

Advanced Kotlin for Android: Architecture, ML and Security

A two-day advanced lab with focused implementation and review

FA-0549 · 2 days

Software Development · Cybersecurity

Review Android architecture and implement selected ML, security and performance improvements in an experienced-developer lab.

Introduction

This two-day advanced workshop examines maintainable Kotlin/Android applications through architecture, state management, ML integration, security and performance. It assumes working Android experience and focuses on selected implementation exercises supported by design reviews.

Participants review an existing or illustrative application, practise a focused inference or integration workflow and investigate chosen security/performance concerns. Broader testing, delivery and device-compatibility topics provide follow-on guidance; the workshop does not promise a production-ready, attack-proof app or large-scale capacity.

Learning outcomes

The course teaches participants to:

- Organise Android presentation, domain and data responsibilities with appropriate module boundaries.
- Use ViewModel state, lifecycle-aware cancellation and background-work boundaries correctly.
- Compare cloud and on-device inference and integrate a selected model or ML service.
- Review transport, permissions, sensitive-data handling, authentication and dependency risks.
- Profile a chosen performance issue and apply coroutine, rendering, memory or packaging improvements.

- Plan tests for architecture, ML behaviour and compatibility rather than assume quality from successful compilation.

Prerequisites

- Working experience with Kotlin (self-taught or otherwise), including coroutines.
- Basic Android development experience: activities/fragments or composables, views, basics of lifecycle.
- Familiarity with REST APIs and JSON.
- Basic understanding of ML/AI concepts (what a model is, inference vs training).

A working, compatible Android Studio/JDK/Gradle environment with an emulator or device and permission to install training dependencies. Model examples and runtime versions are checked for the chosen device; the course is not an introduction to Kotlin or Android.

Training outline

1. Day 1 — Architecture, State and Security

Code Architecture and MVVM

Clean architecture versus more ad-hoc architecture

- Layers: Presentation, Domain, Data
- Repositories, Use Cases / Interactors
- Models vs DTOs vs Entities
- Dependency inversion, interface segregation

Modularization

- Feature modules, core modules (e.g. network, util)
- Gradle module dependencies: api vs implementation vs compileOnly etc.
- Reuse, sharing code, versioning of modules

Project & Package Organization

- Best practices for folder/package structure
- Consistent naming conventions, visibility modifiers, internal vs public APIs
- Avoiding God classes; keeping code decoupled

MVVM Pattern Deep Dive

- Role of Model, View, ViewModel; responsibilities
- State management: "single ViewState" vs many mutable LiveData/StateFlow etc.
- Handling UI events, side-effects, navigation cleanly in MVVM

- Data binding (if using XML) vs Compose (if using Jetpack Compose)

ViewModel Best Practices

- Scoping correctly to screens, not holding onto references to Context, Views, Resources.
- Using viewModelScope, structured concurrency; cancellation.
- SavedStateHandle and lifecycle-aware state restoration.
- Screen-related data fetching and caching; distinguish ViewModel work from persistent background tasks that need a separate lifecycle.

Mobile Security Review

Fundamental security guidelines (Android best practices)

- HTTPS/TLS and correct trust configuration. Discuss certificate-pinning operational risks rather than recommend it as a universal Android default.
- Limiting permissions; runtime permissions; minimal permissions principle
- Sensitive-data storage and key management using appropriate platform cryptography and Android Keystore; do not present deprecated EncryptedSharedPreferences as the default recommendation.

Obfuscation / Code protection

- ProGuard / R8 usage (shrinking, obfuscation, resource shrinking)
- Avoiding metadata exposure; handling Kotlin metadata issues.
- Android root and tamper-detection signals: limitations and their role in a broader security design, not a guarantee of device integrity.

Secure authentication & authorization

- Token storage, refresh, session management
- Avoiding exposing secrets in code / assets
- Keeping dependencies up to date; using safe cryptography libraries
- Secure error handling, logging (avoid logging secrets), crash reporting considerations

2. Day 2 — ML Integration and Performance

Cloud and On-Device ML

Overview: On-device vs Cloud AI/ML

- When to use what; trade-offs (latency, cost, privacy)

Using cloud-based AI APIs

- Approved cloud-inference APIs or a custom backend: authentication, data transfer and error handling.
- Best practices (throttling, batching, retries, fallback)

Differentiate prebuilt ML Kit capabilities, Teachable Machine model creation/export and the LiteRT on-device runtime (formerly TensorFlow Lite).

- How to train simple models (image, audio, classification) via Teachable Machine; export, integrate into Android app
- LiteRT/TensorFlow Lite-compatible models: quantisation, mobile optimisation and model-update considerations.
- On-device inference, handling input/output, threading, resource constraints

Model lifecycle & version management

- How to package models, update models (via app update, or dynamic delivery), fallback if model fails
- Handling permissions, privacy of input data

AI & ML in MVVM / clean architecture

- Where ML inference code belongs (data layer or domain)
- Testing AI integration

Profiling and Performance

Kotlin language and code level optimizations

- Use of val over var, avoiding !, avoiding unnecessary object allocations, use of inline functions carefully etc.
- Coroutine best practices: dispatchers (Main vs IO vs Default), structured concurrency, avoiding blocking operations on main thread
- Efficient view rendering, using Compose optimally if using it; avoiding over-recomposition; use of keys, remember, derivedState etc.
- List rendering (RecyclerView optimizations or Compose lazy lists), diffing, paging

Memory usage, profiling

- Detecting leaks, use of leak detection tools
- Reducing memory footprint of bitmaps, caching strategies, avoiding large allocations

App size reduction

- Shrinking, resource optimization, splitting APKs or using Android App Bundle

Threading and concurrency

- Async task patterns; avoiding race conditions; proper error propagation
- Startup performance, cold & warm starts

Quality, Delivery and Compatibility — Follow-On Practice

Testing strategies

- Unit tests for ViewModels, Use Cases, data repositories
- Integration tests (e.g. API + parsing)
- UI tests (Espresso or Compose UI tests)
- ML model testing (accuracy, robustness, fallback)

Continuous Integration / Deployment considerations

- Automated builds, code style enforcement (linters), static code analysis
- Code review practices; architecture review; performance regression tests

Logging, Monitoring & Debugging

- Use available Android profiling tools for selected memory, CPU, rendering or network investigations; capabilities depend on the device and tooling.
- Crash reporting, analytics, handling user error reports
- Versioning and backward/forward compatibility
- Handling different device configurations: foldables, various densities, locales