Faysal Aziz

Modern Kotlin Engineering

From Android Applications to MVP Architecture, and Scalable Backend Systems — 3 Days

FA-0324 · 3 Days

Software Development · DevOps, Cloud & Infrastructure · Cybersecurity

---

Introduction

Kotlin has evolved far beyond its early reputation as a cleaner alternative to Java for Android development. Modern Kotlin developers are now expected to understand how applications interact with APIs, backend systems, cloud infrastructure, AI-driven services, real-time communication platforms, security controls, and modern product requirements. Mobile applications today operate as part of much larger digital ecosystems, requiring developers to think beyond UI implementation and feature delivery alone.

This course is designed to help Kotlin and Android developers move from feature-centric coding into modern product-focused engineering. Participants will learn how to structure maintainable Kotlin applications, implement MVP (Model-View-Presenter) architecture properly, improve scalability and performance, secure applications correctly, integrate AI capabilities, and understand how Android systems connect with modern backend and distributed platforms.

A major focus of this course is also the concept of the MVP, or Minimum Viable Product. Participants will learn how to identify the smallest valuable version of a product, prioritize features correctly, avoid unnecessary engineering complexity, validate assumptions early, and evolve applications toward scalable production systems.

The course is delivered by an instructor with over 30 years of industry experience using real industry-demanded content rather than overly academic material. The training reflects current industry expectations surrounding Kotlin development, Android platform evolution, modern application architecture, cloud-connected systems, AI-enabled applications, and scalable software engineering practices.

Learning outcomes

By the end of this course, participants will be able to:

- Understand Kotlin’s role in modern Android and backend development
- Design cleaner and more maintainable Kotlin applications
- Apply MVP (Model-View-Presenter) architecture in Android applications
- Structure scalable Android applications using proper separation of concerns
- Plan and structure a Minimum Viable Product effectively
- Identify core features, supporting features, and future enhancements
- Improve application security using authentication, authorization, and secure coding practices
- Optimize application performance, memory usage, threading, and application size
- Prepare Android applications for modern Android platform changes
- Integrate AI capabilities using API-driven and on-device approaches
- Understand how Kotlin applications connect with backend and distributed systems
- Understand event-driven systems and real-time communication concepts
- Build applications with improved maintainability, scalability, and production readiness

Prerequisites

- Basic knowledge of Kotlin programming
- Basic familiarity with Android application development
- Understanding of object-oriented programming concepts
- Basic knowledge of REST APIs and networking
- Familiarity with Git or version control systems
- Optional but helpful: basic Java knowledge
- Optional but helpful: exposure to backend development concepts

Training outline

1. Modern Kotlin Development Overview

Kotlin’s role in modern software development

1. Evolution of Kotlin in enterprise environments
2. Kotlin adoption trends
3. Kotlin ecosystem overview
4. Modern Android development expectations

Kotlin for Android Applications

1. Native Android development
2. Kotlin language advantages
3. Android ecosystem overview
4. Android application lifecycle awareness

Kotlin Beyond Mobile Development

1. Backend development with Kotlin
2. Kotlin multiplatform overview
3. Cloud-connected application development
4. Shared business logic concepts

Product-Oriented Engineering

1. Moving beyond feature-centric development
2. Understanding product-focused engineering
3. Aligning development with business goals
4. Building maintainable long-term systems

2. Minimum Viable Product Thinking for Kotlin Applications

Understanding Minimum Viable Product Concepts

1. Meaning of MVP
2. Difference between prototype and MVP
3. Difference between MVP and production systems
4. Importance of early validation

Product Planning Fundamentals

1. Identifying business problems
2. Defining target users
3. Understanding product requirements
4. Prioritizing business value

Feature Prioritization

1. Identifying core features
2. Separating must-have and optional features
3. Avoiding feature overload
4. Technical prioritization strategies

MVP Delivery Strategies

1. Avoiding overengineering
2. Balancing speed and maintainability
3. Technical decisions for rapid delivery
4. Planning future scalability

Product Evolution

1. User feedback collection
2. Iterative improvement planning
3. Technical debt awareness
4. Transitioning from MVP to scalable products

3. Kotlin Project Structure and Code Organization

Standard Kotlin Project Structure

1. Package organization
2. File organization strategies
3. Naming conventions
4. Source set management

Organizing Application Layers

1. UI layer organization
2. Business logic organization
3. Data layer organization
4. Shared utility management

Gradle and Build Management

1. Gradle project structure
2. Dependency management
3. Build configuration handling
4. Environment-based configuration

Modular Application Design

1. Introduction to modularization
2. Feature module concepts
3. Benefits of modular architecture
4. Scalability considerations

4. Application Architecture Fundamentals

Importance of Application Architecture

1. Maintainability concerns
2. Scalability concerns
3. Common architectural failures
4. Long-term project sustainability

Separation of Concerns

1. Decoupling application layers
2. Reducing tight coupling
3. Improving code maintainability
4. Improving testability

Clean Architecture Concepts

1. Presentation layer
2. Domain layer
3. Data layer
4. Dependency inversion principles

Architectural Design Patterns

1. Repository pattern
2. Use case pattern
3. Interface-driven development
4. Dependency management concepts

5. MVP (Model-View-Presenter) Architecture Pattern

Introduction to MVP Architecture

1. Purpose of MVP architecture
2. Benefits of MVP in Android development
3. MVP compared with alternative architectures
4. MVP suitability for scalable applications

Core MVP Components

1. Model responsibilities
2. View responsibilities
3. Presenter responsibilities
4. Communication flow in MVP

View and Presenter Interaction

1. View contracts and interfaces
2. User action handling
3. Presenter-driven UI updates
4. Managing screen state

Business Logic Separation

1. Separating UI and business logic
2. Repository integration with MVP
3. Navigation handling strategies
4. Error and success state handling

MVP Best Practices

1. Avoiding tightly coupled components
2. Lifecycle-aware Presenter handling
3. Memory management considerations
4. Common MVP implementation mistakes

MVP for Enterprise Applications

1. Scalable MVP architecture
2. Structuring large Android applications
3. Team collaboration considerations
4. MVP architecture in Minimum Viable Product development

6. Presenter Design and Best Practices

Presenter Responsibilities

1. Managing application logic
2. Coordinating View interactions
3. Handling business workflows
4. Presenter responsibility boundaries

Presenter Lifecycle Handling

1. Lifecycle-aware operations
2. Preventing memory leaks
3. Managing references safely
4. Presenter cleanup strategies

Asynchronous Operations in Presenters

1. Coroutines within Presenters
2. Background task handling
3. Error handling strategies
4. Retry mechanisms

Presenter Testing

1. Unit testing strategies
2. Mocking dependencies
3. Testing View interactions
4. Testing business logic independently

Structuring Presenters

1. Presenters for simple screens
2. Presenters for complex workflows
3. Reusable Presenter strategies
4. Organizing Presenter dependencies

7. Reactive and Asynchronous Programming

Modern Asynchronous Programming Concepts

1. Need for asynchronous execution
2. Concurrency concerns
3. UI responsiveness considerations
4. Safe background processing

Kotlin Coroutines

1. Suspend functions
2. Coroutine scopes
3. Dispatchers
4. Structured concurrency

Reactive Data Streams

1. Flow fundamentals
2. StateFlow fundamentals
3. SharedFlow overview
4. Reactive UI update patterns

Android Reactive Patterns

1. LiveData overview
2. LiveData versus Flow
3. Avoiding UI thread blocking
4. Handling asynchronous UI updates

8. Security Fundamentals for Kotlin Applications

Application Security Principles

1. Secure coding practices
2. Common Android security risks
3. Security-first development mindset
4. Application attack surfaces

Authentication and Authorization

1. Authentication concepts
2. Authorization concepts
3. Token-based authentication
4. OAuth overview
5. JWT concepts

Secure Data Handling

1. Secure local storage
2. Android Keystore concepts
3. Encryption basics
4. Key management concepts

Network Security

1. HTTPS enforcement
2. Certificate pinning overview
3. API communication security
4. Secure session handling

Production Security Considerations

1. Obfuscation and code shrinking
2. Secure release preparation
3. Security during MVP releases
4. Security testing awareness

9. Performance Tuning and Optimization

Performance Fundamentals

1. User experience impact
2. CPU performance concerns
3. Memory performance concerns
4. Network performance considerations

Memory Management

1. Garbage collection overview
2. Detecting memory leaks
3. Efficient resource management
4. Memory optimization techniques

Application Optimization

1. Reducing application size
2. Resource shrinking
3. Code minification
4. Startup optimization

UI Performance Optimization

1. Smooth rendering concepts
2. RecyclerView optimization
3. Efficient UI updates
4. Avoiding unnecessary rendering

Monitoring and Profiling

1. Profiling tools overview
2. Performance monitoring
3. Production diagnostics
4. Performance trade-offs during MVP development

10. Android Platform Evolution and Compatibility

Android Platform Changes

1. Android 12 through Android 16 overview
2. Platform evolution trends
3. Modern Android expectations
4. SDK targeting considerations

Security and Privacy Changes

1. Runtime permission updates
2. Privacy-related restrictions
3. Background execution limitations
4. Security-related behavior changes

Device Compatibility

1. Large-screen support
2. Foldable device awareness
3. Multi-device compatibility
4. Backward compatibility planning

Modernization Strategies

1. Refactoring legacy Android applications
2. Compatibility testing
3. Multi-version testing strategies
4. Release readiness planning

11. AI Integration in Kotlin and Android Applications

AI in Modern Applications

1. AI as a product feature
2. AI-driven user experiences
3. AI opportunities in Android applications
4. AI limitations and constraints

AI Integration Strategies

1. API-based AI integration
2. Cloud AI services
3. On-device AI concepts
4. AI request and response handling

AI Product Planning

1. Selecting AI features for MVP releases
2. Responsible AI usage
3. User consent considerations
4. AI privacy concerns

AI Operational Considerations

1. Latency concerns
2. Reliability concerns
3. AI feature testing
4. Planning future AI capabilities

12. Backend Services for Kotlin Developers

Backend Awareness for Mobile Developers

1. Client-server application models
2. Role of backend systems
3. Mobile-backend communication patterns

4. Backend scalability awareness

REST API Concepts

1. Request and response handling
2. API integration strategies
3. API error handling
4. API validation awareness

Backend Security

1. Authentication with backend systems
2. Authorization handling
3. Secure API communication
4. Session and token handling

Kotlin for Backend Development

1. Kotlin backend ecosystem overview
2. Java interoperability
3. Backend framework awareness
4. Shared business logic opportunities

13. Microservices and Distributed Systems Basics

Distributed System Fundamentals

1. Monolithic architecture concepts
2. Microservices architecture concepts
3. Service decomposition strategies
4. Distributed system trade-offs

Service Communication

1. REST communication
2. gRPC overview
3. API gateway concepts
4. Service-to-service communication

Reliability and Scalability

1. Fault tolerance basics
2. Retry strategies
3. Circuit breaker concepts
4. Horizontal scaling basics

Cloud-Native Considerations

1. Cloud deployment overview
2. Containerization awareness
3. Scalability planning
4. MVP-first architecture thinking

14. Event Streaming and Real-Time Systems

Event-Driven Architecture

1. Request-response versus event-driven communication
2. Real-time processing concepts
3. Event-driven system benefits
4. Event-driven trade-offs

Apache Kafka Fundamentals

1. Topics and partitions
2. Producers and consumers
3. Event streaming concepts
4. Message ordering considerations

Real-Time Mobile Integration

1. Mobile real-time updates
2. Notification systems
3. Backend event integration
4. Real-time synchronization concepts

Operational Considerations

1. Data consistency awareness
2. Reliability considerations
3. Event-driven scalability
4. Event streaming suitability for MVP systems

15. Connecting Mobile Applications with Cloud Systems

Cloud-Connected Applications

1. Mobile applications as cloud-connected clients
2. Cloud communication patterns
3. Cloud service integration
4. Backend synchronization awareness

Offline and Synchronization Concepts

1. Offline-first strategies
2. Data synchronization basics
3. Conflict resolution awareness
4. Data consistency considerations

Monitoring and Observability

1. Logging strategies
2. Metrics and monitoring
3. Tracing basics
4. Crash reporting concepts

Product Improvement Cycles

1. Release monitoring
2. User behavior analysis
3. Product feedback loops
4. Improving products after MVP launch

16. Special Note

Some topics, particularly those involving AI services, backend systems, cloud platforms, package repositories, event streaming platforms, and external APIs, may require internet connectivity and firewall exceptions during training depending on the tools, services, and environments selected by the instructor.

Disclaimer

This course outline is intended solely as a general training guideline. The trainer reserves the right to amend, reorganize, expand, reduce, substitute, resequence, or otherwise modify any topic, sub-topic, tool, technology, sequence, lab activity, or delivery approach as deemed necessary based on participant background, class progress, technical constraints, environmental limitations, industry developments, or evolving training objectives without prior notice.

© 2026 Faysal Aziz · Kuala Lumpur