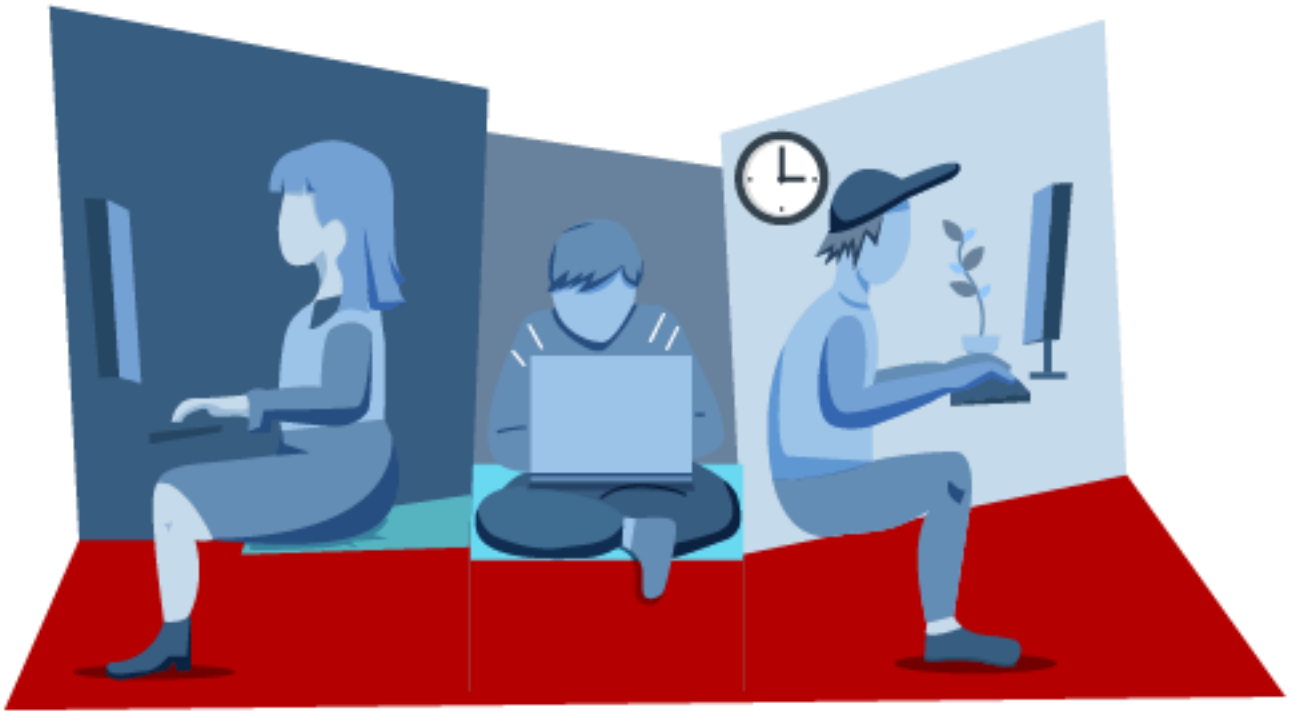
Mapa conceptual sobre la arquitectura del sistema operativo Android y el NDK



MÓDULO: PRO401-9523-225081-ONL-TALLER DE APLICACIONES MÓVILES

SEMANA: Semana 2

Docente: IVAN AYALA AYALA

Estudiante: Matías Pérez Nauto

Resumen

El presente informe propone una exploración estructurada de la arquitectura del sistema operativo Android mediante la elaboración de un mapa conceptual que sintetiza sus componentes clave y sus relaciones funcionales. La hipótesis central plantea que comprender la arquitectura modular de Android —desde el kernel hasta el framework de aplicaciones— permite al estudiante desarrollar habilidades técnicas y cognitivas esenciales para el diseño, optimización y documentación de aplicaciones móviles. La estructura del informe se organiza en torno a cinco capas principales: Kernel de Linux, Librerías nativas, Android Runtime, HAL (Hardware Abstraction Layer) y Framework de aplicaciones. Cada una de estas capas se ramifica en nodos específicos que explican funciones como la gestión de procesos, memoria y seguridad; el uso de bibliotecas compartidas (.so), OpenGL ES y libc; la ejecución de código mediante ART, AOT/JIT y recolección de basura; la traducción de hardware como cámara, sensores y audio; y la interacción con el usuario a través de actividades, vistas y proveedores de contenido. El mapa conceptual se complementa con explicaciones breves y naturales que permiten una lectura clara y trazable, alineada con criterios académicos. Como aporte, el trabajo fortalece la capacidad de análisis, síntesis y comunicación técnica del estudiante, además de proyectar competencias aplicables en entornos laborales que demandan precisión, modularidad y sensibilidad por la experiencia humana en tecnología. La conclusión destaca el valor de esta actividad como herramienta formativa que vincula teoría y práctica, consolidando una base sólida para el desarrollo profesional y personal del estudiante en el ámbito del desarrollo móvil y la arquitectura de sistemas.

Índice

Introducción

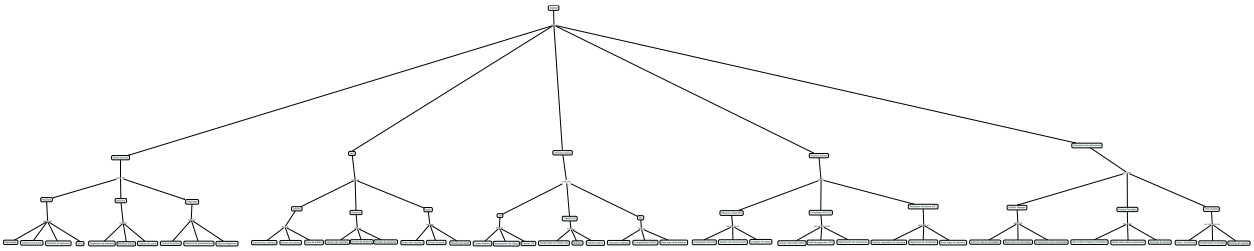
El presente informe tiene como propósito analizar y representar la arquitectura del sistema operativo Android mediante un mapa conceptual que sintetice sus componentes fundamentales y su funcionamiento interno. Esta actividad se enmarca en el estudio de las aplicaciones móviles y su relación directa con las capas estructurales del sistema, permitiendo al estudiante comprender cómo se articulan elementos como el kernel, las librerías nativas, el entorno de ejecución y el framework de aplicaciones para dar soporte a las apps desarrolladas en Java o Kotlin.

La arquitectura de Android se caracteriza por su modularidad y por la interacción entre componentes que operan en distintos niveles de abstracción. Desde el núcleo del sistema —basado en Linux— hasta el framework que expone APIs para el desarrollo de interfaces y funcionalidades, cada capa cumple un rol específico en la ejecución, gestión de recursos y experiencia del usuario. En este contexto, el mapa conceptual elaborado busca representar de forma clara y ordenada esta estructura, destacando nodos clave como Android Runtime (ART), el compilador AOT/JIT, el recolector de basura, el gestor de actividades y el sistema de vistas, entre otros.

Además, se incorporan conceptos relacionados con el NDK (Native Development Kit), que permiten extender las capacidades del sistema mediante código nativo, optimizando el rendimiento en aplicaciones que requieren procesamiento intensivo. La actividad fomenta el pensamiento crítico, la capacidad de síntesis y la trazabilidad entre componentes, habilidades esenciales en la formación de un programador y analista de sistemas.

Este documento incluye el mapa conceptual, la descripción breve y natural de cada concepto, y una reflexión final sobre el aporte de esta actividad al desarrollo académico y profesional del estudiante.

Desarrollo



Explicación de cada nodo

Kernel de Linux: Es el núcleo del sistema Android. Se encarga de conectar el software con el hardware, gestionando procesos, memoria y recursos del dispositivo. Es la base sobre la que se construyen las capas nativas y el entorno de ejecución.

- **Procesos**: El kernel administra la ejecución de programas, asignando recursos y controlando su ciclo de vida.
    - Planificador (Scheduler): Decide qué proceso se ejecuta y cuándo, según prioridades y estados.
    - Estados de proceso: Define si un proceso está activo, suspendido, esperando o terminado.
    - Contexto de ejecución: Conjunto de registros y memoria que definen el estado de un proceso.
- **Memoria**: Controla el uso de la RAM, asignando espacio a cada proceso y evitando conflictos.
    - Asignación de memoria: El kernel distribuye espacio en RAM a cada proceso según sus necesidades.
    - Memoria virtual: Permite que cada proceso tenga su propio espacio lógico, aislado y seguro.
    - Gestión de caché: Optimiza el acceso a datos frecuentes guardándolos temporalmente en memoria.
- **Seguridad**: Aplica permisos y políticas para proteger el sistema y los datos del usuario.

    - Control de acceso: Define qué procesos pueden acceder a qué recursos, usando permisos y credenciales.
    - Espacios de usuario/kernel: Aísla el código del usuario del núcleo para evitar interferencias maliciosas.
    - SELinux/AppArmor: Módulos de seguridad que aplican políticas estrictas para proteger el sistema.

HAL (Hardware Abstraction Layer): Es una capa que permite que el sistema Android se comunique con el hardware del dispositivo sin depender de sus detalles específicos, facilitando la compatibilidad entre diferentes fabricantes.

- **Cámara:** Permite capturar imágenes y video desde el hardware, accesible vía HAL.
    - Camera HAL Interface: Define funciones estándar para capturar imágenes, controlar enfoque y exposición.
    - Drivers de cámara: Traducen las llamadas HAL en instrucciones específicas para el hardware del fabricante
    - Flujo de buffers: Gestiona el envío de datos de imagen desde el sensor al sistema, optimizando rendimiento.
- **Sensores:** Incluye acelerómetro, giroscopio, proximidad, entre otros, para captar el entorno.
    - Sensor HAL Interface: Define funciones estándar para acceder a datos como aceleración, orientación o luz.
    - Drivers de sensores: Traducen las llamadas HAL en instrucciones específicas para cada tipo de sensor físico.
    - Eventos de sensores: Sistema que entrega datos en tiempo real al framework Android, optimizando respuesta y eficiencia.
- **Audio:** Controla entrada (micrófono) y salida (altavoces) de sonido.
    - Audio HAL Interface: Define funciones estándar para reproducir, grabar y controlar volumen desde el hardware.
    - Drivers de audio: Traducen las llamadas HAL en instrucciones específicas para el chip de sonido del dispositivo.
    - Streams de audio: Gestionan los flujos de entrada/salida (micrófono, altavoces), optimizando calidad y sincronización.

Librerías nativas: Son archivos compilados en C/C++ (.so o .a) que permiten mejorar el rendimiento de la app y reutilizar código existente, especialmente útil en tareas exigentes como gráficos o procesamiento intensivo.

- **.so (Shared Object):** Son bibliotecas compartidas que se cargan en tiempo de ejecución, permitiendo reutilizar código nativo en distintas partes de la app.
    - **Carga dinámica:** Las bibliotecas .so se cargan en tiempo de ejecución, lo que permite modularidad y ahorro de memoria.
    - **Reutilización de código:** Permiten compartir funciones nativas entre distintas partes de la app sin duplicar lógica.
    - **Interfaz JNI:** Actúan como puente entre Java/Kotlin y C/C++, facilitando llamadas a funciones nativas.
- **OpenGL ES:** Librería gráfica optimizada para dispositivos móviles, usada para renderizar gráficos 2D y 3D en tiempo real.
    - **Renderizado en tiempo real:** Permite dibujar gráficos 2D y 3D de forma eficiente, ideal para juegos y visualizaciones interactivas.
    - **Shaders:** Programas que controlan cómo se dibujan los píxeles y vértices, optimizando efectos visuales.
    - **Pipeline gráfico:** Flujo estructurado que transforma datos en imágenes en pantalla, desde geometría hasta color final.
- **libc:** Biblioteca estándar del lenguaje C, que ofrece funciones básicas como manejo de memoria, cadenas y archivos.
    - **Funciones estándar:** Proporciona operaciones básicas como manejo de memoria, cadenas, archivos y matemáticas.
    - **Compatibilidad POSIX:** Implementa parte del estándar POSIX, facilitando portabilidad entre sistemas tipo Unix.
    - **Base para otras librerías:** Muchas bibliotecas nativas dependen de libc para funciones esenciales, actuando como núcleo funcional.

Android Runtime (ART): Es el entorno que ejecuta las apps Android. Convierte el código Java en instrucciones que el sistema puede entender, optimizando el rendimiento mediante técnicas como AOT y recolección de basura.

- **Máquina virtual ART**: Ejecuta las apps Android transformando el código Java en instrucciones nativas, reemplazando a la antigua máquina Dalvik.
    - **Compilación AOT/JIT:** ART usa AOT para compilar antes de ejecutar y JIT para optimizar en tiempo real según el uso.
    - **Gestión de memoria (GC):** ART incluye un recolector de basura que libera memoria automáticamente y evita fugas.
    - **Cargador de clases:** Carga dinámicamente las clases necesarias para ejecutar la app, según demanda y contexto.
- **Compilador AOT/JIT:** AOT compila antes de ejecutar; JIT optimiza durante la ejecución según el uso.
    - Código intermedio (DEX): Es el formato que Android genera a partir del código Java, antes de compilarlo a nativo.
    - Perfil de ejecución (JIT): Registra qué métodos se usan con frecuencia para optimizar su compilación en tiempo real.
    - Optimización de rendimiento: AOT mejora el arranque de la app; JIT reduce el uso de recursos al adaptar la compilación.
- **Recolector de basura (GC)**: Libera memoria automáticamente al eliminar objetos que ya no se usan, evitando fugas de memoria.
    - Gestión automática de memoria: El GC identifica y elimina objetos que ya no se usan, liberando espacio sin intervención manual.
    - Algoritmos de recolección: ART usa algoritmos como *generational* y *concurrent* para equilibrar eficiencia y rendimiento.
    - Evita fugas de memoria: Al eliminar referencias obsoletas, previene que la app consuma más memoria de la necesaria.

Framework de aplicaciones: Es el conjunto de APIs y servicios que permite construir apps Android en Java o Kotlin, facilitando el acceso a componentes como actividades, vistas, notificaciones y sensores

- **Activity Manager:** Administra el ciclo de vida de las actividades y la navegación entre pantallas.
    - Ciclo de vida de actividades: Supervisa estados como creación, pausa, reanudación y destrucción de cada pantalla.
    - Back stack de actividades: Mantiene un historial de navegación que permite volver a pantallas anteriores.
    - Transiciones entre actividades: Coordina el paso de una actividad a otra, incluyendo animaciones y paso de datos.
- **Content Providers:** Facilitan el acceso compartido a datos entre aplicaciones de forma segura.
    - URI (Uniform Resource Identifier): Identifica de forma única los datos que se comparten entre apps, como si fueran direcciones.
    - CRUD sobre datos compartidos: Permiten realizar operaciones de Crear, Leer, Actualizar y Eliminar sobre datos externos.
    - Permisos de acceso: Controlan qué apps pueden acceder a qué datos, garantizando seguridad y privacidad.
- **View System:** Define y renderiza los elementos visuales de la interfaz de usuario.
    - View y ViewGroup: Son los bloques básicos de la interfaz: *View* representa un elemento visual, *ViewGroup* los organiza.
    - Renderizado en pantalla: Convierte los elementos definidos en XML o código en gráficos visibles y táctiles.
    - Eventos de usuario: Captura interacciones como toques, gestos o desplazamientos, y los traduce en acciones.

Conclusión

La realización de este trabajo ha representado un aporte significativo en tres dimensiones complementarias: académica, laboral e individual. En el ámbito académico, permitió consolidar conocimientos sobre la arquitectura de Android, articulando conceptos técnicos mediante un mapa conceptual que favorece la comprensión estructural y la trazabilidad entre componentes. Esta capacidad de sintetizar y representar información compleja es clave en la formación como programador y analista de sistemas, ya que fortalece habilidades de análisis, documentación y comunicación técnica.

Desde una perspectiva laboral, el ejercicio de descomponer y justificar cada nodo del sistema operativo Android prepara al estudiante para enfrentar escenarios reales de desarrollo y mantenimiento de aplicaciones móviles. Comprender cómo interactúan el kernel, el entorno de ejecución y el framework de aplicaciones permite tomar decisiones más informadas en proyectos que requieren optimización, modularidad y escalabilidad. Además, el enfoque en hospitalidad digital y experiencia de usuario refuerza competencias valoradas en entornos profesionales actuales.

A nivel individual, el trabajo fomenta el pensamiento crítico, la autodisciplina y la mejora continua. La elaboración del mapa conceptual y la redacción del informe implicaron un proceso de reflexión profunda, validación de fuentes y estructuración clara de ideas, lo que contribuye al desarrollo de una marca personal basada en autenticidad, precisión y sensibilidad por la experiencia humana en tecnología.

En conjunto, esta actividad no solo cumple con los objetivos académicos, sino que proyecta habilidades transferibles al mundo laboral y fortalece el crecimiento personal del estudiante como agente de cambio en su entorno local.

Bibliografía

Android Open Source Project. (2025). Android Runtime (ART). Mountain View, Estados Unidos: Google Inc. Recuperado de https://source.android.com/docs/core/ota/modular-system/art

Goel, M., & Singal, G. (2021). Android OS Case Study: Architecture and Runtime. Nueva Delhi, India: arXiv.org, capítulo 2, páginas 4–5. Recuperado de https://arxiv.org/pdf/2104.09487

Android Developers. (2025). Guide to App Architecture. Mountain View, Estados Unidos: Google Inc., sección Framework. Recuperado de https://developer.android.com/topic/architecture