UNIVERSIDAD DE GRANADA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
INFORMÁTICA Y TELECOMUNICACIONES

Programación de Dispositovos Móviles

MEMORIA TÉCNICA PARA LA APLICACIÓN

HELLO TO ME !

Jose Manuel Osuna Luque

Índice

1. Introducción

En este proyecto se ha construido una aplicación Android que facilite la interacción entre dos personas. En un primer momento, la aplicación se pensó exclusivamente para personas con cierta discapacidad a la hora de hablar o escuchar, creándose esta con la intención de permitir a una persona con discapacidad auditiva poder captar lo que otra persona esté diciendo en la pantalla de su dispositivo móvil, y de permitir que una persona con dificultad al hablar pudiese escribir lo que quisiera decir y transmitirlo en audio a la persona con la que esté hablando, creando así una comunicación más directa y natural, y con tan solo disponer de su dispositivo móvil (que, como es sabido, hoy día, más del 80 por ciento de la población dispone de uno).

La aplicación cuenta con una interfaz principal desde donde se gestiona toda la funcionalidad de la aplicación. Cuenta con una caja de texto donde el usuario podrá escribir manualmente, o donde se recogerá aquello que este diga al teléfono. Antes de entrar en más detalles, se explicará qué es lo que puede hacerse con la aplicación, cuyas funcionalidades se ejecutan a través de los botones que tiene en la parte inferior.

Figura 1: Interfaz Principal

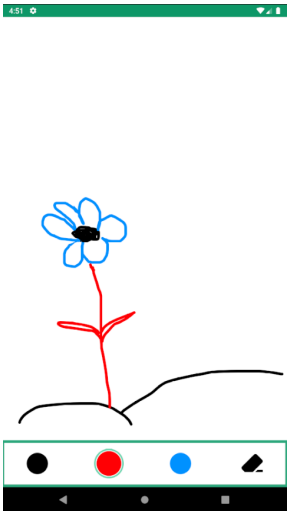

Figura 2: Pizarra

Explicando los botones de izquierda a derecha, estos realizan la siguientes funcionalidades:

- **Text-To-Speech.** Al pulsar el usuario este botón se reproducirá el contenido escrito. en la caja de texto superior (en el idioma indicado por la bandera a la derecha).
- **Blackboard.** Al pulsar este botón se abrirá una nueva pantalla en la que el usuario podrá realizar un dibujo.
- **Speech-To-Text.** Si el usuario mantiene pulsado este botón, el teléfono comenzará a captar lo que este le diga, y lo redactará en la caja de texto superior.

2. Desarrollo

Figura 3: HelloToMe!

Figura 4: Icono

HelloToMe! es una aplicación pensada para facilitar la comunicación a aquellas personas que posean alguna discapacidad, como por ejemplo Discapacidad Auditiva o Discapacidad del Habla. De esta forma, una persona incapaz de hablar, o tener grandes dificultades para ello, puede escribir el mensaje que quiera transmitir en la caja de texto, y pulsar el botón de *Text-To-Speech* para reproducir lo escrito de forma oral. Por otra parte, está el caso de una persona que no pueda escuchar, donde el individuo podría pulsar el botón *Speech-To-Text* para recoger lo que la persona con la que está comunicándose esté diciendo y transcribirlo en texto en la caja superior.

Por último, en el botón *Blackboard* se ha incluido una pizarra, para que el usuario, de sentirse más cómodo, o en la necesidad de expresar algo gráficamente, pueda usar unas sencillas herramientas para ello.

Aunque esta era la visión inicial de la aplicación, se decidió además incluir un asistente de traducción, y una elección de idiomas para que el usuario, además de poder comunicarse en el idioma natal, además fuese posible traducir lo que se escucha al idioma en el que el usuario se sienta más cómodo. Gracias a esta decisión, la aplicación se expande a un mayor número de usuarios.

2.1. Esquema de Color

Primary color:	#A5EAD2 #A5EAD2	#70D5B1 #70D5B1	#49BD94 #49BD94	#2CA77C #2CA77C	#0D9767 #0D9767
Secondary color (1):	#A9C9E9 #A9C9E9	#76A5D2 #76A5D2	#5085B9 #5085B9	#336BA2 #336BA2	#175693 #175693
Secondary color (2):	#CBF7AE #CBF7AE	#ABEF7F #ABEF7F	#91E759 #91E759	#79DB39 #79DB39	#59C711 #59C711

Figura 5: Esquema de Colores

2.2. Planteamiento

Para el desarrollo de la Aplicación se ha elegido el IDE de Google, Android Studio. Además, para la contrucción de la aplicación, se ha elegido el SDK 30 de Google, la última versión de Android, aunque se ha fijado una API mínima donde la aplicación funcionaría, la API v28. Esto quiere decir que la aplicación desarrollada funcionaría sin problemas en dispositivos de versión 9.0 de Android en adelante.

El por qué de esta decisión se debe a que aproximadamente, alrededor del cincuenta por ciento de los dispositivos android ya usan esta versión. Además, trabajar con versiones inferiores a esta limitaría el uso de nuevas tecnologías implementadas en Android para las versiones más recientes.

El lenguaje de programación usado para la aplicación ha sido JAVA, que si bien es conocido que Kotlin es la nueva apuesta para Android, es aun un lenguaje en auge del que hay menos información en Internet para trabajar cómodamente con él (aunque no se niega un re-desarrollo en dicho lenguaje). A esto se añade, como ya es conocido por el funcionamiento de Android, que se han usado ficheros XML para aplicar diseños a las distintas actividades y navegación posible en la aplicación.

Figura 6: JAVA

Figura 7: Android Studio

2.3. Navegacion de la Aplicación

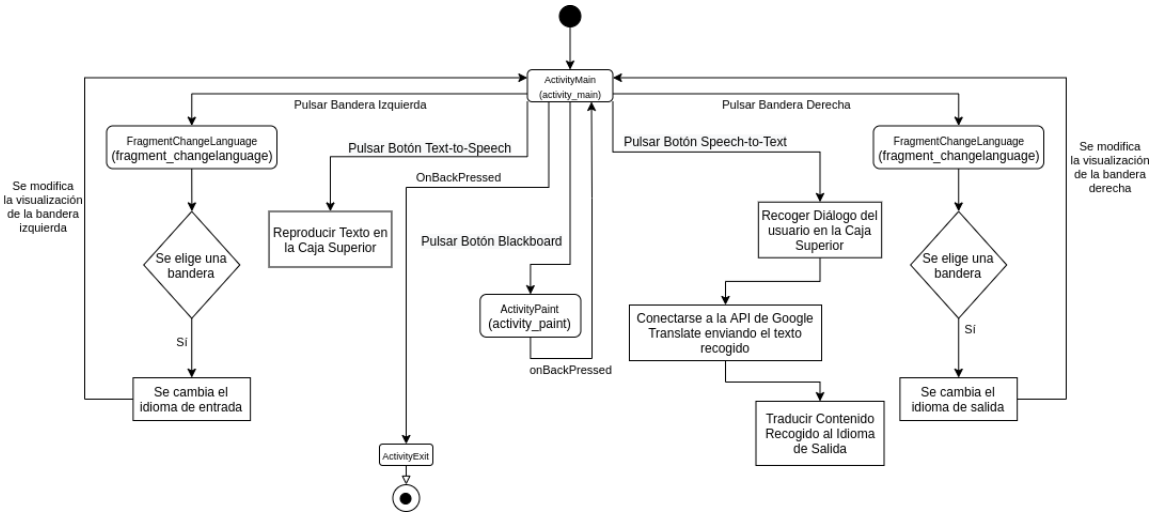

Figura 8: Diagrama de Navegacion

En la imagen anterior se puede apreciar un Diagrama de Navegación de la aplicación, detallando a grandes rasgos el recorrido que hace la aplicación desarrollada según la acción del usuario. A continuación, se van a nombrar las Actividades que componen el proyecto y se hablará más en profundidad sobre el código y la programación llevada a cabo, es decir, la implementación de la misma.

2.4. Estructura

La aplicación se trabaja únicamente en un único fichero *ActivityMain*, dónde se realiza toda la funcionalidad de traducción y de recogida de datos, así como la representación de los mismos. Cuenta con una actividad auxiliar llamada *ActivityPaint* que es la permite al usuario usar su dispositivo móvil como una pizarra. Esta última actividad si divide su funcionalidad en dos clases *Brush* y *PaintView*. Cada componente de la actividad se explicará de forma más detallada a continuación.

3. Implementación

3.1. ActivityMain

Esta es la interfaz y actividad principal, por sobre la que se gestiona todo lo referente a la funcionalidad de la aplicación.

Figura 9: Activity Main

Está compuesta por varios elementos. En la parte inferior contiene una serie de botones que permiten realizar las funciones antes mencionadas pero que se detallarán en profundidad. Contiene dos elementos imagen que representan el idioma de entrada y de salida de la aplicación, es decir, en qué idioma se ha representado lo plasmado en la Caja Superior (Bandera Izquierda), y el idioma de salida (Bandera Derecha), es decir, el idioma al que debería ser traducido lo escrito o recogido por voz. El botón central *Borrar* elimina el contenido de la Caja Superior. Y por último, pero no menos importante, la Caja Superior, el eje central de la aplicación.

Text-To-Speech

Es el botón de la izquierda de la parte inferior de la aplicación. Este botón permite reproducir mediante un agente de voz el texto expresado en la Caja Superior. Para conseguir esto se ha usado un objeto del tipo *TextToSpeech*.

```
tts = new TextToSpeech(this, new TextToSpeech.OnInitListener() {
    @Override
    public void onInit(int status) {
        // Si no hay ningun problema en la inicializacion del Text-To-Speech
        if (status != TextToSpeech.ERROR) {
            SwitchTag(out_language);
            tts.setLanguage(new Locale(localTag.first, localTag.second));

        }
    }
});
```

Una vez se inicializa, para poder reproducir lo escrito en la Caja Superior, se llama al método textToSpeech() que lo que hace es llamar al método *speak* del objeto TextToSpeech.

```
private void textToSpeech() {
    String voz = texBox.getText().toString();
    tts.speak(voz, TextToSpeech.QUEUE_FLUSH, null, null);
}
```

Blackboard

Es el botón central de la interfaz. Al pulsar sobre él la aplicación se redirecciona a otra interfaz donde aparecerá una pantalla en blanco y una gama de colores en el borde inferior que el usuario podrá elegir. Por defecto aparecerá el color negro seleccionado, aunque también tendrá la opción de pulsar sobre diversos colores. En la nueva actividad a la que el usuario es redireccionado, este podrá usarla como una pizarra simple. Sin embargo, esta funcionalidad se explicará más adelante en la sección ***ActivityPaint***.

```
fb_blackboard.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent blackboard = new Intent(getApplicationContext(), ActivityPaint.class);
        ActivityOptions animacion = ActivityOptions.makeCustomAnimation(
                getApplicationContext(), R.anim.slide_in, R.anim.slide_out);
        startActivity(blackboard, animacion.toBundle());
    }
});
```

Speech-To-Text

Es el botón de la derecha la parte inferior de la aplicación. Al mantener pulsado este botón, el dispositivo entrará en modo escucha y comenzará a captar todo lo que el usuario diga. Para realizar esta funcionalidad se ha creado un objeto del tipo *SpeechRecognizer*. A diferencia del *TextToSpeech*, esta funcionalidad contempla algo más de complejidad ya que es necesario inicializar y controlar varios valores para mantener en escucha al dispositivo.

Lo primero de todo será la inicialización del objeto, mostrada a continuación. Al crear el intent, la acción con la que se inicializa es la **(ACTION-RECOGNIZE-SPEECH)** para conseguir que el dispositivo se mantenga en segundo plano con el reconocimiento de voz activo. Un detalle añadido para permitir la opción de recoger en tiempo real lo que el usuario vaya diciendo es **(EXTRA-PARTIAL-RESULTS, true)**. Con esta opción añadida al *Intent* y modificando el método *onPartialResults* al gestionar el *Listener* del *SpeechRecognizer* se puede ir escribiendo en la Caja Superior, en tiempo real, lo que vaya captando el objeto.

```
// Inicia una actividad que escuchara al usuario y mandara los datos al speech
final Intent speechRecognizerIntent =
        new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
// Se le informa al speech sobre el modelo preferido al ejecutarse
speechRecognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
        RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
speechRecognizerIntent.putExtra(RecognizerIntent.EXTRA_LANGUAGE,
        Locale.getDefault());
```

A continuación aparecerá el código con la inicialización del *Listener* con el que se puede realizar la escucha y controlar los datos que recibe, procesarlos, y convertirlos a texto. Los cuatro métodos más importantes y que se usarán serán:

- **onBeginningOfSpeech().** Se ejecuta en cuanto se activa el *Listener* y lo hace es eliminar el texto que hubiese en la Caja Superior.
- **onError().** Si la escucha falla, se reinicia la Caja Superior.
- **onResults().** Procesa toda la información recibida durante la escucha y la transforma en texto.
- **onPartialResults().** Permite ir mostrando en la Caja Superior lo que el dispositivo va escuchando y lo procesa como texto.

Para poder usar esta funcionalidad, es necesario que el usuario conceda permisos de escucha a la aplicación. Para poder conceder estos permisos, es necesario indicar en el *AndroidManifest.XML* la siguiente línea.

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

```
// Habilitando el modo dictado se puede ir recogiendo y
//procesando (Debe usarse el metodo onPartialResults

speechRecognizerIntent.putExtra(RecognizerIntent.EXTRA_PARTIAL_RESULTS, true);
speech.setRecognitionListener(new RecognitionListener() {

    @Override
    public void onReadyForSpeech(Bundle params) {
    }

    @Override
    public void onBeginningOfSpeech() {
        texBox.setText(""); // Para eliminar lo que ya hubiese escrito
        texBox.setHint(getString(R.string.tx_escuchando));
    }

    @Override
    public void onRmsChanged(float rmsdB) {
    }

    @Override
    public void onBufferReceived(byte[] buffer) {
    }

    @Override
    public void onEndOfSpeech() {
    }

    @Override
    public void onError(int error) {
        texBox.setHint(getString(R.string.textBox_hint));
    }

    //Este es el metodo que recoge lo escuchado por el dispositivo y lo traduce
    @Override
    public void onResults(Bundle results) {
        ArrayList<String> data =
            results.getStringArrayList(SpeechRecognizer.RESULTS_RECOGNITION);
        String mensaje = traduction(data.get(0));
        texBox.setText(mensaje);
        texBox.setHint("");
    }

    @Override
    public void onPartialResults(Bundle partialResults) {
        ArrayList<String> data =
            partialResults.getStringArrayList(SpeechRecognizer.RESULTS_RECOGNITION);
        // Aqui se va escribiendo al mismo tiempo que se habla
        texBox.setText(data.get(0));
    }

    @Override
    public void onEvent(int eventType, Bundle params) {
    }
});
```

La aplicación tiene como ayuda auxiliar un botón denominado *Borrar*. Este simplemente se usa para eliminar todo el texto que pueda haber en la Caja Superior.

Por último, y para finalizar esta sección, se hablará de las dos imágenes a ambos lados en la interfaz. Estas imágenes se usán como una representación del idioma de entrada y de salida. Y es que como se explicó en la introducción, aunque esta aplicación nació con un objetivo concreto que era transmitir o recoger audio como ayuda a personas con cierta discapacidad, el funcionamiento de la aplicación se amplió un poco más, aprovechando estas cualidades para crear lo que sería un traductor, de forma que cualquier usuario que use dicha aplicación, y se encuentre con alguien de distinta nacionalidad con el que le sea complicado comunicarse, pueda apoyarse en esta aplicación para facilitar la comunicación al permitir la traducción de lo que un usuario dijo.

Figura 10: Selección Idioma

Para cambiar el idioma de entrada (Derecha) o el de salida (Izquierda), basta con pulsar sobre alguna de ellas. Esto abrirá una actividad temporal (en ambos casos la misma) y saldrá un menú deslizable con las opciones disponibles, que por el momento son Español, Inglés y Alemán. En todo momento, en la aplicación, desde el primer arranque de esta se inicializan los posibles valores de los idiomas, es decir, existe para cada bandera, una identificación asociada a un valor de idioma

(por ejemplo, la bandera de España a (spa, ES-es), de forma que cada vez que se cambia la elección actual, estos valores se actualizan. Estos valores están contemplados en la creación de las banderas en la actividad, para la que se ha creado una clase, ***Flag***, que contiene el identificador del recurso de la imagen y una cadena que identifica al idioma que representa.

```
private void initializeFlag() {
    // Se recoge el array del fichero string.xml que seran los idiomas disponibles
    List<String> language_options =
            Arrays.asList(getResources().getStringArray(R.array.language_array));
    // Spanish
    idiomsList.add(new Flag(language_options.get(0), R.drawable.flag_spanish));
    // British
    idiomsList.add(new Flag(language_options.get(1), R.drawable.flag_british));
    // German
    idiomsList.add(new Flag(language_options.get(2), R.drawable.flag_german));
}
```

Cuando se pulsa sobre la bandera se llama a la Actividad *FragmentChangeLanguage* (imagen anterior) pero, en este caso, en vez de hacerlo mediante un *Intent* como el antes explicado, se hace mediante el método *StartActivityOnResult* que al hacerlo así, mantiene en segundo plano la actividad que realiza la llamada mientras se gestiona lo necesario en la actividad llamada. Cuando la llamada finaliza, devuelve unos valores junto a un código para que la actividad principal sepa a qué está respondiendo esta. Mediante el método *onActivityResult()* se gestiona el cambio de una bandera u otra ya que al iniciar el *Intent* a la otra actividad, se le pase un código asociado que identifica que el cambio debe hacerse a la bandera de entrada o de salida.

```
image_in_language.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent inMethod =
                new Intent(ActivityMain.this, FragmentChangeLanguage.class);
        inMethod.putExtra("METHOD", REQUEST_IN_LANGUAGE);
        ActivityOptions animacion = ActivityOptions.makeCustomAnimation(
                getApplicationContext(), R.anim.fade_in, R.anim.fade_out);
        startActivityForResult(inMethod, REQUEST_IN_LANGUAGE, animacion.toBundle());
    }
});
```

3.2. ActivityPaint

En esta actividad, se ha creado una pizarra para que el usuario que sienta la necesidad, o le resulta más fácil expresarse mediante un simple dibujo, pueda usarlo para su comunicación.

Figura 11: Activity Paint

Esta actividad, por sí sola no tiene más que un simple diseño, como puede observarse, mientras que la funcionalidad de todo esto reside en dos clases auxiliares denominadas *PaintView* y *Brush*. Toda la parte blanca de la interfaz es un objeto vista del tipo *PaintView* el cual extiende sus propiedades de la clase *View*. Al hacer esto, se ha creado una clase propia *Canvas* que permite la realización de trazados de color en la pantalla y que queden expresados en esta.

PaintView

Esta clase que extiende de *View* es el eje principal de toda la pizarra. El método importante en esta clase es *onDraw()* que apoyado en el objeto de tipo *Canvas* permite hacer trazados en dicha vista y quedar almacenados. Con el objeto de tipo *Paint* se permite el dibujo en el Canvas y con objeto de tipo *Path* lo que hace es calcularse el trazado que el usuario está haciendo con su dedo (o con algún lápiz capacitivo).

```
@Override
protected void onDraw(Canvas canvas) {
    canvas.save();
    mCanvas.drawColor(DEFAULT_BG_COLOR);

    for(Brush br : paths){
        mPaint.setColor(br.getColor());
        mPaint.setStrokeWidth(br.getStrokeWidth());
        mPaint.setMaskFilter(null);
        mCanvas.drawPath(br.getRuta(), mPaint);
    }

    canvas.drawBitmap(mBitmap, 0f, 0f, mBitmapPaint);
    canvas.restore();
}
```

El siguiente método importante es *onTouchEvent()* apoyado en tres métodos privados creados para facilitar la obtención de las rutas, que serían *touchStart()*, *touchUp()* *touchMove()*, los cuales, el primero es usado para indicar el inicio del deslizamiento por la pantalla y hacer una localización de las coordenadas, y el segundo permite seguir el trazo de este deslizamiento, calculando cuánto se ha desplazado desde la posición anterior. El último sería el que marcaría el fin de dicho deslizamiento por la pantalla y calcularía la ruta total.

```
private void touchStart(float x, float y){
    mPath = new Path();
    Brush br = new Brush(currentColor, size, mPath);
    paths.add(br);
    mPath.reset();
    mPath.moveTo(x, y);
    mX = x;
    mY = y;
}
```

```
private void touchUp(){
    mPath.lineTo(mX, mY);
}
```

```
private void touchMove(float x, float y){
    float dx = Math.abs(x-mX);
    float dy = Math.abs(y-mY);

    if (dx >= TOUCH_TOLERANCE || dy >= TOUCH_TOLERANCE){
        mPath.quadTo(mX, mY, (x+mX)/2, (y+mY)/2);
        mX = x;
        mY = y;
    }
}
```

Todos estos métodos unen su fucionalidad en un cuarto y último método que permite hacer representar el trazado en el Canvas antes mencionado para que conforme el usuario dibuje esto se muestre en la pantalla. Este método es *onTouchEvent* que detecta la acción del usuario (pulsar la pantalla, mantenerla pulsada mientras se desliza por ella, y dejar de pulsarla).

```
@Override
    public boolean onTouchEvent(MotionEvent event) {
        float x = event.getX();
        float y = event.getY();

        switch (event.getAction()){
            case MotionEvent.ACTION_DOWN:
                touchStart(x, y);
                invalidate();
            break;

            case MotionEvent.ACTION_MOVE:
                touchMove(x, y);
                invalidate();
            break;

            case MotionEvent.ACTION_UP:
                touchUp();
                invalidate();
            break;

        }
        return true;
    }
```

Brush

Esta clase simplemente se usa para crear un pincel con el que dibujar al que se le da un color (que puede ir actualizándose), el grosor con el que se dibujará en la pantalla y la ruta que realiza este, con la cual es fácil realizar el recorrido en tiempo real.

3.3. Translate

Por último, en esta sección se va a explicar un poco el proceso de traducción implementado en la aplicación usando la API de Google, *Cloud Translation*.

Antes de poder realizar este proceso y poder usar un objeto de la clase ***Translate*** ha sido necesario realizar una serie de pasos previos. El primero de estos pasos ha sido crear una cuenta en Google Cloud e iniciar (al menos en este proyecto educativo) una cuenta gratuita, ya que el servicio de **Google Translate** es de pago, pero, Google tiene la posibilidad de abrir una cuenta gratuita de prueba para probar sus servicios de pago.

Una vez se ha realizado este proceso, se debe de activar la API de Cloud Translate en la cuenta de Google Cloud previamente abierta, y obtener una KEY que se deberá de implementar en el proyecto para que sea posible conectarse con los servicios de Google y así usarlos.

Con todo este proceso realizado, lo siguiente que se deberá hacer para poder usar el objeto de tipo *Translate* de Google es importar los paquetes desde Google Cloud:

```
implementation('com.google.cloud:google-cloud-translate:1.12.0') {
    exclude group: 'org.apache.httpcomponents'
    exclude group: 'org.json', module: 'json'
}
annotationProcessor 'com.google.cloud:google-cloud-translate:1.12.0'
```

Lo siguiente es indicar en el proyecto que la aplicación va a necesitar conectarse a Internet, por lo que es necesario pedir los permisos necesarios para poder realizar la conexión a internet. Esto se indica en el Manifest:

```
<!-- Para poder comprobar si existe una conexion a internet -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Con todos estos pasos realizados, ahora es posible crear un objeto de tipo *Translate* en el proyecto y pasar a inicializarlo, conectarlo a los servidores de Google y comenzar a realizar peticiones de traducción.

Mediante el método privado creado ***getTranslateService()*** se inicializan los servicios y el objeto usado para la traducción.

```
// Obtiene una configuraci n del servicio de traduccion
private void getTranslateService() {
    StrictMode.ThreadPolicy policy =
            new StrictMode.ThreadPolicy.Builder().permitAll().build();
    StrictMode.setThreadPolicy(policy);

    try (InputStream is =
            getResources().openRawResource(R.raw.hellotome314111d64f73544295)) {

        //Get credentials:
        final GoogleCredentials myCredentials = GoogleCredentials.fromStream(is);

        //Set credentials and get translate service:
        TranslateOptions translateOptions =
            TranslateOptions.newBuilder().setCredentials(myCredentials).build();

        translate = translateOptions.getService();

    } catch (IOException ioe) {
        ioe.printStackTrace();

    }
}
```

Por último, para facilitar la llamada a la traducción una vez que se ha inicializado todo correctamente, se ha creado un método privado ***translate()*** que recibe como parámetro el texto a traducir y devuelve la traducción asociada. En el atributo *targetLanguage* se indica *localTag* que aunque no ha sido mencionada directamente hasta ahora, es el grupo **PAIR** que guarda el idioma actual al que se debe traducir el texto. Esta variable se actualiza cuando se cambia la bandera de salida.

```
private String translate(String texto) {
    // El texto que se va a traducir
    Translation translation =
        translate.translate(texto,
        Translate.TranslateOption.targetLanguage(localTag.first),
        Translate.TranslateOption.model("base"));

    return translation.getTranslatedText();

}
```

Tal y como antes se observó en el código de *onResults*, en este se llamaba a un método denominado *traduction()*. Ahora puede hacerse alusión a él. Este método llama a todo lo anterior antes explicado para conectar a los servidores de Google, y realizar la traducción, siempre y cuando el dispostivo tenga conexión a internet, se lo contrario no será posible realizar la traducción.

```
private String traduction(String mensaje) {
    // Se traduce de un idioma de entrada (in_language) a otro de salida (out_language)
    if(checkInternet()){
        getTranslateService();
        mensaje = translate(mensaje);
    }else{
        toastAdvice(getString(R.string.not_internet_translation));
    }

    return mensaje;
}
```

3.4. Sensor de Proximidad

Con el revuelo de estos últimos años debido a la Pandemia que se ha vivido, se ha pensado en una capacidad auxiliar para añadir a la aplicación, y es que sea posible usarla mediante proximidad. Es decir, se ha hecho uso del ***Sensor de Proximidad*** para que sea posible hacer que el telefono entre en modo escucha sin ser necesario tocar el botón de la pantalla. Esto también añade un plus de familiarización con la aplicación ya que bastaría con poner la mano frente al dispositivo para recoger lo que el usuario quiera decir. Lo primero necesario para conseguir esto, es comprobar que el dispositivo en el que se ejecuta la aplicación dispone de dicho sensor.

```
<!-- Para poder usar el sensor de proximidad -->
<uses-feature android:name="android.hardware.sensor.proximity" />
```

```
SensorManager sensorMang;
Sensor sensor;
SensorEventListener sensorEvtList;

...
protected void onCreate()... {

    ...
    sensorMang = (SensorManager) getSystemService(SENSOR_SERVICE);
    ...

    // Tratamiento del SENSOR
    sensor = sensorMang.getDefaultSensor(Sensor.TYPE_PROXIMITY);

    if (sensor != null) {
        // Sirve para detectar si nos hemos alejado o acercado al dispositivo
        sensorEvtList = new SensorEventListener() {
            @Override
            public void onSensorChanged(SensorEvent event) {
                // Si estamos en el ratio de detecci n del sensor
                if (event.values[0] < sensor.getMaximumRange()) {

                    checkAudioPermissions();
                    speech.startListening(speechRecognizerIntent);
                } else {
                    // Al alejarnos termina la escucha
                    speech.stopListening();
                }
            }

            @Override
            public void onAccuracyChanged(Sensor sensor, int accuracy) {

            }
        };
    }

    startSensor();
}
```

4. Bibliografia

Referencias

[1] Android Developers
https://developer.android.com/

[2] Google Cloud
https://cloud.google.com/translate/

[3] GeekForGeeks
https://www.geeksforgeeks.org/

[4] Stackoverflow
https://stackoverflow.com/

[5] Medium
https://medium.com/@yeksancansu/

[6] Color Schema Paletton
https://paletton.com/

[7] Iconify Design
https://iconify.design/icon-sets/