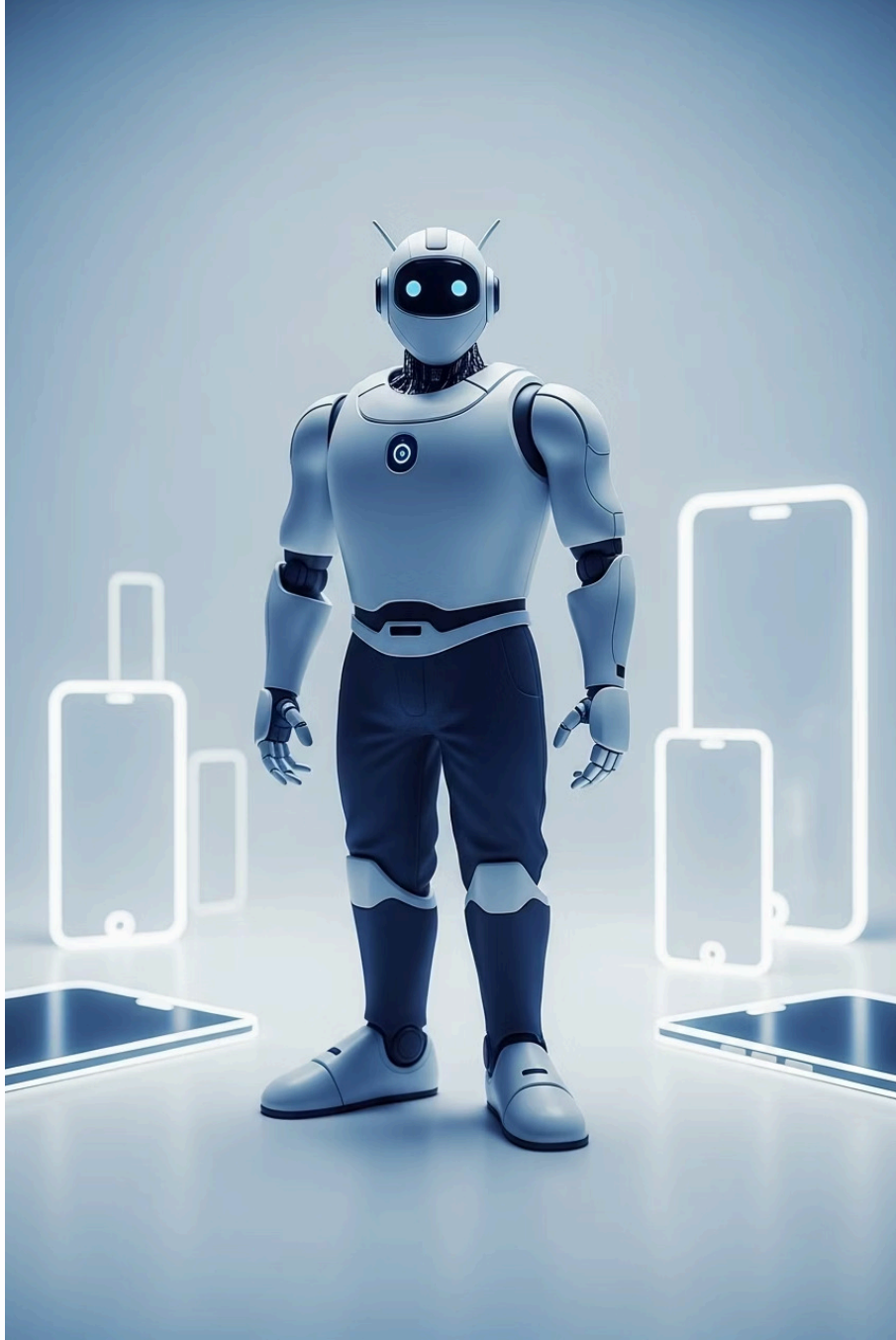
Android: From Origins to Architecture and File Structure

This presentation explores the fascinating journey of Android from its inception to its current status as the world's dominant mobile operating system. We'll examine the architectural foundations that power billions of devices worldwide and delve into the file structure that organises this complex ecosystem.



Chapter 1

The Birth and Evolution of Android

Android's journey from a small startup to the world's most popular mobile operating system is a remarkable story of innovation, strategic vision, and open-source development. This section explores how Android began and how it grew to dominate the global smartphone market.

The Android Genesis: 2003 to 2008

Android was founded by four visionaries with a bold idea: to create an open, accessible mobile platform that could run on any device. Unlike Apple's closed ecosystem, Android was designed to be adaptable and customisable from the start.

The first commercial version, Android 1.0, launched alongside the HTC Dream (T-Mobile G1) in 2008. This marked the beginning of what would become a global technological revolution.

Founded in 2003

By Andy Rubin, Rich Miner, Nick Sears, and Chris White

First Release in 2008

Android 1.0 on 23 September

Open-Source Vision

Linux-based OS for touchscreen devices

Android's Rise to Dominance

Global Leadership

World's most used smartphone OS since 2011, overtaking Symbian, iOS, and BlackBerry OS

Massive User Base

Over 3 billion monthly active users as of 2025, representing over 70% of global market share

Continuous Innovation

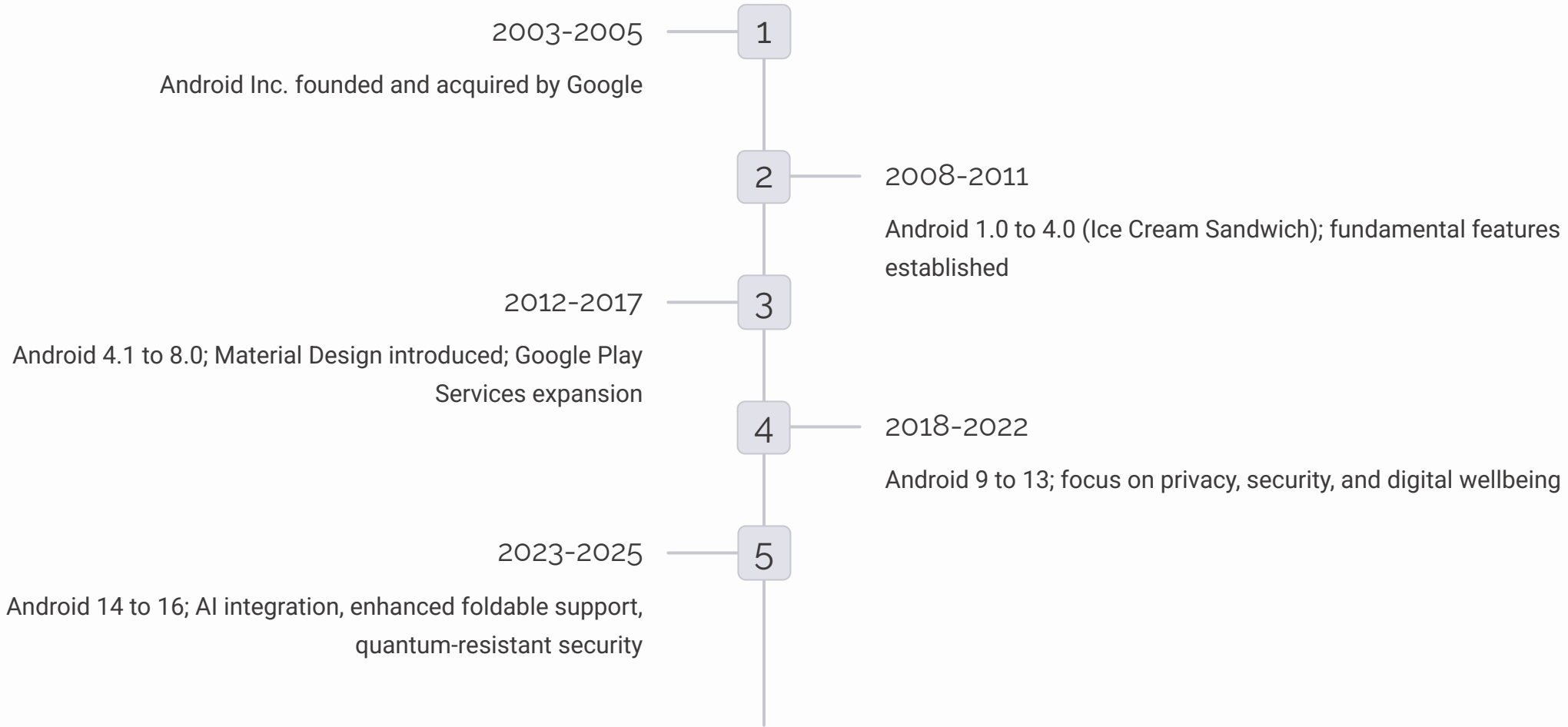
Latest release: Android 16 (June 10, 2025) with enhanced AI capabilities and improved privacy controls

Diverse Ecosystem

Beyond phones: tablets, TVs, watches, cars, IoT devices running modified Android versions

Android's success stems from its adaptability, allowing manufacturers to customise the OS while maintaining compatibility with the broader app ecosystem. This approach has enabled brands like Samsung, Xiaomi, and Google to create distinct experiences whilst sharing the same foundation.

Android's Evolution: Key Milestones



Chapter 2

Android Architecture - The Software Stack

Android's architecture is designed as a sophisticated software stack, with each layer serving specific functions whilst communicating with the layers above and below it. This modular approach enables flexibility, security, and performance optimization.

The Linux Kernel: Android's Foundation

At Android's core lies a modified version of the Linux kernel—the same robust foundation that powers many servers, supercomputers, and embedded systems worldwide. Google has customised this kernel specifically for mobile environments, focusing on power efficiency, resource management, and security.

The Linux kernel manages the most fundamental system operations: hardware drivers, process management, memory allocation, networking, and security enforcement. It provides the critical abstraction layer between physical hardware and the software stack above.

Core Functions

- Process management
- Memory management
- Device driver architecture

Security Features

- Process isolation
- Permission model
- SELinux implementation

Mobile Optimisations

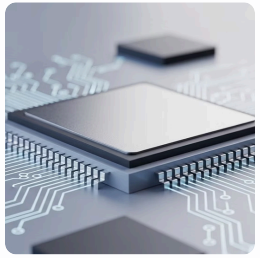
- Power management
- Low memory killer
- Wake locks

Hardware Abstraction Layer (HAL)

The Hardware Abstraction Layer (HAL) bridges the gap between hardware-specific code and the platform-agnostic Android system. This critical middleware allows Android to run consistently across thousands of different device models with varying hardware components.

Camera HAL

Standardises access to camera hardware features regardless of sensor manufacturer or capabilities



Bluetooth HAL

Enables consistent Bluetooth functionality across devices with different radio chips

Biometric HAL

Provides standard interfaces for fingerprint, face, and iris recognition hardware

Without the HAL, device manufacturers would need to heavily modify Android for each hardware configuration, fragmenting the ecosystem and complicating app development.

Android Runtime (ART)

The Android Runtime (ART) is the execution environment where applications run. It replaced the original Dalvik Virtual Machine in Android 5.0 (Lollipop), bringing significant performance improvements and better battery efficiency.

ART employs a combination of compilation strategies to optimise application performance. While Dalvik used Just-in-Time (JIT) compilation exclusively, ART primarily uses Ahead-of-Time (AOT) compilation during installation, supplemented with JIT compilation for dynamic code paths.

Each application runs in its own isolated process with a dedicated instance of ART, enhancing security through process isolation whilst enabling concurrent execution.

Compilation Approaches

- Ahead-of-Time (AOT) for better startup
- Just-in-Time (JIT) for dynamic code
- Profile-guided optimisations

Memory Management

- Concurrent, compacting garbage collector
- Reduced memory fragmentation
- Lower GC pauses during rendering

Performance Benefits

- Faster app startup times
- Reduced battery consumption
- Improved multitasking capability

Native Libraries and Platform Libraries

Core C/C++ Libraries

- SQLite: Lightweight relational database
- OpenGL/Vulkan: 3D graphics rendering
- WebKit: Web content rendering engine
- Media Framework: Audio/video codecs

Java Platform Libraries

- java.* packages: Core Java functionality
- javax.* packages: Java extension APIs
- android.* packages: Android-specific APIs
- androidx.* packages: Jetpack libraries

Security Libraries

- BoringSSL: Secure communication
- Conscrypt: Cryptographic operations
- Keystore: Credential management
- Biometric: Authentication frameworks

These libraries provide essential services to both the Android framework and applications. They implement common tasks efficiently, allowing developers to focus on application-specific logic rather than reinventing fundamental components.

Many of these libraries are open-source projects maintained by the broader developer community, exemplifying Android's collaborative development model.

Application Framework Layer

The Application Framework layer provides high-level services in the form of Java classes that developers use to build applications. These standardised components handle common tasks and ensure consistent behavior across the Android ecosystem.

Activity Manager

Controls application lifecycle, manages back stack, and handles task switching

Notification Manager

Enables apps to display alerts, updates, and messages to users consistently

Content Providers

Facilitates structured data sharing between applications with permission controls

Window Manager

Manages application windows, transitions, and multi-window operations

The framework layer essentially serves as the API that developers interact with most frequently. It encapsulates the complexity of lower layers whilst providing a rich set of tools for creating compelling applications. By standardising these components, Android ensures consistency in user experience whilst simplifying development.

Applications Layer: The User Experience

The Applications layer is what users directly interact with—the apps that make Android valuable. This layer includes both pre-installed system applications and third-party applications downloaded from the Google Play Store or other sources.

Applications are primarily written in Java or Kotlin, though they may include native C/C++ code for performance-critical components. All app code is compiled to run on the Android Runtime (ART), which translates bytecode into machine instructions.

System Applications

Pre-installed apps like Phone, Contacts, Messages, Settings, and Camera that provide core functionality

Google Applications

Google-provided apps like Gmail, Maps, Photos, and YouTube that integrate Google services

Third-Party Applications

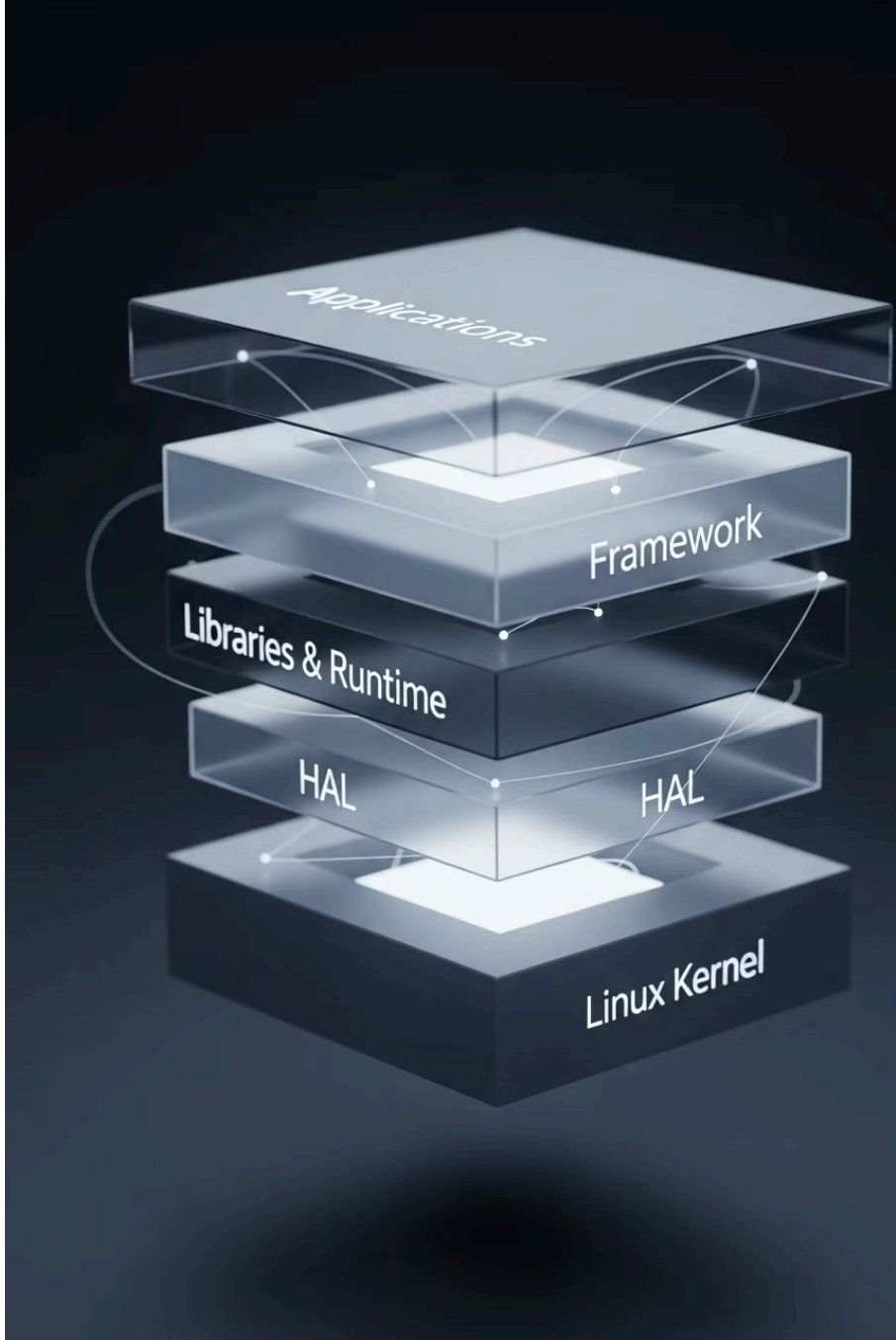
Over 3.5 million apps on Google Play Store spanning games, productivity, entertainment, and utilities

Android Architecture: The Complete Picture

Android's layered architecture enables remarkable flexibility while maintaining security and performance. Each layer has clear responsibilities and interfaces, allowing components to evolve independently without disrupting the entire system.

This architecture also facilitates the Android compatibility program, which ensures that applications work consistently across thousands of different device models from hundreds of manufacturers. By providing hardware abstractions and standardised APIs, Android achieves the seemingly contradictory goals of hardware diversity and application compatibility.

> The architecture has evolved significantly since Android's inception, with major enhancements in runtime performance (ART), security model (permissions, SELinux), and modularisation (Project Treble, Project Mainline).



Chapter 3

Android File System Structure

Understanding Android's file system structure is essential for developers, system administrators, and power users. The structure reflects Android's security model, with clearly defined partitions for system files, applications, and user data.

Root Directory and Core Partitions

Android's file system follows a hierarchical structure similar to other Unix-like systems but with specific modifications for mobile devices. The system is divided into several partitions, each mounted at specific locations in the directory tree and serving distinct purposes.

Many partitions are mounted read-only during normal operation to prevent modification, enhancing security and stability. System updates typically involve remounting these partitions as read-write temporarily or replacing them entirely.

This partitioning scheme also facilitates the A/B system updates introduced in Android 7.0, where two complete sets of system partitions exist to enable seamless updates.

/ (Root)

The top-level directory containing all others; the starting point of the file system hierarchy

/boot

Contains the kernel image and bootloader files necessary to start the device

/system

Read-only partition housing the Android OS, framework, and pre-installed system applications

/recovery

Alternative boot partition with tools for system recovery, updates, and factory resets

User and App Data Storage

/data

- Private app data in /data/data/[package_name]/
- App databases, preferences, and cache files
- User accounts and settings
- Encrypted with device credentials

/cache

- Temporary system cache files
- Cleared during system updates
- Used for system recovery operations
- Not for app-specific caching

/sdcard or /storage

- User-accessible storage for media files
- Photos, videos, documents, downloads
- May be internal flash or external SD card
- Accessible through Media Store APIs

Android's storage model has evolved significantly over time, particularly with the introduction of Scoped Storage in Android 10. This change restricted direct filesystem access for privacy and security reasons, requiring apps to use content providers and media store APIs instead of direct file paths.

Android Project Folder Structure (Development Perspective)

When developing Android applications, you'll work with a standardised project structure enforced by Android Studio. This organisation separates code, resources, and configuration files, facilitating easier maintenance and collaboration.

The separation of code and resources enables powerful capabilities like alternative resources for different screen sizes, languages, and device configurations. The build system automatically selects the appropriate resources at runtime based on the device characteristics.

Modern Android projects typically use the Gradle build system, which defines how resources are compiled, code is processed, and the final application package is assembled.

manifests/

Contains AndroidManifest.xml with app metadata, components, and permissions

java/ or kotlin/

Source code files organised by package structure

res/

Resource files: layouts, drawables, strings, styles, animations, and more

gradle/

Build configuration scripts and dependency management

Android Studio Project Structure

Android Studio organises projects to optimise developer workflow. The Project view can be toggled between "Android" view (which groups files by logical function) and "Project" view (which shows the actual filesystem hierarchy).

res/layout/

XML files defining UI screen layouts, dialogs, and custom views

res/drawable/

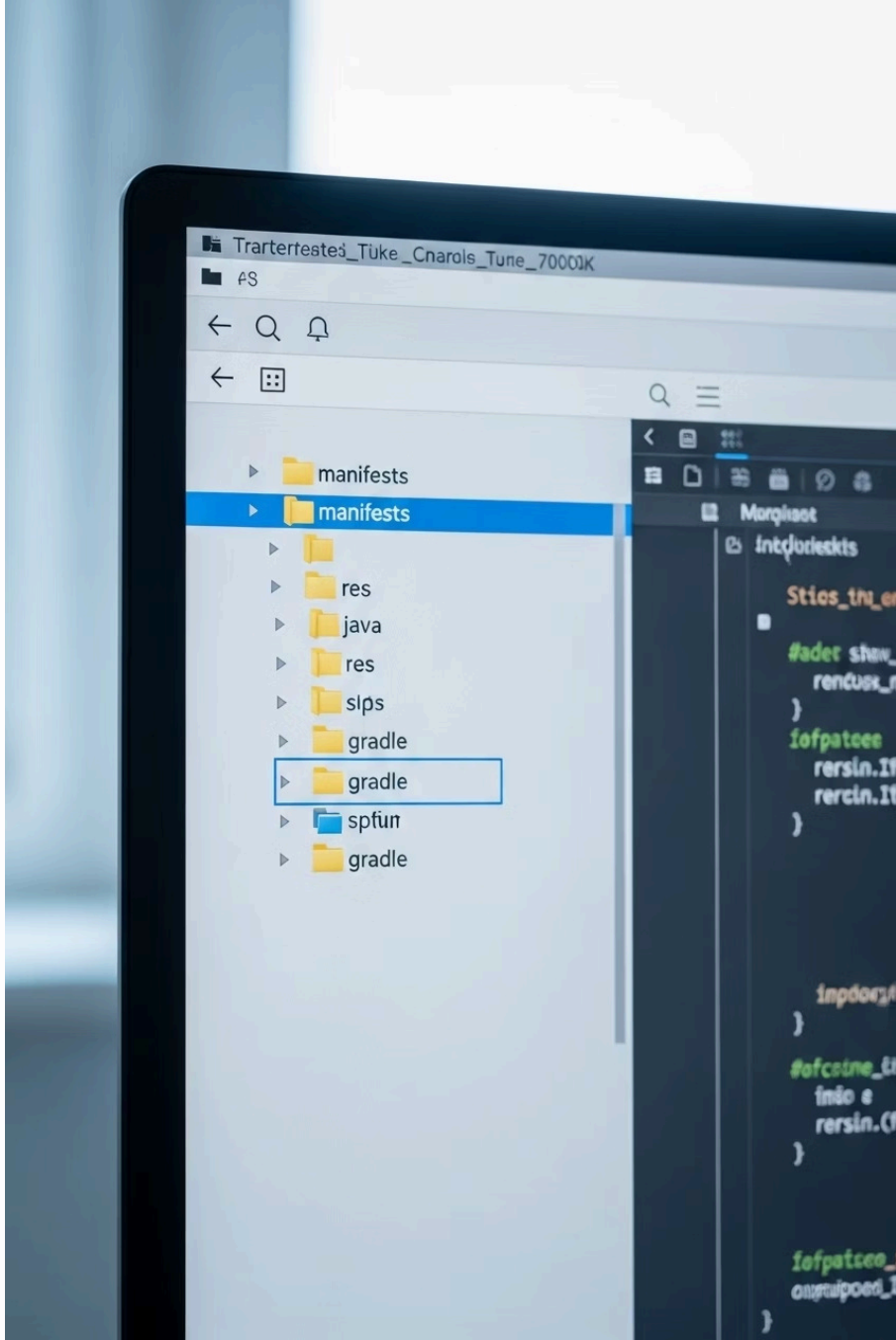
Images, shapes, and animations displayed in the user interface

res/values/

String resources, dimensions, colors, styles, and themes for consistent UI

res/mipmap/

App icons in various densities for different screen resolutions



Why Understanding Architecture and File Structure Matters

Development Efficiency

Navigate frameworks, APIs, and storage efficiently. Debug issues by knowing where to look for logs and system information.

Custom ROM Development

Modify the Android operating system to create custom distributions with enhanced features or specialised use cases.

Performance Optimisation

Make informed decisions about caching, thread usage, and resource access based on how Android manages processes and memory.

System Troubleshooting

Diagnose and resolve device issues by examining logs, checking permissions, and understanding system partitions.

Security Implementation

Understand the permission model, sandbox restrictions, and encryption mechanisms to build secure applications.

Innovation Potential

Discover opportunities to enhance or extend Android in ways the original designers may not have anticipated.

For enterprise app developers, system integrators, and device manufacturers, this knowledge is not just academic—it's essential for creating robust, efficient, and secure Android-based solutions.

The Android Journey Continues

Android's evolution continues as technology advances. From its modest beginnings as a startup project to becoming the world's most widespread operating system, Android demonstrates the power of open-source development and collaborative innovation.

Open Ecosystem

Android's open-source foundation continues to fuel innovation across manufacturers, enabling device diversity and accessibility at all price points

Evolving Architecture

Project Mainline, Treble, and future initiatives improve updatability, security, and performance whilst maintaining compatibility

Developer Gateway

With modern Kotlin-first development, Jetpack libraries, and comprehensive tools, Android remains the platform of choice for mobile innovation