

Intermediate Kotlin for Android

Architecture, security, performance and maintainability

FA-0636 · 3 days

Software Development · Cybersecurity

Improve a prepared Kotlin Android application through architecture review, selected security practices, profiling and tests.

Introduction

This three-day course is for practising Kotlin Android developers who understand coroutines, app lifecycle and API integration. It examines layered architecture, modularisation, state management, security, performance and maintenance using a prepared application.

Participants complete selected refactoring, security-review, profiling and testing exercises; the remaining topics use short demonstrations and design discussions. One UI approach is used for hands-on work, with XML/Views and Compose compared where useful. The course does not promise bulletproof protection, production readiness or measurable performance gains on every device.

Learning outcomes

The course teaches participants to:

- Review responsibilities, module dependencies and boundaries in a small layered Android project.
- Use screen-level state holders and lifecycle-aware coroutine cancellation appropriately.
- Distinguish UI-related work from persistent background operations.
- Evaluate communication, permission, storage, authentication and logging risks.
- Recognise deprecated storage APIs and the limitations of obfuscation and device-integrity checks.
- Use profiling evidence to investigate selected memory, rendering or startup issues.
- Add selected tests and identify CI, compatibility and maintenance considerations.

Prerequisites

- Working experience with Kotlin (self-taught or otherwise), including coroutines.
- Professional Android development experience: activities/fragments or composables, views, basics of lifecycle.
- Familiarity with REST APIs and JSON.

A prepared compatible Android Studio, JDK/Gradle and dependency set, plus a supported emulator or training device and permission to install the supplied application.

Training outline

1. Day 1 – Architecture and State

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

- Consistent project/package organisation suitable to the application's scale.
- 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

- Compare XML/Views and Jetpack Compose; use one prepared UI stack for exercises.

ViewModel Best Practices

- Scope ViewModels appropriately and avoid retaining Views or lifecycle-bound references.
- Using viewModelScope, structured concurrency; cancellation.
- SavedStateHandle and lifecycle-aware state handling; distinguish saved UI state from durable application data.
- UI-related data fetching and state preparation belong in suitable screen-level state holders; persistent background work requires an appropriate background-work API rather than a ViewModel lifetime assumption.

Guided Architecture Exercise

Trace a feature's state and data flow, then refactor a selected boundary or state holder in the prepared application.

2. Day 2 — Application Security

Fundamental security guidelines (Android best practices)

- HTTPS/TLS and certificate validation; examine pinning's operational risks instead of treating it as a default recommendation.
- Limiting permissions; runtime permissions; minimal permissions principle
- Sensitive-data storage, Android Keystore-backed key management and appropriate maintained cryptographic APIs. Recognise deprecated EncryptedSharedPreferences in existing code; plain preferences are not an equivalent encrypted store. Discuss database/file encryption choices, including SQLCipher where relevant.

Obfuscation / Code protection

- R8 shrinking and obfuscation in current Android builds; recognise older ProGuard configuration terminology. Obfuscation is not a complete security boundary.
- Avoiding metadata exposure; handling Kotlin metadata issues.

Rooting, device-integrity and tampering indicators: bypass and false-result limitations; Android terminology rather than an iOS jailbreak promise.

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

Review one prepared communication/storage/authentication flow using fictional data; compare risks and document limitations.

3. Day 3 – Profiling, Testing and Maintenance

Performance Investigation

Kotlin language and code level optimizations

- Immutability, null handling and allocation choices; measure before applying speculative optimisations.
- Coroutine best practices: dispatchers (Main vs IO vs Default), structured concurrency, avoiding blocking operations on main thread
- Rendering/recomposition examples for the selected UI stack; use keys, remembered state and derived state where appropriate, not as universal tuning rules.
- 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 optimisation and application packaging options; size/distribution benefits depend on configuration and platform.

Threading and concurrency

- Async task patterns; avoiding race conditions; proper error propagation

Startup performance, cold & warm starts

Quality and Maintenance

Testing strategies

- Unit tests for ViewModels, Use Cases, data repositories
- Integration tests (e.g. API + parsing)
- UI tests (Espresso or Compose UI tests)

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 relevant Android Studio profiling tools and inspect available memory, CPU, rendering and network evidence; tool capability varies by environment.
- Crash reporting, analytics, handling user error reports

Versioning and backward/forward compatibility

Handling different device configurations: foldables, various densities, locales

Profile one selected bottleneck, compare before/after measurements and add an appropriate test; use demonstrations for the remaining tool and CI topics.

