



INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE COMPUTO



INGENIERÍA EN SISTEMAS COMPUTACIONALES

PRACTICA 3: APLICACIONES NATIVAS

HECTOR ALEJANDRO HERNANDEZ ARANDA
2022630481

7CV2

GABRIEL HURTADO AVILÉS

15/10/2025

INTRODUCCIÓN

El presente documento detalla el proceso de diseño, desarrollo y pruebas de una aplicación móvil nativa para el sistema operativo Android, como parte de la tercera práctica evaluativa de la asignatura de Desarrollo de Aplicaciones Móviles Nativas.

De los tres ejercicios propuestos, se ha seleccionado el Ejercicio 3: Aplicación Nativa de Libre Elección, desarrollando una utilidad denominada "Kit de Herramientas Sencillo". La justificación de esta elección radica en el interés de centrar el esfuerzo en la implementación directa y robusta de múltiples recursos nativos del sistema. Se priorizó el dominio de la interacción con APIs de hardware, como CameraManager y Vibrator, y la correcta gestión del ciclo de vida de los permisos en tiempo de ejecución, aspectos fundamentales del desarrollo nativo, en lugar de abordar la complejidad de la interfaz de usuario que requerían los ejercicios 1 y 2.

MARCO TEÓRICO

Aplicaciones Nativas vs. Híbridas vs. Web Apps

El desarrollo de aplicaciones móviles se puede clasificar en tres categorías principales:

- **Nativas:** Desarrolladas específicamente para un sistema operativo (como Android o iOS) utilizando sus lenguajes y SDKs oficiales (Kotlin/Java para Android). Ofrecen el máximo rendimiento, acceso completo a las APIs del dispositivo y la mejor experiencia de usuario al seguir las guías de diseño de la plataforma.
- **Híbridas:** Se construyen utilizando tecnologías web (HTML, CSS, JavaScript) y se ejecutan dentro de un contenedor nativo (WebView). Frameworks como Flutter o React Native permiten compartir una base de código entre plataformas, pero introducen una capa de abstracción que puede impactar el rendimiento y el acceso a ciertas funcionalidades nativas.
- **Web Apps (PWA):** Son sitios web diseñados para funcionar como aplicaciones. Se acceden a través de un navegador pero pueden ser "instaladas" en la pantalla de inicio y ofrecer funcionalidades offline. Su acceso al hardware del dispositivo es el más limitado de las tres.

Arquitectura Android y Componentes Principales

El sistema operativo Android está estructurado sobre un kernel de Linux y provee un framework de aplicaciones para los desarrolladores. Los componentes esenciales de una aplicación son:

- **Activities:** Representan una única pantalla con una interfaz de usuario. Son el punto de entrada para la interacción del usuario.

- Services: Componentes que se ejecutan en segundo plano para realizar operaciones de larga duración sin una interfaz de usuario, como reproducir música.
- Broadcast Receivers: Permiten a la aplicación responder a mensajes de difusión del sistema u otras aplicaciones (ej. batería baja, red conectada).
- Content Providers: Gestionan un conjunto compartido de datos de la aplicación, permitiendo que otras apps (con los permisos adecuados) los consulten o modifiquen.

Patrones de Diseño Utilizados

Dada la escala del proyecto, no se implementó un patrón arquitectónico complejo como MVVM o Clean Architecture. Sin embargo, se aplicó de forma implícita el Patrón Observer (Observador) a través del uso de `setOnClickListener`. En este esquema, el Button actúa como el "sujeto" que emite un evento (un clic), y el código dentro de la función lambda actúa como el "observador" que se suscribe a dicho evento y reacciona ejecutando una acción específica.

Descripción General y Flujo de Usuario

La aplicación "Kit de Herramientas Sencillo" es una utilidad nativa que agrupa tres funcionalidades básicas que requieren acceso directo a los recursos del dispositivo.

Flujo de Usuario:

- El usuario inicia la aplicación y se presenta la pantalla principal, que contiene las secciones de Linterna, Vibración y Notificación.
- Al presionar el botón "Encender" en la sección de Linterna, la aplicación solicita el permiso de CAMERA si es la primera vez. Una vez concedido, el flash del dispositivo se activa y el texto del botón cambia a "Apagar".
- Al pulsar el botón "Vibrar Corto", el dispositivo ejecuta una vibración háptica de 200 milisegundos.
- El usuario puede introducir un texto en el campo de edición y presionar "Notificar en 5 seg". La aplicación solicita el permiso POST_NOTIFICATIONS (si es necesario) y, tras una pausa de cinco segundos, muestra una notificación del sistema con el texto introducido.

Arquitectura Técnica

La arquitectura de la aplicación es deliberadamente simple, centralizando toda la lógica en un único componente para enfocarse en la interacción con las APIs del sistema.

Diagrama de Componentes:

- Estructura: El proyecto consta de una única Activity que controla la lógica y una vista definida por un archivo de layout XML.
- Clases Principales:
 - MainActivity.kt: Actúa como controlador y vista. Es responsable de inicializar la UI, configurar los listeners de los botones y llamar a las APIs del sistema (CameraManager, Vibrator, NotificationManager) para ejecutar las funcionalidades.

Acceso a Recursos Nativos

A continuación, se muestran fragmentos de código clave que ilustran la implementación de cada funcionalidad nativa.

- Linterna (API CameraManager):Kotlin// Se obtiene una instancia del CameraManager para interactuar con el hardware de la cámara.

```
// Se obtiene una instancia del CameraManager para interactuar con el
hardware de la cámara.
private val cameraManager = getSystemService(Context.CAMERA_SERVICE) as
CameraManager

private fun toggleFlashlight(turnOn: Boolean) {
    try {
        // Se obtiene el ID de la primera cámara (usualmente la trasera).
        val cameraId = cameraManager.cameraIdList[0]
        // Se activa o desactiva el modo antorcha (flash).
        cameraManager.setTorchMode(cameraId, turnOn)
    } catch (e: Exception) {
        // Manejo de errores si el acceso a la cámara falla.
        e.printStackTrace()
    }
}
```

- Vibrador (API Vibrator/VibratorManager):

```
private fun vibratePhone() {
    // Se obtiene el servicio de vibración del sistema, con
    // compatibilidad para versiones nuevas y antiguas de Android.
    val vibrator = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.S) {
        val vibratorManager =
            getSystemService(Context.VIBRATOR_MANAGER_SERVICE) as VibratorManager
        vibratorManager.defaultVibrator
    } else {
        getSystemService(VIBRATOR_SERVICE) as Vibrator
    }

    // Se crea un efecto de vibración de un solo pulso de 200ms con
    // amplitud por defecto.
    vibrator.vibrate(VibrationEffect.createOneShot(200,
```

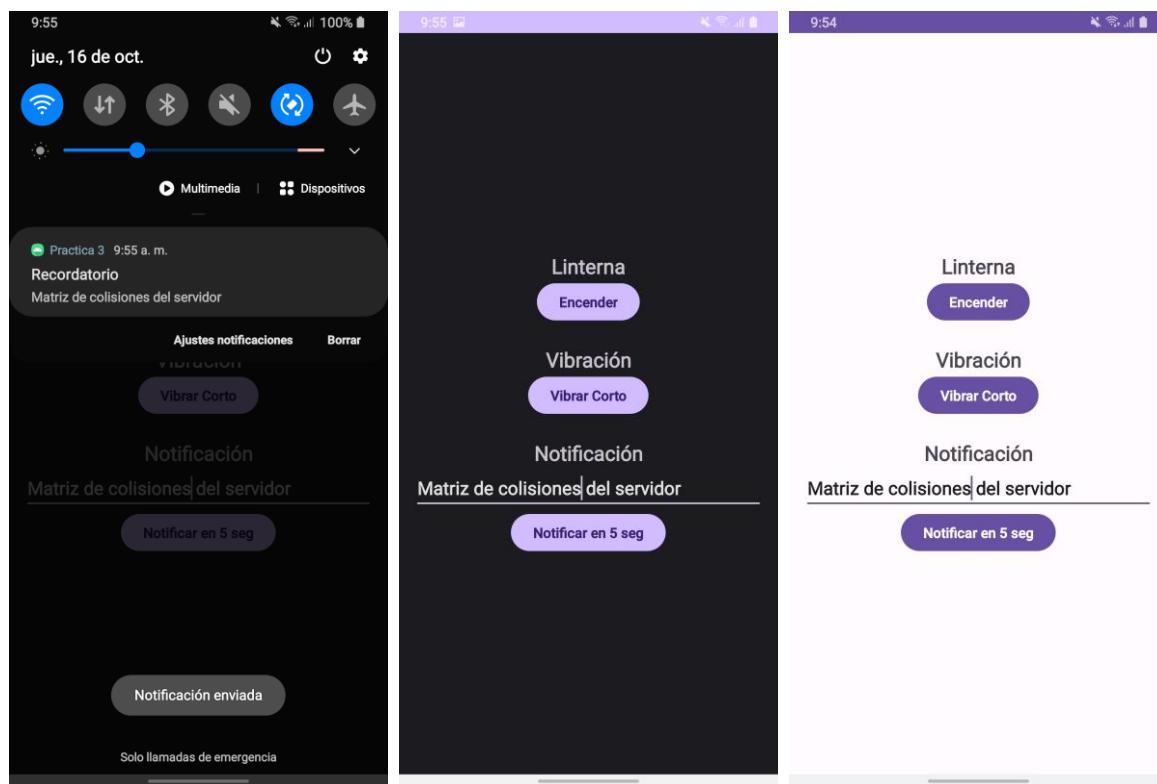
```
VibrationEffect.DEFAULT_AMPLITUDE) )  
}
```

- Notificaciones (API NotificationManager):

```
private fun sendNotification(message: String) {  
    // Se utiliza un NotificationCompat.Builder para construir la  
    // notificación de forma compatible con versiones antiguas.  
    val builder = NotificationCompat.Builder(this, "CHANNEL_ID_1")  
        .setSmallIcon(R.drawable.ic_launcher_foreground) // Icono  
        obligatorio para la notificación.  
        .setContentTitle("Recordatorio")  
        .setContentText(message) // Mensaje personalizado introducido por  
        el usuario.  
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)  
  
    // Se obtiene el NotificationManager y se publica la notificación con  
    // un ID único.  
    val notificationManager =  
        getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager  
        notificationManager.notify(1, builder.build())  
}
```

Interfaz de Usuario (UI)

En esta sección se adjuntan las capturas de pantalla de la aplicación.



Persistencia de Datos

Tecnología Usada: La versión actual de la aplicación no implementa persistencia de datos, ya que su funcionalidad se basa en acciones instantáneas que no requieren guardar información entre sesiones.

Para una futura mejora, se podría utilizar SharedPreferences para guardar el último estado de la linterna (encendido/apagado). Esto permitiría restaurar su estado al volver a abrir la aplicación, mejorando la experiencia del usuario.

Gestión de Permisos

Lista de Permisos y Justificación:

- android.permission.CAMERA: Indispensable para acceder al controlador del flash de la cámara y poder ejecutar la función de linterna.
- android.permission.VIBRATE: Requerido para utilizar el motor de vibración del dispositivo y proporcionar feedback háptico.
- android.permission.POST_NOTIFICATIONS: Obligatorio en Android 13 (API 33) y superior para que la aplicación pueda mostrar notificaciones en la barra de estado.

Implementación del Flujo de Solicitud:

```
// Se registra un contrato para solicitar un permiso y recibir el resultado de forma asíncrona.
private val requestPermissionLauncher =
    registerForActivityResult(ActivityResultContracts.RequestPermission()) { isGranted: Boolean ->
        if (isGranted) {
            // Si el permiso es concedido, se ejecuta la acción deseada (ej: encender linterna).
            toggleFlashlightAndUpdateButton()
        } else {
            // Si es denegado, se podría mostrar un mensaje al usuario explicando la necesidad del permiso.
        }
    }

// Dentro del OnClickListener, antes de usar el recurso, se verifica el estado del permiso.
when {
    ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA) == PackageManager.PERMISSION_GRANTED -> {
        // Si el permiso ya fue concedido, se procede con la acción.
        toggleFlashlightAndUpdateButton()
    }
    else -> {
```

```

        // Si no, se lanza la solicitud de permiso al usuario a través
        del launcher registrado.
        requestPermissionLauncher.launch(Manifest.permission.CAMERA)
    }
}

```

Pruebas Realizadas

Tabla de Dispositivos/Emuladores Utilizados

Dispositivo	Versión de Android	Tipo
Google Pixel 7 Pro	Android 14 (API 34)	Emulador
Samsung Galaxy S10	Android 12 (API 31)	Dispositivo Físico

Casos de Prueba y Resultados

ID	Caso de Prueba	Pasos a Seguir	Resultado Esperado	Resultado Obtenido
1	Permiso de Cámara	1. Iniciar la app por primera vez. 2. Pulsar "Encender".	Se muestra el diálogo del sistema para solicitar permiso de cámara.	Pasa <input checked="" type="checkbox"/>
2	Funcionalidad Linterna	1. Otorgar permiso de cámara. 2. Pulsar "Encender". 3. Pulsar "Apagar".	El flash se enciende y el botón cambia a "Apagar". El flash se apaga y el botón cambia a "Encender".	Pasa <input checked="" type="checkbox"/>
3	Funcionalidad Vibración	1. Pulsar "Vibrar Corto".	El dispositivo vibra una vez.	Pasa <input checked="" type="checkbox"/>
4	Funcionalidad Notificación	1. Escribir "Prueba" en el campo de texto. 2. Pulsar "Notificar en 5 seg".	Una notificación con el título "Recordatorio" y el texto "Prueba" aparece después de 5 segundos.	Pasa <input checked="" type="checkbox"/>

Conclusiones

El desarrollo de esta aplicación permitió consolidar los conocimientos sobre la interacción con APIs nativas de Android, un pilar fundamental del desarrollo móvil. La implementación de funcionalidades que acceden directamente al hardware del dispositivo subraya la importancia de un manejo cuidadoso de los recursos y del ciclo de vida de los componentes de Android.

El principal aprendizaje obtenido fue la gestión moderna de permisos en tiempo de ejecución. Comprender el flujo asíncrono de solicitud y respuesta (ActivityResultContracts) es crítico para crear aplicaciones seguras y que ofrezcan una buena experiencia de usuario, respetando su privacidad.

La dificultad más significativa durante el desarrollo fue la gestión de errores de sincronización de Gradle en Android Studio, que esporádicamente impedían la generación de la clase R, causando errores en cascada. La solución recurrente fue utilizar la función "File > Invalidate Caches / Restart" del IDE, lo que forzaba una reconstrucción limpia del proyecto y resolvía los problemas de indexación.

Referencias Bibliográficas

- Google. (2025). Documentación oficial para desarrolladores de Android. developer.android.com. Obtenido de <https://developer.android.com>
- Google. (2025). CameraManager. Android Developers. Obtenido de <https://developer.android.com/reference/android/hardware/camera2/CameraManager>
- Google. (2025). Crear una notificación. Android Developers. Obtenido de <https://developer.android.com/training/notify-user/build-notification>
- Google. (2025). Solicitar permisos de la app. Android Developers. Obtenido de <https://developer.android.com/training/permissions/requesting>