

Nouveauté

Ref. KOTL-AV

Android – Développement avancé en Kotlin (performances, capteurs et tests)

3 jours (21 heures)

Perfectionnement

Présentiel ou classe à distance

à partir de

1905 € HT par personne (TVA en vigueur)

 Repas inclus en présentiel

Contenu mis à jour le 25/09/2026. Document téléchargé le 02/10/2026.

Objectifs

A l'issue de cette formation, vous serez capable de :

- Identifier les architectures avancées Android (Clean Architecture, MVI, modularisation)
- Optimiser les performances applicatives : rendu, mémoire, réseau, démarrage, batterie
- Intégrer et exploiter les capteurs matériels dans un contexte industriel (vibration, température, géolocalisation...)
- Mettre en place une stratégie de tests robuste : unitaires, snapshot, UI, end-to-end
- Utiliser les outils d'IA Générative (Copilot, Gemini) comme assistants transversaux tout au long du développement.

Les compétences visées par cette formation font l'objet d'une évaluation.

À qui s'adresse cette formation ?

Développeurs Android confirmés, Tech Leads, architectes.

Prérequis

Avoir suivi la formation KOTL-AND "Android – Développement natif en Kotlin" ou maîtriser les fondamentaux du développement Android.

Programme

1 Pendant votre formation

Android – Développement avancé en Kotlin (performances, capteurs et tests)

Jour 1 – Matin

Clean Architecture et modularisation

- Limites de MVVM (Model View ViewModel) seul : couplage fort, testabilité réduite, scalabilité
- Clean Architecture : couches Domain / Data / Presentation – responsabilités
- Use Cases / Interactors : rôle, contrat Kotlin, flux de données
- Repository pattern : abstraction, sources multiples, cache
- Hilt avancé : scopes (Singleton, ActivityRetained, ViewModel), multibinding, qualifieurs
- Modularisation du projet : feature modules, :core:network, :core:ui, :app
- Gestion des dépendances inter-modules : API/impl pattern, visibilité Gradle
- Navigation multi-module avec Jetpack Navigation Component
- Utiliser Copilot pour générer le squelette Clean Architecture (interfaces, Data classes, modules Hilt)
- Gemini Code Assist : expliquer les dépendances Gradle générées et proposer des optimisations

Exemples de travaux pratiques (à titre indicatif)

- *Fil Rouge – Etape 1 : Création du projet IndustrialSense*
 - Créer le projet multi-module : :app, :feature:dashboard, :core:network, :core:sensors
 - Mettre en place la Clean Architecture pour le module :core:network (Retrofit + Repository)
 - Configurer Hilt avec les scopes appropriés pour l'injection cross-modules
 - Générer les interfaces et implémentations initiales avec GitHub Copilot, puis les adapter

Jour 1 – Après-midi

Pattern MVI (Model View Intent) et recompositions Compose

- Différences MVVM vs MVI : Intent, State, Effect – quand choisir MVI
- Implémentation MVI : StateFlow pour le state, SharedFlow pour les effets
- Gestion des side-effects (navigation, snackbars, dialogs) via ChannelUnidirectional Data Flow (UDF) et son lien avec Compose
- Optimisation des recompositions : remember, derivedStateOf, key, lambda stability
- Stabilité des types Compose : @Stable, @Immutable, Compose compiler report
- Profiling des recompositions avec Layout Inspector et Composition tracing
- Générer le boilerplate MVI (State/Intent/Effect) avec Copilot et analyser la qualité
- Utiliser Gemini pour détecter les types instables dans le rapport Compose compiler

Exemples de travaux pratiques (à titre indicatif)

- *Fil Rouge – Etape 2 : Ecran Dashboard en MVI*

- Implémenter le *DashboardViewModel* en MVI (état Loading / Success / Error, effets de navigation)
- Créer le *DashboardScreen* Compose consommant le *StateFlow* – optimiser les recompositions
- Détecter et corriger au moins 2 recompositions inutiles via le *Layout Inspector*
- Générer les tests unitaires du *ViewModel* avec Copilot (cas nominal + erreur réseau)

Jour 2 – Matin

Profiling et performance Android

- Android Profiler : CPU (Method trace, Callstack), Memory, Network, Energy
- Perfetto et Systrace : analyse frame-by-frame, jank, thread scheduling
- Optimisation du rendu : overdraw, GPU rendering, layers Compose
- Gestion mémoire : LeakCanary, allocation patterns, GC pressure, Bitmap pool
- Optimisation réseau : caching OkHttp / HTTP, pagination (Paging 3), compression
- App Startup et cold start : ContentProvider, Initializer, Baseline Profiles
- WorkManager avancé : contraintes, chaînage, périodicité, observation via Flow
- Optimisation batterie : Doze, App Standby buckets, background restrictions
- Copilot : générer la configuration Baseline Profile et le script de warmup Macrobenchmark
- Gemini : analyser une trace Perfetto et suggérer des corrections ciblées

Exemples de travaux pratiques (à titre indicatif)

- *Fil Rouge – Etape 3 : Performance du dashboard*
 - Identifier et corriger une fuite mémoire simulée dans le module `:core:network` avec LeakCanary
 - Mesurer l'impact d'un Baseline Profile sur le cold start (avant / après avec Macrobenchmark)
 - Optimiser le chargement de la liste d'équipements avec Paging 3 et cache OkHttp
 - Utiliser Gemini pour analyser la trace Perfetto et documenter les gains obtenus

Jour 2 – Après-midi

SensorManager et capteurs de mouvement / environnementaux

- Architecture SensorManager : Sensor, SensorEvent, SensorEventListener, batch
- Taxonomie des capteurs Android : hardware vs software (virtual / composite)
- Capteurs de mouvement : accéléromètre, gyroscope, vecteur de rotation
- Capteurs environnementaux : température ambiante, pression barométrique, humidité
- Capteur de luminosité (lux) : usage pour l'ergonomie des IHM industrielles
- Fréquence d'échantillonnage : SENSOR_DELAY_NORMAL/UI/GAME/FASTEST – impact batterie
- Batching et FIFO : report latency, lecture différée, économie d'énergie
- Intégration avec Kotlin Flow : SensorEventListener > callbackFlow > StateFlow
- Filtrage du signal : filtre passe-bas, filtre complémentaire, lissage exponentiel
- Détection de vibrations anormales : seuillage, RMS, FFT simplifiée
- Gemini : générer le wrapper callbackFlow pour un capteur donné et expliquer le cycle de vie
- Copilot : générer l'algorithme de détection d'anomalie vibratoire à partir d'une description

Exemples de travaux pratiques (à titre indicatif)

- *Fil Rouge – Etape 4 : Module capteurs de mouvement et environnementaux*
 - Créer le module `:core:sensors` exposant un Flow pour accéléromètre et capteur de pression
 - Implémenter la détection de vibration anormale (seuil RMS configurable) pour simuler une alerte machine
 - Afficher en temps réel les données de pression et température dans un composant Compose dédié
 - Générer avec Copilot le filtre passe-bas et valider son comportement avec un test unitaire (données simulées)

Jour 3 – Matin

Capteurs de position, fusion et applications industrielles

- Capteurs de position : magnétomètre, boussole, orientation 3D
- Géolocalisation : GPS, FusedLocationProvider (Google Play / alternatives Open Source)
- Fusion de capteurs : calcul d'orientation par accéléromètre + magnétomètre
- Algorithmes de fusion avancés : filtre de Kalman simplifié, filtre complémentaire
- Précision et latence : compromis selon le cas d'usage (navigation, tracking d'actifs)
- Suivi d'actifs industriels mobiles : chariots élévateurs, opérateurs, équipements de terrain
- Geofencing et zones de sécurité : déclenchement d'alertes à l'entrée / sortie de périmètres
- Intégration avec des capteurs externes BLE / MQTT (IoT industriel) : architecture client Android
- Permissions et bonnes pratiques : ACCESS_FINE_LOCATION, rationale, fallback, RGPD
- Gestion du cycle de vie : arrêt / reprise des capteurs selon foreground / background / service
- Cas d'usage secteur industrie : maintenance prédictive, rondes de surveillance, AGV
- Copilot : générer le code de geofencing (ajout / suppression de zones) à partir d'une description fonctionnelle
- Gemini : proposer une architecture BLE > Android > MQTT broker pour un contexte Industrie 4.0

Exemples de travaux pratiques (à titre indicatif)

- *Fil Rouge – Etape 5 : Localisation, Geofencing et alertes terrain*
 - Intégrer FusedLocationProvider dans `:core:sensors` et afficher la position de l'équipement sur le dashboard
 - Implémenter 2 zones de geofencing (zone de travail autorisée / zone de danger) avec alertes visuelles Compose
 - Simuler la réception de données d'un capteur BLE externe via une interface mock + Flow
 - Utiliser Gemini pour générer la documentation du module `:core:sensors` (KDoc) et valider les cas limites

Jour 3 – Après-midi

Stratégie de tests avancés

- Pyramide de tests Android : équilibre unit / intégration / UI / E2E
- Tests unitaires avancés : MockK (spy, relaxed, slot), coroutines-test, Turbine pour Flow
- Tests de ViewModel MVI : vérification des states, intents et effets
- Tests d'intégration : Hilt testing, Room in-memory, Fake repositories
- Snapshot testing avec Paparazzi : tests de régression visuelle sur composants Compose
- Tests UI Compose : sémantique, actions, assertions, test tags

- Tests End-to-End avec Espresso + UIAutomator : scénarios complets
- Test Driven Development (TDD) appliqué : cycle Red / Green / Refactor
- Couverture de code : JaCoCo, configuration des seuils, rapport HTML, intégration CI/CD
- Copilot : générer les cas de test edge-case à partir de la signature d'une fonction et valider leur pertinence
- Gemini : proposer des scénarios de test pour le module capteurs (données aberrantes, permission refusée...)

Exemples de travaux pratiques (à titre indicatif)

- *Ecrire les tests Turbine du SensorViewModel (flux normal, déconnexion capteur, dépassement de seuil)*
- *Mettre en place des snapshot tests Paparazzi sur les composants Dashboard et SensorCard*
- *Créer un test d'intégration Hilt du module :core:sensors avec un FakeSensorManager*
- *Configurer le rapport JaCoCo avec seuil 75% et exécuter la suite complète – revue de code finale assistée IA*

Financement

Plusieurs dispositifs de financement sont accessibles via les OPCO (Opérateurs de Compétences), organismes agréés par le ministère du Travail dont le rôle est d'accompagner, collecter et gérer les contributions des entreprises au titre du financement de la formation professionnelle.

Pour plus d'information, une équipe de gestionnaires SKOLAE Formation spécialisée vous accompagne dans le choix de vos formations et la gestion administrative.

[Nous contacter pour en savoir plus →](#)

Modalités pédagogiques

Formation délivrée en présentiel ou distanciel* (blended-learning, e-learning, classe virtuelle, présentiel à distance).

Le formateur alterne entre méthode** démonstrative, interrogative et active (via des travaux pratiques et/ou des mises en situation).

Variables suivant les formations, les moyens pédagogiques mis en oeuvre sont :

- Ordinateurs Mac ou PC (sauf pour certains cours de l'offre Management), connexion internet fibre, tableau blanc ou paperboard, vidéoprojecteur ou écran tactile interactif (pour le distanciel)
- Environnements de formation installés sur les postes de travail ou en ligne
- Supports de cours et exercices

En cas de formation intra sur site externe à M2i, le client s'assure et s'engage également à avoir toutes les ressources matérielles pédagogiques nécessaires (équipements informatiques...) au bon déroulement de l'action de formation visée conformément aux prérequis indiqués dans le programme de formation communiqué.

* nous consulter pour la faisabilité en distanciel

** ratio variable selon le cours suivi

Accessibilité aux personnes en situation de handicap

Détails de la session

Du 12 octobre 2027 au 14 octobre 2027 (3 jours / 21 heures)

Paris La Défense

Du 12 octobre 2027 au 14 octobre 2027

Paris La Défense

- 12 octobre 2027 9h00 – 16h00
- 13 octobre 2027 9h00 – 16h00
- 14 octobre 2027 9h00 – 16h00