Aplicación Aprendizaje de Idiomas con Kotlin

Proyecto final de ciclo formativo superior “Desarrollo de aplicaciones multiplataforma”

Iván Ruiz Pedrós

Tutor del proyecto: Jorge Orta Lopez
Centro: CIP FP Batoi

CONTENIDO

Introducción

Comencé el proyecto sin tener mucha idea sobre lo que hacer ya que comencé las prácticas antes de poder elegirlo, por lo que tuve que pensar en algo sin tener en cuenta lo que fuera a hacer en ellas.
Como me interesaban las aplicaciones y el tema de los idiomas se me ocurrió hacer una aplicación de traducción con uso de IA, niveles, etc

Al final no me centré mucho en el tema “jugabilidad” de la aplicación del apartado ejercicios interactivos y más en las funciones especificadas en la propuesta del proyecto.

El objetivo principal de la aplicación es ser una aplicación funcional para traducir en tiempo real con la cámara, aprender idiomas y tener una aplicación de traducción sencilla para usar

Descripción general

El proyecto se ha desarrollado siguiendo una arquitectura modular basada en MVVM y Clean Architecture. La aplicación móvil está implementada en Kotlin para Android e integra:

- Google ML Kit para el reconocimiento de texto (OCR)
- Whisper de OpenAI para transcribir voz a texto
- API de traducción para convertir el texto reconocido a otros idiomas
- TextToSpeech de Android para la síntesis de voz del texto traducido
- Firebase (Firestore/Realtime Database) para almacenar y sincronizar en tiempo real el progreso y los datos del usuario

De este modo, toda la lógica de presentación, dominio y datos queda correctamente separada en distintos módulos, facilitando el mantenimiento y la escalabilidad de la aplicación.

Herramientas y tecnologías

En el desarrollo de esta aplicación he integrado múltiples herramientas y tecnologías modernas, centradas en la inteligencia artificial, procesamiento de texto, traducción automática y diseño de interfaces modernas. A continuación, se describen las principales:

Jetpack Compose y AndroidX

Jetpack Compose es el framework moderno de UI de Android basado en funciones **@Composable**, que permite construir interfaces declarativas, dinámicas y reactivas. Se ha usado para toda la interfaz de usuario, incluyendo menús, ejercicios interactivos y controles de cámara.

androidx.compose.ui, **material3**, **ui-tooling-preview:** Proveen los componentes visuales y herramientas de diseño necesarias para construir interfaces modernas, con soporte de Material Design 3.

androidx.activity:activity-compose: Permite integrar Jetpack Compose con el ciclo de vida de las Activities.

androidx.core:core-ktx y **androidx.lifecycle:lifecycle-runtime-ktx:** Facilitan el acceso a funciones del sistema y la gestión del ciclo de vida con Kotlin, especialmente útil al trabajar con corutinas y componentes como ViewModels.

Firebase

Firebase actúa como backend principal para autenticación, almacenamiento de datos y analítica:

firebase-auth-ktx: Se utiliza para permitir el registro e inicio de sesión de usuarios mediante correo electrónico y contraseña.

firebase-database-ktx y **firebase-database:** Permiten almacenar traducciones realizadas, progreso del usuario y configuraciones de idioma en tiempo real.

firebase-analytics: Se emplea para recolectar estadísticas sobre el uso de la aplicación, como qué tipo de ejercicios se realizan más, qué idiomas se utilizan, entre otros.

Gracias a Firebase, se evita la necesidad de crear un servidor propio, permitiendo una integración rápida, escalable y segura.

CameraX y AR

Para las funciones de reconocimiento de texto desde imágenes y realidad aumentada, se integraron:

androidx.camera:camera-core, **camera-camera2**, **camera-view**, **camera-lifecycle:** Estas librerías simplifican el acceso a la cámara, permitiendo capturar imágenes o analizarlas en tiempo real sin lidiar directamente con la complejidad de la API Camera2.

com.google.ar.sceneform:sceneform: Se utilizó en ciertas versiones para mostrar objetos virtuales, lo cual es útil en ejercicios interactivos de vocabulario con modelos 3D.

ML Kit (Machine Learning on-device)

La aplicación integra varios módulos de ML Kit, que permiten realizar tareas de inteligencia artificial directamente en el dispositivo, sin requerir conexión a internet:

text-recognition: Se usa para detectar y extraer texto de imágenes capturadas por la cámara, permitiendo traducir automáticamente carteles, libros o etiquetas.

language-id: Detecta el idioma del texto ingresado o capturado, permitiendo adaptar dinámicamente el modelo de traducción.

translate: Permite traducir texto automáticamente entre varios idiomas. Este componente puede descargar modelos localmente, funcionando sin conexión.

object-detection y **object-detection-custom:** Se pueden utilizar para reconocer objetos dentro de imágenes y relacionarlos con sus nombres en diferentes idiomas, lo que es útil para crear ejercicios de vocabulario visual.

Text-to-Speech

Text-to-Speech (TTS): Se emplea la API nativa de Android para convertir el texto traducido en voz, lo cual permite a los usuarios escuchar la pronunciación del contenido en el idioma de destino.

Whisper de OpenAI

Whisper (OpenAI): Se ha integrado como sistema de reconocimiento de voz (ASR), permitiendo que los usuarios puedan practicar pronunciación oral de forma precisa. Whisper detecta y transcribe la voz del usuario en texto, que luego se compara con el texto objetivo para verificar la pronunciación.

Retrofit y OkHttp

Retrofit: Se usa como cliente HTTP para consumir APIs externas, como puede ser la integración con servicios de traducción o procesamiento adicional de texto.

Gson Converter: Convierte automáticamente JSON en objetos Kotlin, simplificando la manipulación de respuestas desde servidores externos.

OkHttp: Se utiliza como capa de bajo nivel para gestionar peticiones HTTP, gestionar headers, loggers, o interceptores si es necesario para autenticación o análisis.

Corutinas de Kotlin

kotlinx-coroutines-android y **kotlinx-coroutines-play-services:** Permiten la ejecución asíncrona de tareas pesadas como traducciones, reconocimiento de texto o llamadas a API, sin bloquear la interfaz del usuario. Además, integran compatibilidad con Firebase Tasks para convertirlos fácilmente en funciones suspend.

Este conjunto de tecnologías me ha permitido crear una aplicación accesible y educativa, que combina visión artificial, traducción automática, interacción por voz y cámara, todo con una arquitectura moderna basada en Kotlin y Jetpack Compose.

Desarrollo de la app con Kotlin

En este apartado voy a explicar cómo he desarrollado la aplicación, su arquitectura, y los principios de diseño utilizados. La aplicación se ha diseñado siguiendo el paradigma de programación modular, la arquitectura Clean Architecture y el patrón MVVM.

Lenguaje Kotlin

Kotlin fué creado en 2010 por JetBrains, pero su popularidad es reciente, desde que en 2017 Google publicó que pasaría a ser lenguaje oficial para Android y que comenzaría a darle soporte, tal fué el auge, que actualmente la mayoría de aplicaciones Android se desarrollan con él, siendo el lenguaje favorito de los desarrolladores del sector.

Esto se debe en gran parte a las siguientes puntos:

- **Curva de aprendizaje fácil:** sintaxis del código simple.
- **Tiempo de programación reducido:** elimina el código redundante.
- **Multiplataforma:** principalmente se ejecuta en JVM, pero también compila a JavaScript, o código nativo a través de LLVM.
- **Semántica:** además de clases y métodos de la programación orientada a objetos, Kotlin soporta programación por procedimientos y uso de funciones.

Programación Modular

La programación modular divide la aplicación en módulos independientes pero interconectados- En el desarrollo de aplicaciones Android con Kotlin, esto se logra creando múltiples módulos Gradle.

Ventajas de la Programación Modular:

- **Mantenibilidad:** Cada módulo es más pequeño y manejable, facilitando el mantenimiento y la actualización del código.
- **Reusabilidad:** Los módulos pueden ser reutilizados en diferentes partes de la aplicación o incluso en diferentes proyectos.
- **Escalabilidad:** Facilita el trabajo en equipos grandes, permitiendo que los desarrolladores trabajen en diferentes módulos de manera independiente.

- **Rendimiento:** Mejora los tiempos de compilación, ya que solo los módulos modificados necesitan ser compilados.

Código de la app

Ahora que ya tenemos una base del funcionamiento de las herramientas y de Kotlin ahora vamos a pasar a la explicación detallada del código de las ventanas y demás para que funcione correctamente la aplicación.

Main Menu

Este es el menú principal utilizando Jetpack Compose para la interfaz y Firebase para la gestión de datos

Variables y constantes

```
companion object {
    const val EXTRA_AUTO_NEXT = "auto_next"
    private val PREFS_NAME = "language_prefs"
    private val KEY_SELECTED_LANG = "selected_lang"
}

private val PERMISSIONS = arrayOf(
    Manifest.permission.CAMERA,
    Manifest.permission.RECORD_AUDIO
)

private var showPermissionDialog by mutableStateOf( value: false)
private var shouldFinishActivity by mutableStateOf( value: false)
private var permissionsGranted by mutableStateOf( value: false)

private val requestPermissionLauncher = registerForActivityResult(
    ActivityResultContracts.RequestMultiplePermissions()
) { perms ->
    permissionsGranted = perms.all { it.value }
    if (!permissionsGranted) showPermissionDialog = true
}

private val userId: String by lazy {
    Settings.Secure.getString(contentResolver, Settings.Secure.ANDROID_ID)
}
```

PERMISSIONS: Permisos requeridos (cámara y micrófono).
Estado de compose:

- **showPermissionDialog:** variable para controlar el diálogo de permisos.
- **shouldFinishActivity:** variable para controlar el cierre de la actividad.
- **permissionsGranted:** variable para controlar el estado de permisos.

requestPermissionLauncher: Usa **ActivityResultContracts** para permisos en tiempo real.
userId: Crea la variable para crear el id del usuario automáticamente(**ANDROID_ID**), la crea perezosamente (**lazy**)

onCreate

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    FirebaseApp.initializeApp( context: this)
    checkPermissions()
    enableEdgeToEdge()

    setContent {
        ProyectoFinalTheme {
            var selectedLang by remember { mutableStateOf(loadSelectedLanguage()) }

            if (showPermissionDialog) {
                PermissionDeniedDialog(
                    onDismiss = { showPermissionDialog = false },
                    onGoToSettings = { openAppSettings() },
                    onExitApp = { shouldFinishActivity = true }
                )
            }
            if (shouldFinishActivity) {
                LaunchedEffect(Unit) { finish() }
            }

            Scaffold(modifier = Modifier.fillMaxSize()) {...}
        }
    }
}
```

FirebaseApp.initializeApp: Inicializa el SDK.
checkPermissions: Verifica los permisos.
enableEdgeToEdge: Habilita pantalla completa.

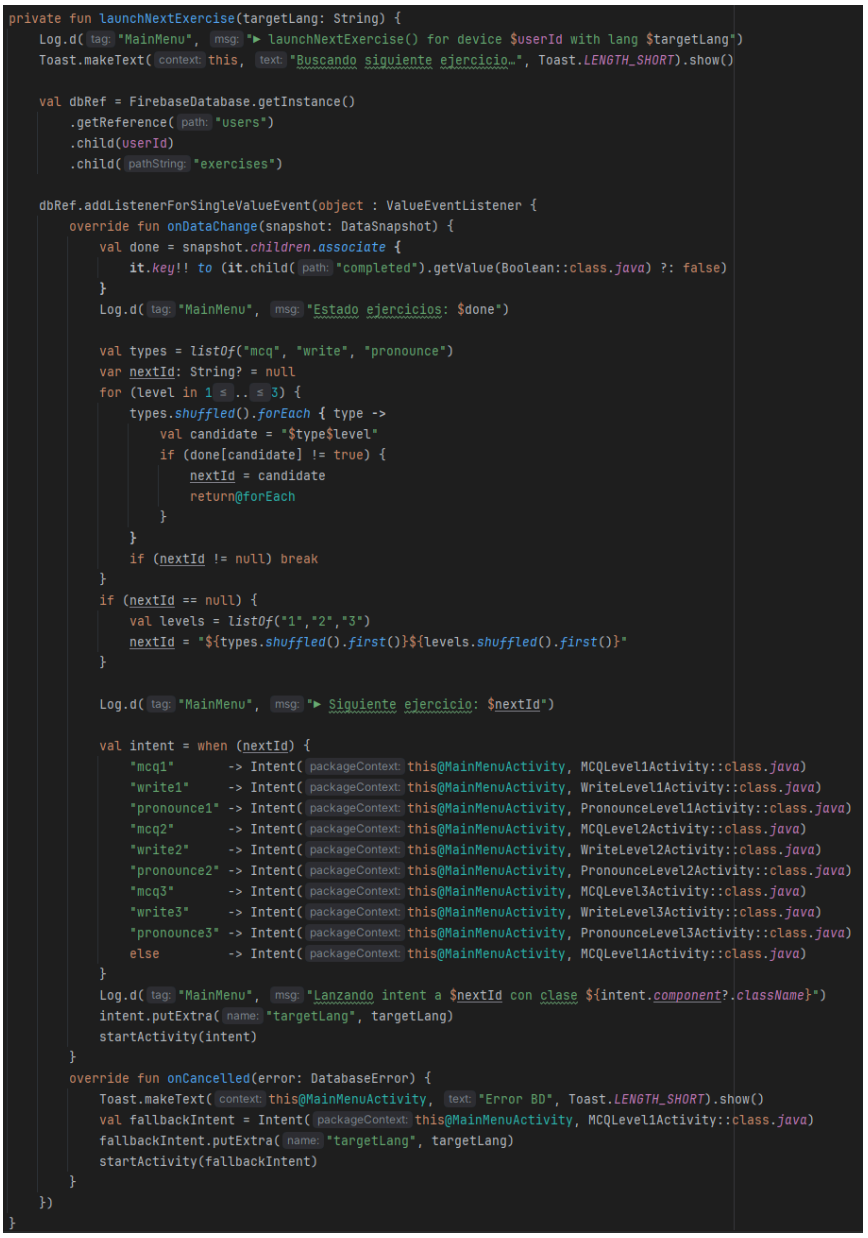
Scaffold

```
Scaffold(modifier = Modifier.fillMaxSize()) { padding ->
    Column(
        Modifier
            .padding(padding)
            .fillMaxSize(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(
            text: "Aprendizaje de Idiomas",
            style = MaterialTheme.typography.headlineLarge,
            fontWeight = FontWeight.Bold,
            fontSize = 24.sp
        )
        Spacer(Modifier.height(20.dp))

        MenuButton( text: "Reconocimiento de texto", {
            startActivity(Intent( packageContext: this@MainMenuActivity, TextRecognitionActivity::class.java))
        }, permissionsGranted)
        Spacer(Modifier.height(20.dp))
        MenuButton( text: "Detección de objetos", {
            startActivity(Intent( packageContext: this@MainMenuActivity, ObjectDetectionActivity::class.java))
        }, permissionsGranted)
        Spacer(Modifier.height(20.dp))
        MenuButton( text: "Reconocimiento de Voz", {
            startActivity(Intent( packageContext: this@MainMenuActivity, WhisperActivity::class.java))
        }, permissionsGranted)
        Spacer(Modifier.height(20.dp))
        MenuButton( text: "Text-To-Speech", {
            startActivity(Intent( packageContext: this@MainMenuActivity, TTSActivity::class.java))
        }, permissionsGranted)
        Spacer(Modifier.height(20.dp))
        MenuButton( text: "Ejercicios", {
            launchNextExercise(selectedLang)
        }, permissionsGranted)
        Spacer(Modifier.height(10.dp))

        Text( text: "Idioma a aprender:", fontSize = 16.sp)
        Spacer(Modifier.height(8.dp))

        LanguageSpinner(selectedLang) {
            selectedLang = it
            saveSelectedLanguage(it)
        }
        Spacer(Modifier.height(40.dp))

        MenuButton( text: "Verificar permisos", {
            checkPermissions()
        }, enabled: true)
    }
}
```

permissionGranted: Si los permisos no están concedidos el botón estará deshabilitado.
LanguageSpinner: Selección del idioma a aprender.
saveSelectedLanguage: Función que almacena en sharedPreferences el lenguaje a aprender.

```
private fun saveSelectedLanguage(lang: String) {
    getSharedPreferences(PREFS_NAME, MODE_PRIVATE)
        .edit()
        .putString(KEY_SELECTED_LANG, lang)
        .apply()
}
```

launchNextExercise



```
private fun launchNextExercise(targetLang: String) {
    Log.d( tag: "MainMenu",  msg: "▶ launchNextExercise() for device $userId with lang $targetLang")
    Toast.makeText( context: this,  text: "Buscando siguiente ejercicio…", Toast.LENGTH_SHORT).show()

    val dbRef = FirebaseDatabase.getInstance()
        .getReference( path: "users")
        .child(userId)
        .child( pathString: "exercises")

    dbRef.addListenerForSingleValueEvent(object : ValueEventListener {
        override fun onDataChange(snapshot: DataSnapshot) {
            val done = snapshot.children.associate {
                it.key!! to (it.child( path: "completed").getValue(Boolean::class.java) ?: false)
            }
            Log.d( tag: "MainMenu",  msg: "Estado ejercicios: $done")

            val types = listOf("mcq", "write", "pronounce")
            var nextId: String? = null
            for (level in 1 ≤ .. ≤ 3) {
                types.shuffled().forEach { type ->
                    val candidate = "$type$level"
                    if (done[candidate] != true) {
                        nextId = candidate
                        return@forEach
                    }
                }
                if (nextId != null) break
            }
            if (nextId == null) {
                val levels = listOf("1","2","3")
                nextId = "${types.shuffled().first()}${levels.shuffled().first()}"
            }

            Log.d( tag: "MainMenu",  msg: "▶ Siguiente ejercicio: $nextId")

            val intent = when (nextId) {
                "mcq1"       -> Intent( packageContext: this@MainMenuActivity, MCQLevel1Activity::class.java)
                "write1"     -> Intent( packageContext: this@MainMenuActivity, WriteLevel1Activity::class.java)
                "pronounce1" -> Intent( packageContext: this@MainMenuActivity, PronounceLevel1Activity::class.java)
                "mcq2"       -> Intent( packageContext: this@MainMenuActivity, MCQLevel2Activity::class.java)
                "write2"     -> Intent( packageContext: this@MainMenuActivity, WriteLevel2Activity::class.java)
                "pronounce2" -> Intent( packageContext: this@MainMenuActivity, PronounceLevel2Activity::class.java)
                "mcq3"       -> Intent( packageContext: this@MainMenuActivity, MCQLevel3Activity::class.java)
                "write3"     -> Intent( packageContext: this@MainMenuActivity, WriteLevel3Activity::class.java)
                "pronounce3" -> Intent( packageContext: this@MainMenuActivity, PronounceLevel3Activity::class.java)
                else         -> Intent( packageContext: this@MainMenuActivity, MCQLevel1Activity::class.java)
            }
            Log.d( tag: "MainMenu",  msg: "Lanzando intent a $nextId con clase ${intent.component?.className}")
            intent.putExtra( name: "targetLang", targetLang)
            startActivity(intent)
        }
        override fun onCancelled(error: DatabaseError) {
            Toast.makeText( context: this@MainMenuActivity,  text: "Error BD", Toast.LENGTH_SHORT).show()
            val fallbackIntent = Intent( packageContext: this@MainMenuActivity, MCQLevel1Activity::class.java)
            fallbackIntent.putExtra( name: "targetLang", targetLang)
            startActivity(fallbackIntent)
        }
    })
}
```

done: Obtiene los ejercicios completados por el usuario en la base de datos de firebase.
level in 1..3: Busca el primer ejercicio sin completar (niveles 1-3, tipos aleatorios)
nextId == null: Si todos los ejercicios están completados pone uno random entre todos los niveles y tipos.
startActivity(intent): Lanza la actividad correspondiente con el idioma seleccionado.

Compose

```kotlin
@Composable
fun MenuButton(text: String, onClick: () -> Unit, enabled: Boolean) {
    Button(onClick = onClick, modifier = Modifier.padding(horizontal = 40.dp), enabled = enabled) {
        Text(text, fontSize = 18.sp)
    }
}

@Composable
fun PermissionDeniedDialog(
    onDismiss: () -> Unit,
    onGoToSettings: () -> Unit,
    onExitApp: () -> Unit
) {
    AlertDialog(
        onDismissRequest = onDismiss,
        title = { Text( text: "Permisos requeridos") },
        text = { Text( text: "La app necesita cámara y micrófono.") },
        confirmButton = { Button(onClick = onGoToSettings) { Text( text: "Abrir Ajustes") } },
        dismissButton = { Button(onClick = onExitApp) { Text( text: "Salir") } }
    )
}

@Composable
fun LanguageSpinner(selectedLang: String, onLangSelected: (String) -> Unit) {
    val context = LocalContext.current
    val languageCodes = remember { TranslateLanguage.getAllLanguages().toList() }

    val displayNames = remember(languageCodes) {
        languageCodes.map { code ->
            Locale(code).getDisplayName(Locale.getDefault()).ifEmpty { code }
        }
    }

    AndroidView(
        factory = {...},
        modifier = Modifier
            .fillMaxWidth( fraction: 0.6f)
            .height(60.dp),
        update = { spinner ->
            val newIndex = languageCodes.indexOf(selectedLang)
            if (newIndex >= 0 && newIndex != spinner.selectedItemPosition) {
                spinner.setSelection(newIndex)
            }
        }
    )
}

@Preview(showBackground = true)
@Composable
fun MainMenuPreview() {
    ProyectoFinalTheme {
        Column(
            Modifier.fillMaxSize(),
            verticalArrangement = Arrangement.Center,
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            MenuButton( text: "Ejercicios", {}, enabled = true)
        }
    }
}
```

Funciones para personalizar los componentes del compose, en este caso he editado **MenuButton**, **PermissionDeniedDialog**, **LanguageSpinner** y **MainMenuPreview**.

Auxiliares

```
private fun checkPermissions() {
    val toRequest = PERMISSIONS.filter {
        ContextCompat.checkSelfPermission( context: this, it) != PackageManager.PERMISSION_GRANTED
    }.toTypedArray()
    if (toRequest.isNotEmpty()) {
        requestPermissionLauncher.launch(toRequest)
    } else {
        permissionsGranted = true
        showPermissionDialog = false
    }
}

private fun openAppSettings() {
    startActivity(
        Intent(Settings.ACTION_APPLICATION_DETAILS_SETTINGS,
            Uri.fromParts( scheme: "package", packageName,  fragment: null))
    )
}
```

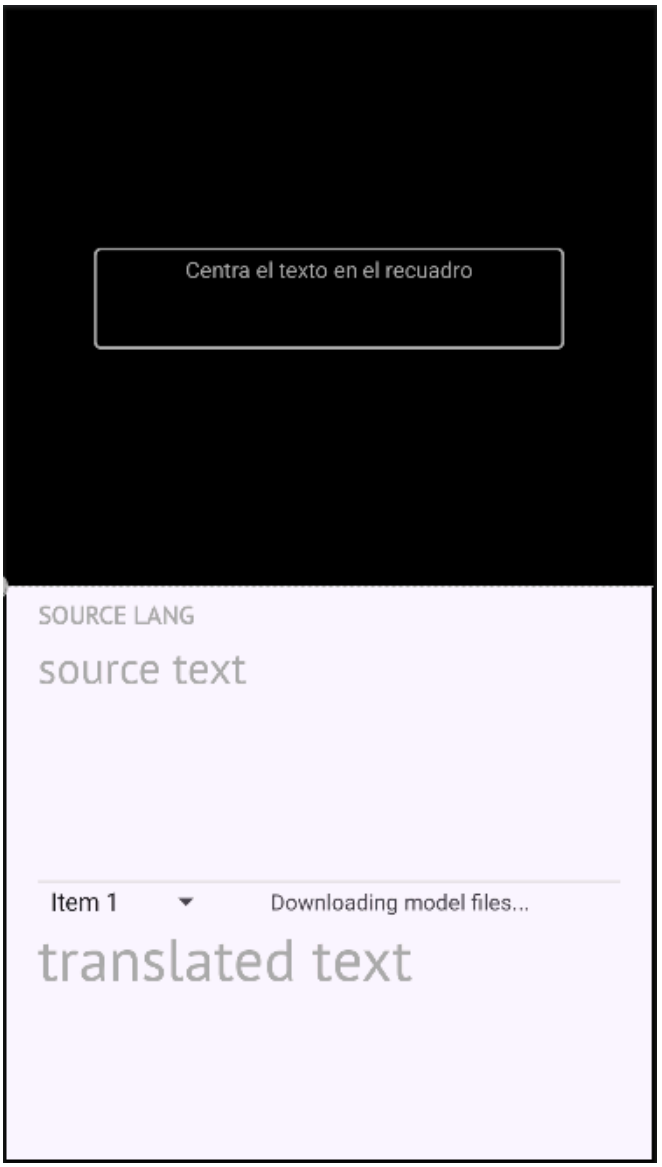
```
private fun saveSelectedLanguage(lang: String) {
    getSharedPreferences(PREFS_NAME, MODE_PRIVATE)
        .edit()
        .putString(KEY_SELECTED_LANG, lang)
        .apply()
}

private fun loadSelectedLanguage(): String {
    return getSharedPreferences(PREFS_NAME, MODE_PRIVATE)
        .getString(KEY_SELECTED_LANG,  defValue: "en") ?: "en"
}
```

checkPermissions: Confirma que los permisos han sido aceptados, si han sido aceptados cambia la variable **permissionsGranted** a true.
openAppSettings: En caso de estar **permissionsGranted** en false automáticamente abre los ajustes del dispositivo para cambiar los permisos (son necesarios).
saveSelectedLanguage: Guarda en SharedPreferences el idioma seleccionado en el spinner.
loadSelectedLanguage: Recoge de SharedPreferences el idioma seleccionado previamente en el spinner.

Text Recognition

Este apartado ha sido uno de los que más me ha costado junto con el de object detection ya que conseguir que detectara correctamente el texto de la cámara y justo donde quería se ha complicado bastante, de primeras detectaba letras raras apuntara donde apuntara ya que no quería que detectara el texto de toda la cámara sino de un pequeño recuadro me ha costado más de lo esperado.

Interfaz



El recuadro negro es la cámara.
En el centro de la cámara aparece un recuadro indicando que coloques el texto dentro de la caja.
En la parte baja tenemos un recuadro típico de traducción, con el apartado del texto a traducir y el texto traducido.

En el Spinner que ahora pone “Item 1” al iniciar la aplicación se carga una lista con todos los idiomas a los que se pueden traducir.

Es una interfaz sencilla por lo que no hay mucho más que explicar, ahora nos vamos a centrar en el código que hace funcionar la ventana.

Variables y constantes

```
private lateinit var binding: ActivityTextRecognitionBinding
private val textRecognizer by lazy {
    TextRecognition.getClient(TextRecognizerOptions.DEFAULT_OPTIONS)
}
private val languageIdentifier by lazy {
    LanguageIdentification.getClient()
}
private var translator: Translator? = null

private val langs = TranslateLanguage.getAllLanguages()
private var lastDetectedText: String? = null
private var lastUpdateTime = 0L
private val MIN_UPDATE_INTERVAL = 1500L

companion object {
    private const val REQUEST_CAMERA_PERMISSION = 1002
}
```

binding: Enlaza la vista con View Binding (UI)
textRecognizer by lazy: Cliente de ML Kit para reconocimiento de texto inicializado perezosamente(lazy)
languageIdentifier by lazy: Cliente de ML Kit para identificar el idioma del texto reconocido por la cámara
translator: Cliente para traducción
langs: Lista de idiomas soportados para traducción
lastDetectedText: Almacena el último texto detectado para evitar reprocesamiento
MIN_UPDATE_INTERVAL: Tiempo mínimo entre actualizaciones (1.5s)
REQUEST_CAMERA_PERMISSION: Código para solicitud de permiso de cámara

Funciones

onCreate

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityTextRecognitionBinding.inflate(layoutInflater)
    setContentView(binding.root)

    val displayNames = langs.map { code ->
        Locale(code).getDisplayName(Locale.getDefault()).ifEmpty { code }
    }
    ArrayAdapter( context: this, android.R.layout.simple_spinner_item, displayNames).also { adapter ->
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
        binding.targetLangSelector.adapter = adapter
    }

    binding.targetLangSelector.onItemSelectedListener =
        object : AdapterView.OnItemSelectedListener {
            override fun onItemSelected(parent: AdapterView<*>, view: View?, pos: Int, id: Long) {
                lastDetectedText?.let { translateText(it) }
            }
            override fun onNothingSelected(parent: AdapterView<*>) {}
        }

    if (allPermissionsGranted()) startCamera()
    else ActivityCompat.requestPermissions(
        activity: this, arrayOf(Manifest.permission.CAMERA), REQUEST_CAMERA_PERMISSION
    )
}
```

El onCreate configura la ventana, configura el spinner para que tenga todos los idiomas traducibles, también maneja el que si se cambia el idioma al que traducir que automáticamente se actualice el texto traducido y por último verifica/solicita los permisos de cámara.

startCamera

```
private fun startCamera() {
    ProcessCameraProvider.getInstance( context: this).apply {
        addListener({
            val cameraProvider = get()
            val preview = Preview.Builder().build().also {
                it.setSurfaceProvider(binding.viewfinder.surfaceProvider)
            }
            val analysis = ImageAnalysis.Builder()
                .setBackpressureStrategy(STRATEGY_KEEP_ONLY_LATEST)
                .build().also {
                    it.setAnalyzer(ContextCompat.getMainExecutor( context: this@TextRecognitionActivity)) { proxy ->
                        analyzeImage(proxy)
                    }
                }
            cameraProvider.unbindAll()
            cameraProvider.bindToLifecycle(
                 lifecycleOwner: this@TextRecognitionActivity,
                androidx.camera.core.CameraSelector.DEFAULT_BACK_CAMERA,
                preview, analysis
            )
        }, ContextCompat.getMainExecutor( context: this@TextRecognitionActivity))
    }
}
```

preview: se muestra la vista previa de la cámara.
ImageAnalysis: configura el analizador de frames para luego llamar a **analyzeImage()** con cada frame.
Con el uso de **STRATEGY_KEEP_ONLY_LATEST** se asegura que solo se procese el frame más reciente.

analyzeImage

```
@androidx.annotation.OptIn(ExperimentalGetImage::class)
private fun analyzeImage(imageProxy: ImageProxy) {
    imageProxy.image?.let { media ->
        val img = InputImage.fromMediaImage(media, imageProxy.imageInfo.rotationDegrees)

        textRecognizer.process(img)
            .addOnSuccessListener { visionText ->
                val imgW = img.width.toFloat()
                val imgH = img.height.toFloat()

                val locVF = IntArray( size: 2)
                binding.viewfinder.getLocationOnScreen(locVF)
                val vfLeft = locVF[0].toFloat()
                val vfTop  = locVF[1].toFloat()
                val vfW = binding.viewfinder.width.toFloat()
                val vfH = binding.viewfinder.height.toFloat()

                val scaleX = vfW / imgW
                val scaleY = vfH / imgH

                val locG = IntArray( size: 2)
                binding.textGuideBox.getLocationOnScreen(locG)
                val scanBox = RectF(
                    locG[0].toFloat(),
                    locG[1].toFloat(),
                     right: locG[0] + binding.textGuideBox.width.toFloat(),
                     bottom: locG[1] + binding.textGuideBox.height.toFloat()
                )

                val mappedBlocks = visionText.textBlocks.mapNotNull { blk ->
                    blk.boundingBox?.let { bbImg ->
                        val left   = vfLeft + bbImg.left   * scaleX
                        val top    = vfTop  + bbImg.top    * scaleY
                        val right  = vfLeft + bbImg.right  * scaleX
                        val bottom = vfTop  + bbImg.bottom * scaleY
                        val bbView = RectF(left, top, right, bottom)
                        if (RectF.intersects(bbView, scanBox)) {
                            Pair(blk, bbView)
                        } else null
                    }
                }

                if (mappedBlocks.isNotEmpty()) {
                    val newText = mappedBlocks.first().first.text
                    val now = System.currentTimeMillis()
                    if (now - lastUpdateTime > MIN_UPDATE_INTERVAL
                        && newText.length >= 2
                        && levenshtein(newText,  b: lastDetectedText ?: "") > 3
                    ) {
                        lastDetectedText = newText
                        lastUpdateTime = now
                        translateText(newText)
                    }
                }
            }
            .addOnFailureListener { e ->
                Log.e( tag: "TD",  msg: e.localizedMessage ?: "error")
            }
            .addOnCompleteListener {
```

Reconocimiento de texto: ML Kit devuelve un objeto visionText con bloques de texto.
Mapeo de coordenadas: Convierte las coordenadas del texto de la imagen a la pantalla.
Área de escaneo: Solo procesa texto dentro de **binding.textGuideBox**.
Filtros:

- Intervalo mínimo entre actualizaciones.
- Texto con al menos 2 caracteres (me detectaba letras random).
- Distancia Levenshtein > 3 (evita cambios mínimos).

translateText

```
private fun translateText(input: String) {
    languageIdentifier.identifyLanguage(input)
        .addOnSuccessListener { lang ->
            val src = if (lang == "und") TranslateLanguage.ENGLISH else lang
            val tgt = langs[binding.targetLangSelector.selectedItemPosition]
            translator?.close()
            translator = Translation.getClient(
                TranslatorOptions.Builder()
                    .setSourceLanguage(src)
                    .setTargetLanguage(tgt)
                    .build()
            )
            translator!!.downloadModelIfNeeded()
                .addOnSuccessListener {
                    translator!!.translate(input)
                        .addOnSuccessListener { translated ->
                            binding.srcLang.text = src.uppercase()
                            binding.srcText.text = input
                            binding.translatedText.text = translated
                        }
                }
        }
        .addOnFailureListener {
            binding.srcText.text = input
            binding.translatedText.text = input
        }
}
```

languageIdentifier: Detecta el idioma del texto. Si es indeterminado, usa inglés.
Translation.getClient: Crea un cliente con idioma origen y destino.
downloadModelIfNeeded: Si es necesario, descarga el modelo de traducción.
addOnSuccesListener: Muestra resultados en la UI.

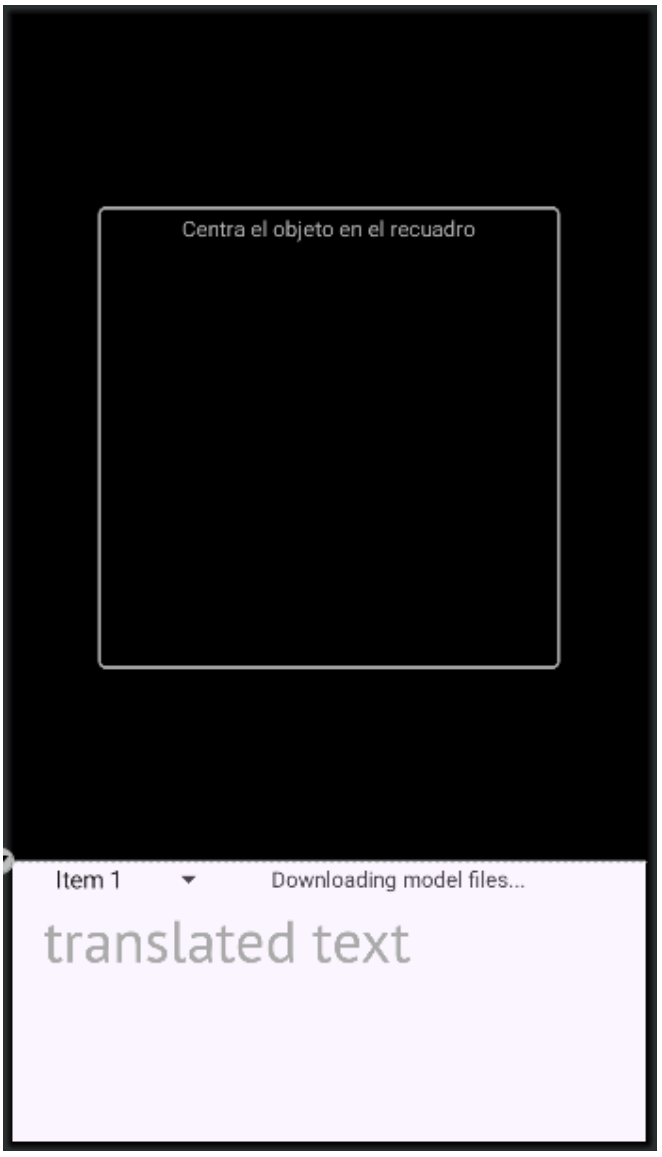
Auxiliares

```
private fun levenshtein(a: String, b: String): Int {
    val dp = Array( size: a.length + 1) { IntArray( size: b.length + 1) }
    for (i in 0 ≤ .. ≤ a.length) dp[i][0] = i
    for (j in 0 ≤ .. ≤ b.length) dp[0][j] = j
    for (i in 1 ≤ .. ≤ a.length) for (j in 1 ≤ .. ≤ b.length) {
        dp[i][j] = minOf(
            a: dp[i - 1][j] + 1,
            b: dp[i][j - 1] + 1,
            c: dp[i - 1][j - 1] + if (a[i - 1] == b[j - 1]) 0 else 1
        )
    }
    return dp[a.length][b.length]
}

override fun onDestroy() {
    super.onDestroy()
    translator?.close()
}
```

levenshtein: Calcula cambios mínimos para convertir un string en otro.
onDestroy: Cierra el cliente de traducción para liberar recursos.

Object Detection

Interfaz



Interfaz parecida a la de text recognition pero eliminando el recuadro de texto reconocido, dejando solo el recuadro de texto traducido y aumentando el tamaño del recuadro para detectar objetos.

Variables y constantes

```
private lateinit var binding: ActivityObjectDetectionBinding
private lateinit var objectDetector: ObjectDetector
private val languageIdentifier by lazy { LanguageIdentification.getClient() }
private var translator: Translator? = null

private val langs = TranslateLanguage.getAllLanguages()
companion object {
    private const val REQUEST_CAMERA_PERMISSION = 1001
    private const val MIN_CONFIDENCE = 0.6f
}
```

Las variables igual son parecidas a las del text recognition ya que es todo igual excepto la parte de detectar objetos en vez de texto.
binding: Enlaza la vista con View Binding (UI).
objectDetector: Cliente de ML Kit para detección de objetos.
languageIdentifier by lazy: Cliente de ML Kit para identificar el idioma del texto al que se ha detectado el objeto (algún objeto se me detectaba en idiomas diferentes).
translator: Cliente para traducción.
langs: Lista de idiomas soportados para traducción.
REQUEST_CAMERA_PERMISSION: Código para solicitud de permiso de cámara.
MIN_CONFIDENCE: Filtra detecciones con menos del 60% de confianza.

Funciones

onCreate

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityObjectDetectionBinding.inflate(layoutInflater)
    setContentView(binding.root)

    val displayNames = langs.map { code ->
        Locale(code).getDisplayName(Locale.getDefault()).ifEmpty { code }
    }
    ArrayAdapter( context: this, android.R.layout.simple_spinner_item, displayNames).also { adapter ->
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
        binding.targetLangSelector.adapter = adapter
    }

    initDetector()
    if (allPermissionsGranted()) startCamera()
    else ActivityCompat.requestPermissions(
        activity: this, arrayOf(Manifest.permission.CAMERA), REQUEST_CAMERA_PERMISSION
    )
}
```

Spinner de Idiomas: Muestra nombres legibles de idiomas.
Inicia Detector: Configura el objeto **objectDetector**.
Permisos: Solicita acceso a la cámara si no está concedido.

initDetector

```
private fun initDetector() {
    val opts = ObjectDetectorOptions.Builder()
        .setDetectorMode(ObjectDetectorOptions.STREAM_MODE)
        .enableMultipleObjects()
        .enableClassification()
        .build()
    objectDetector = ObjectDetection.getClient(opts)
}
```

STREAM_MODE: Optimizado para video en tiempo real.
enableClassification: Permite obtener etiquetas de los objetos detectados.

startCamera

```
private fun startCamera() {
    ProcessCameraProvider.getInstance( context: this).apply {
        addListener({
            val cp = get()
            val preview = Preview.Builder().build().also {
                it.setSurfaceProvider(binding.viewfinder.surfaceProvider)
            }
            val analysis = ImageAnalysis.Builder()
                .setBackpressureStrategy(STRATEGY_KEEP_ONLY_LATEST)
                .build().also {
                    it.setAnalyzer(ContextCompat.getMainExecutor( context: this@ObjectDetectionActivity)) { p ->
                        analyzeImage(p)
                    }
                }
            cp.unbindAll()
            cp.bindToLifecycle(
                lifecycleOwner: this@ObjectDetectionActivity,
                androidx.camera.core.CameraSelector.DEFAULT_BACK_CAMERA,
                preview, analysis
            )
        }, ContextCompat.getMainExecutor( context: this@ObjectDetectionActivity))
    }
}
```

preview: Muestra el feed de la cámara en **binding.viewfinder**.
ImageAnalysis: Procesa cada fotograma con **analyzeImage()**.

analyzeImage

```
@androidx.annotation.OptIn(ExperimentalGetImage::class)
private fun analyzeImage(imageProxy: ImageProxy) {
    imageProxy.image?.let { media ->
        val img = InputImage.fromMediaImage(media, imageProxy.imageInfo.rotationDegrees)
        objectDetector.process(img)
            .addOnSuccessListener { results ->
                val loc = IntArray( size: 2)
                binding.objectGuideBox.getLocationOnScreen(loc)
                val scanBox = Rect(
                    loc[0],
                    loc[1],
                     right: loc[0] + binding.objectGuideBox.width,
                     bottom: loc[1] + binding.objectGuideBox.height
                )

                val filtered = results.filter { obj ->
                    obj.labels.any { it.confidence >= MIN_CONFIDENCE } &&
                            scanBox.contains(
                                obj.boundingBox.centerX(),
                                obj.boundingBox.centerY()
                            )
                }

                if (filtered.isNotEmpty()) {
                    val best = filtered.maxByOrNull {
                        it.labels.maxOf { lbl -> lbl.confidence }
                    }!!

                    val label = best.labels
                        .filter { it.confidence >= MIN_CONFIDENCE }
                        .maxByOrNull { it.confidence }!!
                        .text

                    translateText(best.boundingBox, label)
                } else {
                    binding.overlay.setObjects(emptyList())
                }
            }
            .addOnFailureListener { e ->
                Log.e( tag: "OD",  msg: e.localizedMessage ?: "error procesando imagen")
            }
            .addOnCompleteListener {
                imageProxy.close()
            }
    } ?: imageProxy.close()
}
```

scanBox: Calcula la posición del cuadro guía (**objectGuideBox**) en la pantalla.
filtered: Filtra los objetos para detectar solo los que cumplan las condiciones: **MIN_CONFIDENCE** y que esté dentro de **boundingBox**.
best: Selecciona el objeto con la etiqueta de mayor confianza.
translateText(x,x): Envía la etiqueta para traducción.

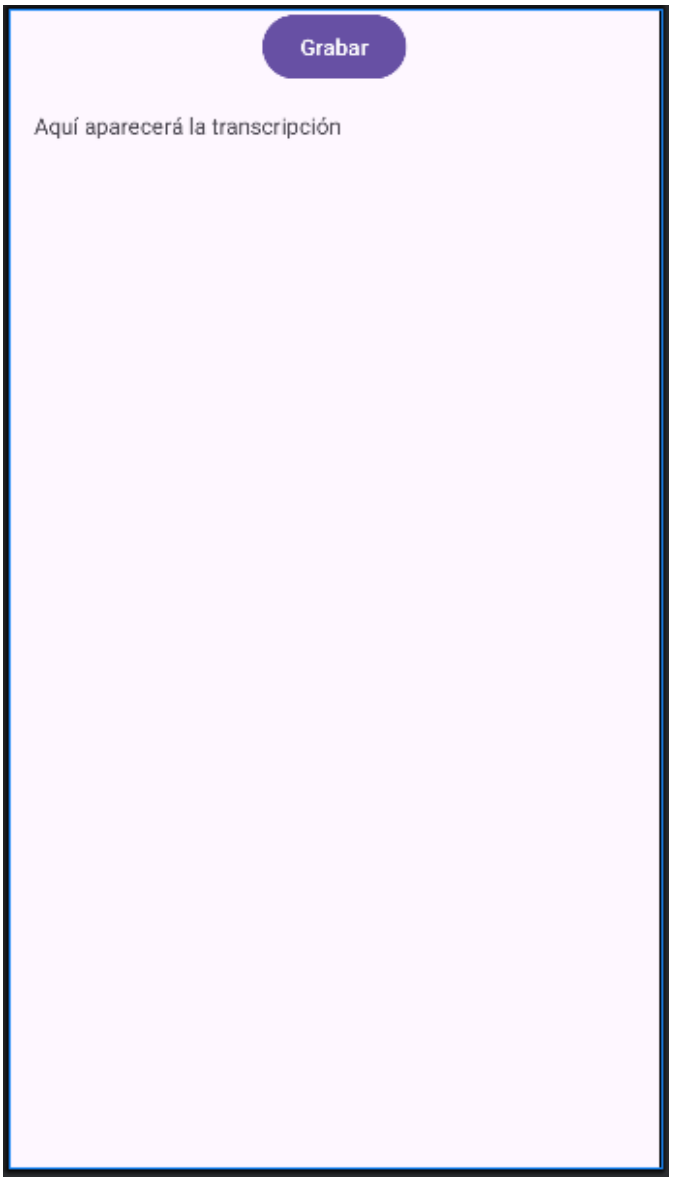
translateText

```
private fun translateText(bbox: Rect, label: String) {
    languageIdentifier.identifyLanguage(label)
        .addOnSuccessListener { lang ->
            val src = if (lang == "und") TranslateLanguage.ENGLISH else lang
            val tgt = langs[binding.targetLangSelector.selectedItemPosition]

            translator?.close()
            translator = Translation.getClient(
                TranslatorOptions.Builder()
                    .setSourceLanguage(src)
                    .setTargetLanguage(tgt)
                    .build()
            )

            translator!!.downloadModelIfNeeded()
                .addOnSuccessListener {
                    translator!!.translate(label)
                        .addOnSuccessListener { translated ->
                            binding.overlay.setObjects(listOf( TranslatedObject(bbox, translated) ))
                        }
                        .addOnFailureListener {
                            binding.overlay.setObjects(listOf( TranslatedObject(bbox, label) ))
                        }
                }
                .addOnFailureListener {
                    binding.overlay.setObjects(listOf( TranslatedObject(bbox, label) ))
                }
        }
        .addOnFailureListener {
            binding.overlay.setObjects(listOf( TranslatedObject(bbox, label) ))
        }
}
```

languageIdentifier: Detecta el idioma del texto. Si es indeterminado, usa inglés.
Translation.getClient: Crea un cliente con idioma origen y destino.
downloadModelIfNeeded: Si es necesario, descarga el modelo de traducción.
addOnSuccesListener: Muestra resultados en la UI.

Whisper

Interfaz



Interfaz sencilla para mostrar la funcionalidad del Whisper con un botón para comenzar y detener la grabación de audio y un TextView en el que aparecerá la transcripción.

Variables y constantes

```
private lateinit var btnRecord: Button
private lateinit var progress: ProgressBar
private lateinit var tvTranscript: TextView
private var recorder: MediaRecorder? = null
private lateinit var audioFile: File
```

btnRecord: Botón para iniciar/detener grabación.
progress: ProgressBar durante la transcripción.
tvTranscript: TextView para mostrar el texto transcrito.
recorder: almacena **MediaRecorder** que graba audio en formato AAC/MP4
audioFile: Archivo temporal para almacenar el audio

Funciones

onCreate

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_whisper)

    btnRecord    = findViewById(R.id.btnRecord)
    progress     = findViewById(R.id.progress)
    tvTranscript = findViewById(R.id.tvTranscript)

    btnRecord.setOnClickListener {
        if (recorder == null) startRecording() else stopRecordingAndTranscribe()
    }

    requestPermissionsIfNeeded()
}
```

setContentView: Configura el layout.
findViewByid: Busca los componentes del layout.
setOnClickListener: Personaliza el funcionamiento del botón llamando a **startRecording** si **recorder** es null al pulsar o **stopRecordingAndTranscribe** si no es null.
requestPermissionsIfNeeded: Verifica que estén los permisos y si no los pide.

requestPermissionsIfNeeded

```
private fun requestPermissionsIfNeeded() {
    val perms = arrayOf(Manifest.permission.RECORD_AUDIO, Manifest.permission.INTERNET)
    val missing = perms.filter {
        ContextCompat.checkSelfPermission( context: this, it) != PackageManager.PERMISSION_GRANTED
    }
    if (missing.isNotEmpty()) {
        ActivityCompat.requestPermissions( activity: this, missing.toTypedArray(), requestCode: 200)
    }
}
```

Revisa si los permisos de **RECORD_AUDIO** e **INTERNET** están activos, sino con **requestPermissions** los pide.

startRecording

```
private fun startRecording() {
    audioFile = File(cacheDir, child: "whisper_input.m4a")
    recorder = MediaRecorder().apply {
        setAudioSource(MediaRecorder.AudioSource.MIC)
        setOutputFormat(MediaRecorder.OutputFormat.MPEG_4)
        setAudioEncoder(MediaRecorder.AudioEncoder.AAC)
        setOutputFile(audioFile.absolutePath)
        prepare()
        start()
    }
    btnRecord.text = "Detener"
}
```

recorder: Configura el grabador en AAC/MP4.
setOutputFile: Guarda el archivo temporal.
btnRecord.text: Cambia el texto del botón a “Detener”.

stopRecordingAndTranscribe

```
private fun stopRecordingAndTranscribe() {
    recorder?.apply {
        stop()
        release()
    }
    recorder = null
    btnRecord.text = "Grabar"
    transcribeWithWhisper(audioFile)
}
```

release: Libera recursos del grabador
recorder: “Reinicia” el grabador”
btnRecord.text: Cambia el texto del botón a “Grabar”.
transcribeWithWhisper: Llama al método encargado de transcribir el audio con Whisper.

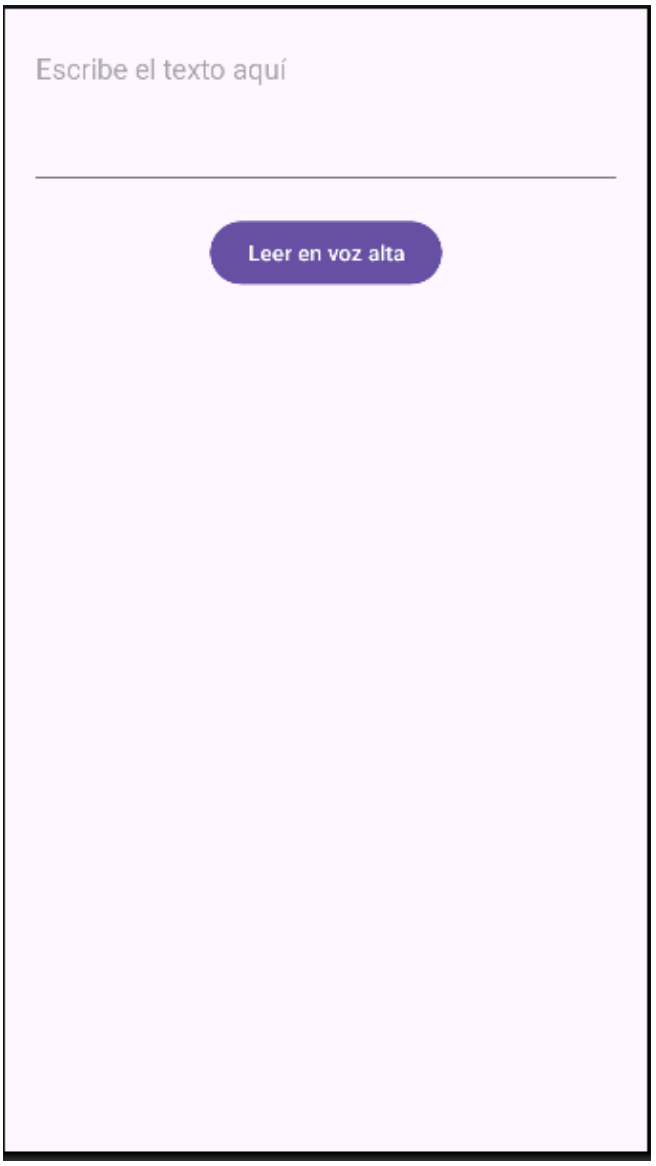
transcribeWithWhisper

```
private fun transcribeWithWhisper(file: File) {
    progress.visibility = View.VISIBLE
    tvTranscript.text = ""

    val client = OkHttpClient()
    val mediaType = "audio/m4a".toMediaType()
    val body = MultipartBody.Builder().setType(MultipartBody.FORM)
        .addFormDataPart( name: "model", value: "whisper-1")
        .addFormDataPart(
            name: "file", file.name,
            file.asRequestBody(mediaType)
        )
        .build()

    val request = Request.Builder()
        .url("https://api.openai.com/v1/audio/transcriptions")
        .addHeader( name: "Authorization", value: "Bearer ${BuildConfig.OPENAI_API_KEY}")
        .post(body)
        .build()

    client.newCall(request).enqueue(object : Callback {
        override fun onFailure(call: Call, e: IOException) {
            runOnUiThread {
                progress.visibility = View.GONE
                tvTranscript.text = "Error: ${e.localizedMessage}"
            }
        }
        override fun onResponse(call: Call, response: Response) {
            val json = response.body?.string()
            Log.d( tag: "WhisperAPI", msg: "Response: $json")
            val text = JSONObject( json: json ?: "{}").optString( name: "text", fallback: "—")
            runOnUiThread {
                progress.visibility = View.GONE
                tvTranscript.text = text
            }
        }
    })
}
```

client: Crea una instancia OkHttpClient, que se encargará de manejar la conexión http con la API.
mediaType: Define el tipo del archivo de audio m4a.
MultipartBuilder: Construye una solicitud para enviar archivos, en ella se especifica el modelo whisper-1 y el nombre del archivo el cual transforma en el tipo correcto (por si acaso)
request: especifica la url específica para la API de OpenAI, pone en el header el **OPENAI_API_KEY** que lo he almacenado en un archivo el cual no se sube a github ya que al hacerlo al tener la key en un repositorio público se cancelaba automáticamente (para que no puedan usar mi key).
onFailure: Muestra un mensaje de error en el TextView se han habido problemas.
onResponse: Obtiene la respuesta en formato JSON y coloca el texto proporcionado por el json en el TextView.

TTS

Interfaz



Es una interfaz sencilla únicamente para mostrar el funcionamiento de la función, con un EditText para escribir lo que quieras y un botón para activar el Whisper y que lea el texto escrito.

Variables, constantes y herencias

```
class TTSActivity : AppCompatActivity(), TextToSpeech.OnInitListener {

    private lateinit var tts: TextToSpeech
    private lateinit var etText: EditText
    private lateinit var btnSpeak: Button
```

AppCompatActivity: Base para actividades.

TextToSpeech.OnInitListener: Interfaz para detectar cuando el motor TTS está inicializado.
tts: Objeto principal del sistema de texto a voz.
etText: Campo de texto para entrada del usuario.
btnSpeak: Botón para activar la conversión.

Funciones

onCreate

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_tts)

    etText = findViewById(R.id.etText)
    btnSpeak = findViewById(R.id.btnSpeak)

    tts = TextToSpeech( context: this,  listener: this)

    btnSpeak.setOnClickListener {
        val text = etText.text.toString()
        if (text.isNotBlank()) {
            speak(text)
        } else {
            Toast.makeText( context: this,  text: "Escribe algún texto", Toast.LENGTH_SHORT).show()
        }
    }
}
```

setContentView: Vincula el layout.
findViewById: Busca los componentes del layout.
TextToSpeech(this, this): Crea la instancia **TTS** usando el contexto y el listener (la propia actividad)
setOnClickListener: Personaliza la funcionalidad del botón al pulsarlo, primero recibe el texto del **etText**, si el texto no está vacío llama a la variable **speak**, sino muestra un mensaje indicando que hay que escribir texto.

onInit

```
override fun onInit(status: Int) {
    if (status == TextToSpeech.SUCCESS) {
        val result = tts.setLanguage(Locale( language: "es",  country: "ES"))
        if (result == TextToSpeech.LANG_MISSING_DATA || result == TextToSpeech.LANG_NOT_SUPPORTED) {
            Toast.makeText( context: this,  text: "Idioma no soportado", Toast.LENGTH_SHORT).show()
        }
    } else {
        Toast.makeText( context: this,  text: "Inicialización de TTS fallida", Toast.LENGTH_SHORT).show()
    }
}
```

Sirve para configurar el **TTS** e indicar si falla la inicialización del mismo.
Primer intenta poner como base el idioma español al **TTS** y revisa si falta el paquete

de idioma (**LANG_MISSING_DATA**) o el idioma no está disponible (**LANG_NOT_SUPPORTED**)

speak

```
private fun speak(text: String) {
    tts.speak(text, TextToSpeech.QUEUE_FLUSH,  params: null,  utteranceId: null
}
```

recibe el texto en formato string.
QUEUE_FLUSH: Detiene cualquier reproducción anterior y comienza la nueva inmediatamente
nulls: Parámetros que no han sido necesarios.

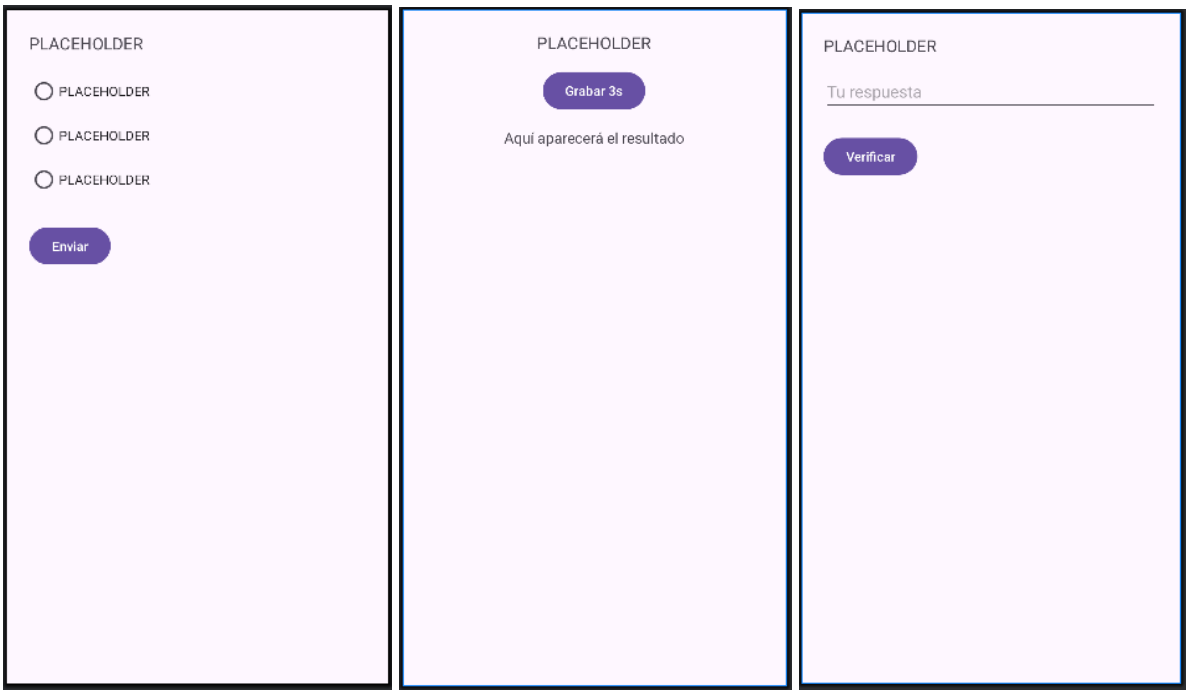
onDestroy

```
override fun onDestroy() {
    if (::tts.isInitialized) {
        tts.stop()
        tts.shutdown()
    }
    super.onDestroy()
}
```

stop: Detiene la reproducción actual.
shutdown: Libera los recursos del **TTS**.

Ejercicios Interactivos

Interfaz



Aquí están las interfaces de los 3 ejercicios diferentes que he implementado, siendo uno de múltiple elección, otro de pronunciación y otro de escritura.
Como se ve aquí pone PLACEHOLDER en bastantes textos, eso es porque luego las propias clases se encargan de proporcionar el texto de las preguntas dependiendo del idioma que se haya elegido en el MainMenu.

Variables y constantes

```
private lateinit var translator: Translator
private lateinit var correctAnswer: String
private lateinit var spanishWord: String

private val allCandidates = listOf(
    "perro", "gato", "pájaro", "caballo", "ratón", "vaca"
)
```

translator: Objeto para traducción de texto usando ML Kit.
correctAnswer: Almacena la respuesta correcta traducida.
spanishWord: Palabra en español a adivinar.
allCandidates: Lista fija de palabras en español para posibles respuestas y opciones.

Funciones

onCreate

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_mcq_level)

    val targetLang = intent.getStringExtra( name: "targetLang")
        ?: TranslateLanguage.ENGLISH

    val opts = TranslatorOptions.Builder()
        .setSourceLanguage(TranslateLanguage.SPANISH)
        .setTargetLanguage(targetLang)
        .build()
    translator = Translation.getClient(opts)

    translator.downloadModelIfNeeded()
        .addOnSuccessListener { prepareExercise(targetLang) }
        .addOnFailureListener {
            Toast.makeText( context: this,  text: "Error cargando modelo: ${it.message}", Toast.LENGTH_LONG).show()
        }
}
```

Recibe el idioma objetivo del **Intent**.
Crea un traductor con ML Kit.
Descarga el modelo de traducción necesario.
Si tiene éxito, prepara el ejercicio con **prepareExercise**.

prepareExercise

```
private fun prepareExercise(targetLang: String) {
    val three = allCandidates.shuffled().take( n: 3)

    spanishWord = three.random()

    translateList(three) { translations ->
        val idx = three.indexOf(spanishWord)
        correctAnswer = translations[idx]

        setupUI(targetLang, translations.shuffled())
    }
}
```

Selecciona 3 palabras aleatorias.
Elige una palabra objetivo (**spanishWord**).
Traduce todas las palabras.
Identifica la traducción correcta y la guarda en **correctAnswer**.
Configura la UI con opciones en orden aleatorio (**shuffled**).

translateList

```
private fun translateList(words: List<String>, onDone: (List<String>) -> Unit) {
    val results = mutableListOf<String>()
    fun recurse(i: Int) {
        if (i == words.size) {
            onDone(results)
            return
        }
        translator.translate(words[i])
            .addOnSuccessListener {
                results.add(it.trim())
                recurse( i: i + 1)
            }
            .addOnFailureListener {
                results.add("…")
                recurse( i: i + 1)
            }
    }
    recurse( i: 0)
}
```

Usa recursión para manejar traducciones asíncronas secuenciales.
Añade resultados (o “...” en errores) a una lista.
Llama al callback onDone cuando completa todas.

setupUI

```
private fun setupUI(targetLang: String, options: List<String>) {
    val displayLang = Locale(targetLang)
        .getDisplayLanguage(Locale.getDefault())
        .replaceFirstChar { it.uppercase() }

    findViewById<TextView>(R.id.tvQuestion).text =
        "¿Cómo se dice '$spanishWord' en $displayLang?"

    val rg = findViewById<RadioGroup>(R.id.radioGroup)
    options.forEachIndexed { idx, opt ->
        (rg.getChildAt(idx) as RadioButton).text = opt
    }

    findViewById<Button>(R.id.btnSubmit).setOnClickListener {
        val selId = rg.checkedRadioButtonId
        if (selId != -1) {
            val choice = findViewById<RadioButton>(selId).text.toString()
            val correct = choice.equals(correctAnswer, ignoreCase = true)
            Toast.makeText(
                context: this,
                if (correct) "✅ Correcto" else "❌ Incorrecto",
                Toast.LENGTH_SHORT
            ).show()
            saveProgress( exerciseId: "mcq1", correct)
            if (correct) {
                Handler(Looper.getMainLooper()).postDelayed({
                    Intent( packageContext: this, MainMenuActivity::class.java)
                        .addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP)
                        .putExtra(MainMenuActivity.EXTRA_AUTO_NEXT,  value: true)
                        .putExtra( name: "targetLang", targetLang)
                        .also { startActivity(it) }
                    finish()
                }, delayMillis: 2000)
            }
        } else {
            Toast.makeText( context: this,  text: "Selecciona una opción", Toast.LENGTH_SHORT).show()
        }
    }
}
```

Muestra la pregunta con el idioma seleccionado en el MainMenu.
Llena 3 RadioButton con las traducciones desordenadas.
Valida la respuesta seleccionada.
Si es correcta:
Guarda progreso con la función **saveProgress**.
Navega al menú principal después de 2 segundos.

saveProgress

```
private fun saveProgress(exerciseId: String, completed: Boolean) {
    val userId = Settings.Secure.getString(contentResolver, Settings.Secure.ANDROID_ID)
    FirebaseDatabase.getInstance()
        .getReference( path: "users")
        .child(userId)
        .child( pathString: "exercises")
        .child(exerciseId)
        .setValue(mapOf("completed" to completed))
}
```

Usa el **ANDROID_ID** como identificador único para el usuario.
Guarda los datos en firebase indicando que el usuario **userId** ha completado (**completed**) el ejercicio **exerciseId** que se pasa como parámetro de la función.

Diferencias entre tipos

Este es un nivel para el tipo MultipleChoiceQuestion, el resto son bastante parecidos, ahora voy a indicar los cambios más significativos entre los otros dos tipos (Write, Pronounce) y el que he mostrado.

Write no hay nada muy importante que recalcar exceptuando esta función:

```
private fun normalize(s:String) = Normalizer.normalize(s,Normalizer.Form.NFD)
    .replace("\\p{InCombiningDiacriticalMarks}+".toRegex(), replacement: "")
    .lowercase(Locale.getDefault())
```

El cual lo único que hace es eliminar signos de acentuación (por si acaso el traductor lo pone sin) y cambiar todo a lowerCase, ya que al ser una aplicación móvil la primera letra se suele escribir en mayúscula y no lo pillaba como correcto.

Pronounce lo más recalcable es esta función que se encarga de confirmar si la pronunciación ha sido correcta.

```
private fun evaluatePronunciation(){
    val prog=findViewById<ProgressBar>(R.id.prog)
    val client=OkHttpClient()
    val body=MultipartBody.Builder().setType(MultipartBody.FORM)
        .addFormDataPart( name: "model", value: "whisper-1")
        .addFormDataPart( name: "file",audioFile.name,
            audioFile.asRequestBody("audio/m4a".toMediaType()))
        .build()
    val req=Request.Builder()
        .url("https://api.openai.com/v1/audio/transcriptions")
        .addHeader( name: "Authorization", value: "Bearer ${BuildConfig.OPENAI_API_KEY}")
        .post(body).build()

    client.newCall(req).enqueue(object:Callback{
        override fun onFailure(c:Call,e:IOException)=runOnUiThread{
            prog.visibility=ProgressBar.GONE
            findViewById<TextView>(R.id.tvResult).text="Error"
        }
        override fun onResponse(c:Call,r:Response){
            val res=JSONObject(r.body!!.string()).optString( name: "text", fallback: "").trim()
            val normalizedRes = res.lowercase().replace(Regex( pattern: "[^\\p{L}\\p{N}\\s]"), replacement: "")
            val normalizedCorrect = correctPhrase.lowercase().replace(Regex( pattern: "[^\\p{L}\\p{N}\\s]"), replacement: "")
            val correct = normalizedRes == normalizedCorrect
            runOnUiThread{
                prog.visibility=ProgressBar.GONE
                findViewById<TextView>(R.id.tvResult).text=
                    if(correct)"✅ ¡Pronunciación correcta!"
                    else "❌ Dijiste: \"$res\""
                saveProgress( id: "pronounce1",correct)
                if(correct){
                    Handler(Looper.getMainLooper()).postDelayed({
                        val intent = Intent( packageContext: this@PronounceLevel1Activity, MainMenuActivity::class.java).apply {
                            addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP)
                            putExtra(MainMenuActivity.EXTRA_AUTO_NEXT, value: true)
                            putExtra( name: "targetLang", intent.getStringExtra( name: "targetLang"))
                        }
                        startActivity(intent)
                        finish()
                    }, delayMillis: 2000)
                }
            }
        }
    })
}
```

Es prácticamente igual que en la actividad Whisper pero ahora al transcribirlo elimina todo lo innecesario de la palabra (comas, puntos, exclamaciones…) e igual que el Write lo transforma a lowerCase.

Webgrafía

Kotlin - No conocía tanto el lenguaje kotlin por lo que esta página me ha ayudado en algunas cosas
Firebase - La propia página oficial tiene una guia perfecta para configurar e implementar firebase
ML Kit - Me ha servido mucho para implementar la traducción y un poco para el text recognizer y el object detector
chatGPT - Me ha servido para las guías que iba viendo por internet prácticamente todas eran muy antiguas poder adaptarlas a la versión que yo utilizaba
Forocoches, StackOverflow - Al buscar problemas concretos estas páginas han sido las que me han salido (habían más pero esas solo me habian salido una vez como mucho y no las encuentro)
TTS - Entre otras cosas este video me ha servido de base para implementar el TTS
Whisper - Guia sencilla para implementar Whisper en Android Kotlin.