en base a los requerimientos del usuario. El objetivo de la aplicación es ayudar a las personas invidentes a orientarse en el espacio que los rodea, además de servir como interfaz accesible de audio para utilizar inteligencia artificial conversacional.

2. Funcionamiento de la aplicación

El dispositivo reconocerá las palabras pronunciadas por el usuario, y será capaz de reconocer comandos, o en su defecto, iniciar el modo de conversación libre.

Los comandos disponibles se pueden listar pronunciando la palabra “menú”, mostrando los siguientes modos:

- **orientación**: en este modo, el programa realizará fotografías cada 20 segundos, y describirá la imagen percibida por la cámara. Para desactivar este modo, pronuncie “desactivar”.
  Ejemplo: “orientación”
- **pregunta**: en este modo, se responderá cualquier pregunta (solo texto).
  Ejemplo: “pregunta ¿Cuánto es 18x31?”
- **semáforo**: en este modo, el dispositivo identificará todos los semáforos percibidos por la cámara, e indicará su estado.
  Ejemplo: “semáforo”
- **lectura**: en este modo, el dispositivo leerá en voz alta todo el texto presente en la imagen.
- **indicaciones**

- **desactivar**
- **Modo libre**

El dispositivo utilizado realizará fotografías mediante la cámara y reconocerá las indicaciones por voz de la persona. Estas fotografías serán enviadas a una inteligencia artificial que reconocerá el espacio en las fotografías e identificará los obstáculos, enviando la respuesta a una solución de texto a voz, para que puedan ser leídas al usuario en voz alta.

3. Interfaz de usuario de la aplicación

La aplicación dispondrá de una vista principal, que tendrá como fondo la cámara del dispositivo, donde se resaltará el texto detectado y desde el que se leerá por el altavoz. La aplicación escuchará a través del micrófono del dispositivo mientras se mantenga la pantalla pulsada. Para aumentar el *feedback* auditivo, se reproduce un sonido cuando empiezas o dejas de mantener pulsado.

Además, la aplicación incluye un apartado de configuración, que permitirá administrar las claves de las API's de ChatGPT y Google Maps.

4. Herramientas utilizadas

- Android Studio (creación de la aplicación y texto a voz)
- https://developer.android.com/studio?hl=es-419
  https://developer.android.com/reference/android/speech/tts/TextToSpeech
  https://developer.android.com/training/camerax?hl=es-419
- API de ChatGPT
- API de Google Maps

5. Desarrollo de la aplicación

En un primer momento, el objetivo de la aplicación era simplemente leer el texto capturado por la pantalla. Para esto, habíamos utilizado la librería de OCR (Reconocimiento óptico de carácteres) *tess4j*. Lamentablemente, esta tecnología está limitada a día de hoy, y presenta muchos errores y fallos, sumados a tiempos de respuesta muy altos.

Debido a esto, acompañado por los recientes avances de la inteligencia artificial, decidimos encaminar BlindView hacia usos más pragmáticos, apostando por una mayor abarcabilidad. De este modo, la aplicación podría interpretar el entorno completo del usuario, en lugar de limitarse al texto de la imagen.

En un primer enfoque, decidimos usar la API de Google, Vertex, con el objetivo de utilizar los modelos de IA Gemini. Tras unas semanas de desarrollo, y pese a haber conseguido hacer llamadas a la API con éxito, encontramos que las limitaciones de uso geográfico, la dificultad de uso y la escasez de documentación de Vertex la hacían incompatible con el desarrollo previamente propuesto.

Posteriormente, elegimos utilizar la API de OpenAI, que nos permitiría acceder a los modelos más avanzados de ChatGPT, en sus versiones *turbo*. Para nuestra sorpresa, el servicio de OpenAI presenta una amplia documentación respecto a la subida multimedia, además de ser mucho más simple y sencillo de usar que los modelos de Google.

El proyecto está desarrollado íntegramente en Android Studio, utilizando el lenguaje de programación Java, caracterizado por su eficiente diseño de programación orientada a objetos. A continuación se describen las principales clases de la aplicación:

- **MainActivity**: se trata de la vista principal o *activity* de la aplicación, en la que se muestra la cámara del dispositivo y el usuario puede interactuar con la inteligencia artificial.

  En su constructor se inicializan los permisos de la cámara, ubicación y micrófono, para posteriormente cargar la previsualización de la cámara (*Bitmap* o mapa de bits) e inicializar el servicio de texto-a-voz (*Text-to-speech*), reconocimiento de voz y ubicación.

- **Interfaz**: se trata de la clase que administra las interacciones con el usuario, cuyos métodos principales se enfocan a procesar la entrada del usuario.

  A continuación, un breve extracto de la función *procesarResultado* en la clase *Interfaz*:

```
//Procesar entrada del usuario
public void procesarResultado(String entrada, Bitmap
bitmap, Context context) {
   //Comprobar si la entrada está vacía
   if (entrada.isEmpty()) return;

   [...]

   //Pasamos la entrada a minúscula para revisar los
casos
   entrada = entrada.toLowerCase();

   [...]
     Toast.makeText(context,entrada,Toast.LENGTH_SHORT
).show();
```

```
    //Convertimos la imagen a Base64, tras reescalar
la imagen
    String base64 = Utils.bitmapToBase64 (
Bitmap.createScaledBitmap(bitmap, bitmap.getWidth() /
2, bitmap.getHeight() / 2, true)
    );

    //Realizamos las comprobaciones
    if (entrada.startsWith("menú")) {
        tts.speak("Las opciones disponibles son:
orientación, pregunta, semáforo, lectura, indicaciones
y desactivar", TextToSpeech.QUEUE_ADD, null);
      return;
   } else if (entrada.startsWith("pregunta")) {
      if (entrada.split(" ").length <= 1) {
          tts.speak("Repite el comando, por favor",
TextToSpeech.QUEUE_ADD, null);
          return;
      }

      tts.speak(Utils.chatGPT(entrada.split(" ",
2)[1]), TextToSpeech.QUEUE_ADD, null);
      return;
   } else if (entrada.startsWith("buscar")) {
      if (entrada.split(" ").length <= 1) {
          tts.speak("Repite el comando, por favor",
TextToSpeech.QUEUE_ADD, null);
          return;
      }
     /* if (mode == Mode.BUSQUEDA) {
```

```
            mode = null;
            tts.speak("Modo búsqueda desactivado",
TextToSpeech.QUEUE_ADD, null);
            return;
        }
        mode = Mode.BUSQUEDA;*/
        buscando = entrada.split(" ", 2)[1];
        switchMode(Mode.BUSQUEDA);
        return;
    }

    [...]

    tts.speak(Utils.chatGPTWithImage(entrada, base64),
TextToSpeech.QUEUE_ADD, null);
}

[...]
```

- **SettingsActivity**: se trata de la actividad de configuración de la aplicación, que permite administrar las claves de las APIs.

- **Utils**: se trata de una clase de utilidades. Particularmente, recopila las funciones para hacer peticiones a las APIs de ChatGPT y Google Maps, además de *parsear* las respuestas obtenidas y convertir el *bitmap* o mapa de bits de la cámara a Base64 (reescalando y aplicando un filtro de *sharpening* previamente).

A continuación, un extracto de las funciones *chatGPT* y *bitmapToBase64*, que devuelven un resultado acorde al *prompt* envíado y transforman un mapa de bits a Base64 respectivamente:

```
//Devuelve el resultado de la petición con el prompt
pasado
public static String chatGPT(String prompt) {
   //Endpoint para los modelos de OpenAI
   String url =
"https://api.openai.com/v1/chat/completions";
   //Modelo utilizado
   String model = "gpt-3.5-turbo";

   //Petición a ChatGPT
   try {
       URL obj = new URL(url);
       HttpURLConnection connection =
(HttpURLConnection) obj.openConnection();
       connection.setRequestMethod("POST");
       //Autenticación mediante clave de API
       connection.setRequestProperty("Authorization",
"Bearer " + MainActivity.CHATGPT_API_KEY);
      //Respuesta obtenida en formato JSON
       connection.setRequestProperty("Content-Type",
"application/json");

        //Petición en formato JSON
       String body = "{\"model\": \"" + model + "\",
\"messages\": [{\"role\": \"user\", \"content\": \"" +
prompt + "\"}]}";
       connection.setDoOutput(true);
```

```
        OutputStreamWriter writer = new
OutputStreamWriter(connection.getOutputStream());
        writer.write(body);
        writer.flush();
        writer.close();

        BufferedReader br = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
        String line;

        StringBuffer response = new StringBuffer();

        while ((line = br.readLine()) != null) {
            response.append(line);
        }
        br.close();

        return
extractMessageFromJSONResponse(response.toString());

    } catch (IOException e) {
        return "La clave de API proporcionada es
incorrecta.";
    }
}

//Convierte el mapa de bits de la cámara a Base64
(tras comprimirlo), para poder enviarlo a ChatGPT
public static String bitmapToBase64(Bitmap bitmap) {
    ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();
```

```
    //Comprimimos la imagen a formato PNG
    bitmap.compress(Bitmap.CompressFormat.PNG, 0,
byteArrayOutputStream);
    //Codificamos el array de bits del mapa de bits en
formato Base64, y lo devolvemos como String
    byte[] byteArray =
byteArrayOutputStream.toByteArray();
    return
Base64.getEncoder().encodeToString(byteArray);
}
```