Documentación Técnica

ML Translate App - Arquitectura y Implementación

Desarrollado por: - Daniel Vega Miranda - Abraham Reyes Cuevas

Fecha: Junio 2025
Versión: 2.0 - Edición Universitaria

Arquitectura General

Stack Tecnológico:

- **Lenguaje:** Kotlin
- **Framework:** Android SDK (API 21+)
- **ML Framework:** Google ML Kit
- **Cámara:** CameraX
- **UI:** Material Design Components
- **Persistencia:** SharedPreferences + Gson
- **Arquitectura:** MVVM Pattern con Fragment-based UI

Estructura del Proyecto:

```
app/src/main/java/com/example/translateApp/
- model/
  - AnalysisHistoryItem.kt
- utils/
  - AnalysisHistory.kt
  - CameraUtils.kt
- view/
  - TranslateActivity.kt (Actividad Principal)
  - HistoryActivity.kt
  - MLKitFeaturesDialog.kt
  - ImageLabelingDialog.kt
  - ImageLabelingDialogRealtime.kt
  - BarcodeScannerDialogRealtime.kt
  - FaceDetectionDialogRealtime.kt
  - LandmarkDetectionDialogRealtime.kt
  - HistoryDetailDialog.kt
- res/
  - layout/ (Layouts XML)
  - values/ (Strings, Colors, Themes)
```

Componentes Principales

1. TranslateActivity.kt

Responsabilidad: Actividad principal que maneja traducción básica y coordinación general.

Funcionalidades Clave: - Configuración de UI principal con Material Design - Gestión de permisos de cámara y almacenamiento - Coordinación entre diferentes funcionalidades ML Kit - Manejo de estado de la aplicación

Dependencias ML Kit:

```
dependencies {
    implementation 'com.google.mlkit:translate:17.0.1'
    implementation 'com.google.mlkit:text-recognition:16.0.0'
    implementation 'com.google.mlkit:barcode-scanning:17.2.0'
    implementation 'com.google.mlkit:image-labeling:17.0.7'
    implementation 'com.google.mlkit:face-detection:16.1.5'
    implementation 'com.google.mlkit:landmark-detection:18.0.1'
}
```

2. Funcionalidades ML Kit Implementadas

A) Reconocimiento de Texto (OCR) **Archivo:** `ImageLabelingDialog.kt` **ML Kit API:** Text Recognition API

Implementación Técnica:

```
private fun recognizeText(image: InputImage) {
    val recognizer = TextRecognition.getClient(TextRecognizerOptions.DEFAULT_OPTIONS)
    recognizer.process(image)
        .addOnSuccessListener { visionText ->
            processTextResult(visionText)
        }
        .addOnFailureListener { e ->
            handleError(e)
        }
}
```

Características: - Soporte offline completo - Múltiples idiomas simultáneos - Extracción de texto estructurado - Detección de líneas y palabras individuales

B) Traducción de Texto **ML Kit API:** Translation API **Implementación:** Integrada en `TranslateActivity.kt`

Código de Implementación:

```
private fun setupTranslator() {
    val options = TranslatorOptions.Builder()
        .setSourceLanguage(sourceLanguageCode)
        .setTargetLanguage(targetLanguageCode)
        .build()

    translator = Translation.getClient(options)

    translator.downloadModelIfNeeded()
        .addOnSuccessListener {
            // Modelo descargado, listo para traducir
```

```
        }
        .addOnFailureListener { e ->
            // Manejo de errores
        }
}
```

Gestión de Modelos: - Descarga automática de paquetes de idiomas - Cache local para uso offline - Más de 50 idiomas soportados - Detección automática de idioma source

C) Escáner de Códigos de Barras (Tiempo Real) **Archivo:** `BarcodeScannerDialogRealtime.kt`
ML Kit API: Barcode Scanning API

Implementación en Tiempo Real:

```
private fun startBarcodeScanning() {
    val options = BarcodeScannerOptions.Builder()
        .setBarcodeFormats(
            Barcode.FORMAT_QR_CODE,
            Barcode.FORMAT_UPC_A,
            Barcode.FORMAT_EAN_13
        )
        .build()

    val scanner = BarcodeScanning.getClient(options)

    // Configuración de CameraX para análisis en tiempo real
    imageAnalysis = ImageAnalysis.Builder()
        .setTargetResolution(Size(640, 480))
        .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
        .build()

    imageAnalysis.setAnalyzer(cameraExecutor) { imageProxy ->
        val inputImage = InputImage.fromMediaImage(
            imageProxy.image!!,
            imageProxy.imageInfo.rotationDegrees
        )

        scanner.process(inputImage)
            .addOnSuccessListener { barcodes ->
                processBarcodes(barcodes)
            }
            .addOnCompleteListener {
                imageProxy.close()
            }
    }
}
```

Formatos Soportados: - QR Code, Data Matrix - UPC-A, UPC-E - EAN-8, EAN-13 - Code 39, Code 128 - PDF417

D) Etiquetado de Imágenes (Tiempo Real) **Archivo:** `ImageLabelingDialogRealtime.kt`
ML Kit API: Image Labeling API

Configuración del Detector:

```
private fun setupImageLabeling() {
    val options = ImageLabelerOptions.Builder()
        .setConfidenceThreshold(0.7f)
        .build()

    imageLabeler = ImageLabeling.getClient(options)
}

private fun analyzeImage(imageProxy: ImageProxy) {
    val inputImage = InputImage.fromMediaImage(
        imageProxy.image!!,
        imageProxy.imageInfo.rotationDegrees
    )

    imageLabeler.process(inputImage)
        .addOnSuccessListener { labels ->
            updateLabelsUI(labels)
        }
        .addOnCompleteListener {
            imageProxy.close()
        }
}
```

Capacidades: - Más de 400 etiquetas diferentes - Niveles de confianza configurables - Análisis en tiempo real (30 FPS) - Categorización automática de objetos

E) Detección de Rostros (Tiempo Real) **Archivo:** `FaceDetectionDialogRealtime.kt`
ML Kit API: Face Detection API

Configuración Avanzada:

```
private fun setupFaceDetection() {
    val options = FaceDetectorOptions.Builder()
        .setPerformanceMode(FaceDetectorOptions.PERFORMANCE_MODE_FAST)
        .setLandmarkMode(FaceDetectorOptions.LANDMARK_MODE_ALL)
        .setClassificationMode(FaceDetectorOptions.CLASSIFICATION_MODE_ALL)
        .setMinFaceSize(0.15f)
        .enableTracking()
        .build()

    faceDetector = FaceDetection.getClient(options)
}

private fun detectFaces(imageProxy: ImageProxy) {
    val inputImage = InputImage.fromMediaImage(
```

```
            imageProxy.image!!,
            imageProxy.imageInfo.rotationDegrees
        )

        faceDetector.process(inputImage)
            .addOnSuccessListener { faces ->
                processFaceResults(faces)
            }
            .addOnCompleteListener {
                imageProxy.close()
            }
    }
```

Información Extraída: - Coordenadas del rostro y landmarks - Probabilidad de sonrisa - Estado de los ojos (abiertos/cerrados) - Ángulos de rotación (X, Y, Z) - Tracking ID para seguimiento

F) Detección de Monumentos (Tiempo Real) **Archivo:** `LandmarkDetectionDialogRealtime.kt`
ML Kit API: Landmark Detection API (Cloud-based)

Implementación Cloud:

```
private fun setupLandmarkDetection() {
    val options = FirebaseVisionCloudDetectorOptions.Builder()
        .setModelType(FirebaseVisionCloudDetectorOptions.LATEST_MODEL)
        .setMaxResults(10)
        .build()

    landmarkDetector = FirebaseVision.getInstance()
        .getCloudLandmarkDetector(options)
}
```

Nota: Esta funcionalidad requiere conexión a internet y integración con Firebase.

Arquitectura de Cámara

Implementación CameraX

Archivo: `CameraUtils.kt`

```
class CameraUtils {
    companion object {
        fun setupCamera(
            context: Context,
            lifecycleOwner: LifecycleOwner,
            previewView: PreviewView,
            imageAnalysis: ImageAnalysis
        ) {
            val cameraProviderFuture = ProcessCameraProvider.getInstance(context)

            cameraProviderFuture.addListener({
```

```
                val cameraProvider = cameraProviderFuture.get()

                val preview = Preview.Builder().build()
                preview.setSurfaceProvider(previewView.surfaceProvider)

                val cameraSelector = CameraSelector.DEFAULT_BACK_CAMERA

                try {
                    cameraProvider.unbindAll()
                    cameraProvider.bindToLifecycle(
                        lifecycleOwner,
                        cameraSelector,
                        preview,
                        imageAnalysis
                    )
                } catch (e: Exception) {
                    Log.e("CameraUtils", "Error binding camera", e)
                }
            }, ContextCompat.getMainExecutor(context))
        }
    }
}
```

Optimizaciones de Rendimiento

1. Gestión de Threads:

```
private val cameraExecutor = Executors.newSingleThreadExecutor()
private val analysisExecutor = Executors.newSingleThreadExecutor()
```

2. Estrategia de Backpressure:

```
.setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
```

3. Resolución Optimizada:

```
.setTargetResolution(Size(640, 480)) // Balance entre calidad y rendimiento
```

Gestión de Datos

Modelo de Historial

Archivo: `AnalysisHistoryItem.kt`

```
data class AnalysisHistoryItem(
    val id: String = UUID.randomUUID().toString(),
    val timestamp: Long = System.currentTimeMillis(),
    val featureType: MLKitFeatureType,
    val originalText: String,
    val translatedText: String? = null,
    val sourceLanguage: String? = null,
    val targetLanguage: String? = null,
```

```
    val confidence: Float? = null,
    val metadata: Map<String, Any> = emptyMap()
)

enum class MLKitFeatureType {
    TEXT_RECOGNITION,
    TRANSLATION,
    BARCODE_SCANNING,
    IMAGE_LABELING,
    FACE_DETECTION,
    LANDMARK_DETECTION
}
```

Persistencia de Datos

Archivo: `AnalysisHistory.kt`

```
class AnalysisHistory(private val context: Context) {
    private val sharedPreferences = context.getSharedPreferences(
        "ml_translate_history",
        Context.MODE_PRIVATE
    )
    private val gson = Gson()

    fun saveAnalysis(item: AnalysisHistoryItem) {
        val currentHistory = getHistory().toMutableList()
        currentHistory.add(0, item) // Agregar al inicio

        // Mantener solo los últimos 100 elementos
        if (currentHistory.size > 100) {
            currentHistory.subList(100, currentHistory.size).clear()
        }

        val jsonString = gson.toJson(currentHistory)
        sharedPreferences.edit()
            .putString("history_items", jsonString)
            .apply()
    }

    fun getHistory(): List<AnalysisHistoryItem> {
        val jsonString = sharedPreferences.getString("history_items", "[]")
        val type = object : TypeToken<List<AnalysisHistoryItem>>() {}.type
        return gson.fromJson(jsonString, type) ?: emptyList()
    }
}
```

Manejo de Permisos

Implementación Robusta

```
class PermissionManager(private val activity: Activity) {
    companion object {
        const val CAMERA_PERMISSION_REQUEST_CODE = 1001
        const val STORAGE_PERMISSION_REQUEST_CODE = 1002
    }

    fun checkAndRequestCameraPermission(): Boolean {
        return if (ContextCompat.checkSelfPermission(
                activity,
                Manifest.permission.CAMERA
            ) != PackageManager.PERMISSION_GRANTED
        ) {
            ActivityCompat.requestPermissions(
                activity,
                arrayOf(Manifest.permission.CAMERA),
                CAMERA_PERMISSION_REQUEST_CODE
            )
            false
        } else {
            true
        }
    }
}
```

Configuración de Build

build.gradle (Module: app)

```
android {
    compileSdk 34

    defaultConfig {
        applicationId "com.example.translateApp"
        minSdk 21
        targetSdk 34
        versionCode 2
        versionName "2.0"
    }

    buildFeatures {
        viewBinding true
    }

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
```

```
    }

    kotlinOptions {
        jvmTarget = '1.8'
    }
}

dependencies {
    // ML Kit Dependencies
    implementation 'com.google.mlkit:translate:17.0.1'
    implementation 'com.google.mlkit:text-recognition:16.0.0'
    implementation 'com.google.mlkit:barcode-scanning:17.2.0'
    implementation 'com.google.mlkit:image-labeling:17.0.7'
    implementation 'com.google.mlkit:face-detection:16.1.5'
    implementation 'com.google.mlkit:landmark-detection:18.0.1'

    // Camera
    implementation 'androidx.camera:camera-camera2:1.3.0'
    implementation 'androidx.camera:camera-lifecycle:1.3.0'
    implementation 'androidx.camera:camera-view:1.3.0'

    // UI y Material Design
    implementation 'com.google.android.material:material:1.9.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'

    // Utilities
    implementation 'com.google.code.gson:gson:2.10.1'
}
```

AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <uses-feature
        android:name="android.hardware.camera"
        android:required="true" />
    <uses-feature
        android:name="android.hardware.camera.autofocus"
        android:required="false" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
```

```
        android:label="@string/app_name"
        android:theme="@style/Theme.TranslateApp">

        <activity
            android:name=".view.TranslateActivity"
            android:exported="true"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name=".view.HistoryActivity"
            android:parentActivityName=".view.TranslateActivity" />

        <meta-data
            android:name="com.google.mlkit.vision.DEPENDENCIES"
            android:value="barcode_scanning,text_recognition,image_labeling,face_dete

    </application>
</manifest>
```

Optimizaciones y Mejores Prácticas

1. Gestión de Memoria

```
override fun onDestroy() {
    super.onDestroy()
    cameraExecutor.shutdown()
    analysisExecutor.shutdown()

    // Liberar recursos ML Kit
    translator?.close()
    textRecognizer?.close()
    barcodeScanner?.close()
    imageLabeler?.close()
    faceDetector?.close()
}
```

2. Manejo de Estados de Ciclo de Vida

```
override fun onResume() {
    super.onResume()
    if (allPermissionsGranted()) {
        startCamera()
    }
}
```

```
override fun onPause() {
    super.onPause()
    stopAnalysis()
}
```

3. Throttling para Análisis en Tiempo Real

```
private var lastAnalysisTimestamp = 0L
private val analysisIntervalMs = 100L // 10 FPS máximo

private fun shouldAnalyze(): Boolean {
    val currentTime = System.currentTimeMillis()
    return currentTime - lastAnalysisTimestamp >= analysisIntervalMs
}
```

4. Cache de Resultados

```
private val resultCache = LruCache<String, AnalysisResult>(50)

private fun getCachedResult(inputHash: String): AnalysisResult? {
    return resultCache.get(inputHash)
}
```

Pruebas y Debugging

Logging Personalizado

```
object MLKitLogger {
    private const val TAG = "MLTranslateApp"

    fun logFeatureUsage(featureType: MLKitFeatureType, duration: Long) {
        Log.d(TAG, "Feature: $featureType, Duration: ${duration}ms")
    }

    fun logError(feature: String, error: Exception) {
        Log.e(TAG, "Error in $feature: ${error.message}", error)
    }
}
```

Métricas de Performance

```
class PerformanceMetrics {
    fun measureAnalysisTime(operation: () -> Unit): Long {
        val startTime = System.currentTimeMillis()
        operation()
        return System.currentTimeMillis() - startTime
    }
}
```

Seguridad y Privacidad

1. Manejo Seguro de Datos

- Todos los análisis se procesan localmente cuando es posible
- Los datos sensibles no se almacenan en logs
- Cache automático de limpieza después de 24 horas

2. Conformidad con GDPR

- Opción para eliminar todo el historial
- Consentimiento explícito para funcionalidades online
- Transparencia en el uso de datos

Deployment y Distribución

ProGuard Configuration

```
# ML Kit
-keep class com.google.mlkit.** { *; }
-keep class com.google.android.gms.** { *; }
-dontwarn com.google.mlkit.**

# Gson
-keepattributes Signature
-keep class com.example.translateApp.model.** { *; }
```

APK Optimization

- Vector drawables para iconos
- WebP para imágenes grandes
- Splits de APK por arquitectura (opcional)

Conclusiones Técnicas

Esta implementación de ML Translate App demuestra:

1. **Integración Completa de ML Kit:** Uso de 6+ APIs diferentes de ML Kit
2. **Arquitectura Escalable:** Patrón MVVM con separación clara de responsabilidades
3. **Performance Optimizado:** Análisis en tiempo real con gestión eficiente de recursos
4. **UX Fluida:** Material Design con navegación intuitiva
5. **Robustez:** Manejo comprehensivo de errores y estados edge

Estadísticas del Proyecto:

- **Líneas de Código:** ~2,500 líneas Kotlin
- **Archivos de Layout:** 15+ layouts XML
- **APIs ML Kit Utilizadas:** 6 diferentes
- **Funcionalidades en Tiempo Real:** 4 implementadas
- **Soporte de Idiomas:** 50+ para traducción

Tecnologías Clave:

- Google ML Kit (Core)
- CameraX (Análisis en tiempo real)
- Material Design Components
- Arquitectura MVVM
- Gson (Persistencia)
- SharedPreferences (Settings)

Desarrollado por: - **Daniel Vega Miranda** - Arquitectura y Backend - **Abraham Reyes Cuevas** - UI/UX e Integración ML Kit

Versión Técnica: 2.0 - Edición Universitaria
Fecha de Documentación: Junio 2025

Esta documentación técnica proporciona una vista completa de la implementación de ML Translate App, desde la arquitectura hasta los detalles de implementación específicos de cada funcionalidad ML Kit.