Notas Técnicas para Desarrolladores

Migración Kotlin 2.0 + AGP 9 + KSP
Fecha: 2026-01-23

🔧 Cambios en Tooling

Gradle Configuration Cache

Estado: Ahora **ACTIVADO** (antes deshabilitado)

```
# gradle.properties
org.gradle.configuration-cache=true
org.gradle.configuration-cache.problems=warn
```

Impacto:

- ⚡ Builds ~50% más rápidos cuando cache hit
- ⚠️ Puede mostrar warnings en tareas que no son cache-friendly
- 🔧 Si hay problemas persistentes, desactivar temporalmente

Comandos útiles:

```
# Invalidar configuration cache
./gradlew --stop
rm -rf .gradle/configuration-cache

# Ver problemas de cache
./gradlew :app:assembleDebug --configuration-cache-problems=warn
```

KSP vs KAPT

Cambio: Toda la generación de código migrada de KAPT → KSP

Aspecto	KAPT (antes)	KSP (ahora)
Path generado	`build/generated/source/kapt/`	`build/generated/ksp/`
Velocidad	Baseline	30-40% más rápido
Incremental	Limitado	Mejor soporte

Aspecto	KAPT (antes)	KSP (ahora)
Compatibilidad Kotlin 2	Degradada	Nativa

Implicaciones para desarrollo:

1. **Clean builds más importantes:**
   - Si ves errores extraños de clases no encontradas, hacer clean
   - KSP es incremental pero a veces requiere regeneración completa
2. **IDE indexing:**
   - Primera vez después de clean puede tardar más
   - Android Studio debe indexar nuevos paths
3. **Debugging:**
   - Código generado está en nuevo path
   - `Find Usages` en clases anotadas puede tardar más la primera vez

Verificar generación de código:

```
# Hilt components
find app/build/generated/ksp -name "*Hilt*"

# Room implementations
find app/build/generated/ksp -name "*_Impl.kt"

# Ver todo lo generado
ls -R app/build/generated/ksp/debug/kotlin/
```

🎨 Compose Compiler

Antes (Kotlin 1.9.x)

```
// app/build.gradle.kts
composeOptions {
    kotlinCompilerExtensionVersion = "1.5.14"
}
```

Ahora (Kotlin 2.0.x)

```
// build.gradle.kts (root)
plugins {
```

```
    id("org.jetbrains.kotlin.plugin.compose") version "2.0.21" apply false
}

// app/build.gradle.kts
plugins {
    id("org.jetbrains.kotlin.plugin.compose")
}

// composeOptions ya NO es necesario
```

Ventajas:

- ✅ Automático: plugin selecciona versión compatible
- ✅ Sin preocuparse por compatibility matrix
- ✅ Mejor optimización con Kotlin 2.0

Compose Compiler Reports (opcional):

Para ver métricas de Compose:

```
// app/build.gradle.kts
android {
    kotlinOptions {
        freeCompilerArgs += listOf(
            "-P",
            "plugin:androidx.compose.compiler.plugins.kotlin:reportsDestination=" +
                "${layout.buildDirectory.get()}/compose_metrics",
            "-P",
            "plugin:androidx.compose.compiler.plugins.kotlin:metricsDestination=" +
                "${layout.buildDirectory.get()}/compose_metrics"
        )
    }
}
```

Ver reportes:

```
./gradlew :app:assembleDebug
open app/build/compose_metrics/
```

📦 Dependencias: Breaking Changes

Retrofit 2.9 → 2.11

Cambios menores, mayormente compatibles

- Kotlin Coroutines support mejorado

- Mejor manejo de `null` en responses
- Serialization plugins actualizados

Verificar:

```
// Si usas custom converters, verificar firma
interface CustomConverter : Converter.Factory {
    // API debería ser igual, pero revisar tipos
}
```

Coroutines 1.7.3 → 1.9.0

Cambios importantes:

1. **`Flow.collect` más estricto:**

```
// Antes: podía no cancelar correctamente
flow.collect { value →
    // ...
}

// Ahora: cancelación más predecible
// (código existente debería funcionar, pero tests pueden comportarse diferente)
```

2. **`TestDispatcher` API cambió:**

```
// Antes (posiblemente en tus tests)
@Before
fun setup() {
    Dispatchers.setMain(TestCoroutineDispatcher())
}

// Ahora (recomendado)
@Before
fun setup() {
    Dispatchers.setMain(StandardTestDispatcher())
}
```

3. **`runTest` más robusto:**

```
@Test
fun myTest() = runTest {
    // Mejor manejo de tiempo virtual
    // Puede detectar leaks que antes no detectaba
}
```

Compose BOM 2024.02 → 2024.12

Nuevos componentes disponibles:

```
// DatePicker moderno
import androidx.compose.material3.DatePicker
import androidx.compose.material3.rememberDatePickerState

@Composable
fun MyDatePicker() {
    val state = rememberDatePickerState()
    DatePicker(state = state)
}

// TimeInput
import androidx.compose.material3.TimeInput
import androidx.compose.material3.rememberTimePickerState

// ModalBottomSheet mejorado
import androidx.compose.material3.ModalBottomSheet
```

Breaking changes menores:

- `BottomSheetScaffold` tiene nuevos parámetros opcionales
- `Snackbar` durations más consistentes
- `TextField` outline rendering mejorado

🔒 Seguridad: Keystore vs security-crypto

Patrón Recomendado

Para SharedPreferences cifradas:

```
// KeystoreManager.kt
object KeystoreManager {
    private const val KEY_ALIAS = "app_master_key"
    private const val KEYSTORE = "AndroidKeyStore"

    fun getOrCreateKey(): SecretKey {
        val keyStore = KeyStore.getInstance(KEYSTORE).apply { load(null) }

        return if (keyStore.containsAlias(KEY_ALIAS)) {
            keyStore.getKey(KEY_ALIAS, null) as SecretKey
        } else {
            KeyGenerator.getInstance(KeyProperties.KEY_ALGORITHM_AES, KEYSTORE)
                .apply {
                    init(KeyGenParameterSpec.Builder(
                        KEY_ALIAS,
                        KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYP
                    )
```

```
                        .setBlockModes(KeyProperties.BLOCK_MODE_GCM)
                        .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
                        .setKeySize(256)
                        .setUserAuthenticationRequired(false) // O true si quieres bi
                        .build()
                    )
                }
                .generateKey()
        }
    }
}

// CryptoHelper.kt
object CryptoHelper {
    private const val TRANSFORMATION = "AES/GCM/NoPadding"

    fun encrypt(plaintext: String): EncryptedData {
        val cipher = Cipher.getInstance(TRANSFORMATION)
        cipher.init(Cipher.ENCRYPT_MODE, KeystoreManager.getOrCreateKey())

        val ciphertext = cipher.doFinal(plaintext.toByteArray(Charsets.UTF_8))
        val iv = cipher.iv

        return EncryptedData(
            ciphertext = Base64.encodeToString(ciphertext, Base64.NO_WRAP),
            iv = Base64.encodeToString(iv, Base64.NO_WRAP)
        )
    }

    fun decrypt(encrypted: EncryptedData): String {
        val cipher = Cipher.getInstance(TRANSFORMATION)
        val iv = Base64.decode(encrypted.iv, Base64.NO_WRAP)
        val ciphertext = Base64.decode(encrypted.ciphertext, Base64.NO_WRAP)

        cipher.init(
            Cipher.DECRYPT_MODE,
            KeystoreManager.getOrCreateKey(),
            GCMParameterSpec(128, iv)
        )

        val plaintext = cipher.doFinal(ciphertext)
        return String(plaintext, Charsets.UTF_8)
    }
}

data class EncryptedData(val ciphertext: String, val iv: String)

// SecureStorage.kt (con DataStore)
class SecureStorage(context: Context) {
    private val dataStore = context.dataStore

    suspend fun saveSecure(key: String, value: String) {
```

```
        val encrypted = CryptoHelper.encrypt(value)
        dataStore.edit { prefs →
            prefs[stringPreferencesKey("${key}_data")] = encrypted.ciphertext
            prefs[stringPreferencesKey("${key}_iv")] = encrypted.iv
        }
    }

    suspend fun getSecure(key: String): String? {
        val prefs = dataStore.data.first()
        val ciphertext = prefs[stringPreferencesKey("${key}_data")] ?: return null
        val iv = prefs[stringPreferencesKey("${key}_iv")] ?: return null

        return try {
            CryptoHelper.decrypt(EncryptedData(ciphertext, iv))
        } catch (e: Exception) {
            Log.e("SecureStorage", "Decryption failed", e)
            null
        }
    }
}
```

Testing del código de cifrado:

```
@Test
fun `encrypt and decrypt returns original value`() = runTest {
    val original = "sensitive_token_123"
    val encrypted = CryptoHelper.encrypt(original)
    val decrypted = CryptoHelper.decrypt(encrypted)

    assertEquals(original, decrypted)
    assertNotEquals(original, encrypted.ciphertext)
}

@Test
fun `different encryptions of same value produce different ciphertexts`() = runTest {
    val value = "test"
    val encrypted1 = CryptoHelper.encrypt(value)
    val encrypted2 = CryptoHelper.encrypt(value)

    // IVs diferentes = ciphertexts diferentes (seguridad)
    assertNotEquals(encrypted1.ciphertext, encrypted2.ciphertext)
    assertNotEquals(encrypted1.iv, encrypted2.iv)

    // Pero ambos decryptan al original
    assertEquals(value, CryptoHelper.decrypt(encrypted1))
    assertEquals(value, CryptoHelper.decrypt(encrypted2))
}
```

🧪 Testing con Kotlin 2.0

TestDispatcher Changes

```
// MainDispatcherRule.kt (actualizado)
class MainDispatcherRule(
    private val dispatcher: TestDispatcher = StandardTestDispatcher()
) : TestWatcher() {
    override fun starting(description: Description) {
        Dispatchers.setMain(dispatcher)
    }

    override fun finished(description: Description) {
        Dispatchers.resetMain()
    }
}

// Uso en tests
class ViewModelTest {
    @get:Rule
    val mainDispatcherRule = MainDispatcherRule()

    @Test
    fun `test with coroutines`() = runTest {
        // Usar advanceUntilIdle() más explícitamente
        viewModel.loadData()
        advanceUntilIdle()

        // Assertions
        assertEquals(expected, viewModel.state.value)
    }
}
```

Room Testing

```
// Con KSP, asegurar que esquema se genera
android {
    defaultConfig {
        javaCompileOptions {
            annotationProcessorOptions {
                arguments["room.schemaLocation"] = "$projectDir/schemas"
            }
        }
    }

    // Con KSP:
    ksp {
        arg("room.schemaLocation", "$projectDir/schemas")
    }
}
```

🚀 Performance Tips

1. Usar Build Analyzer

```
# Build con análisis
./gradlew :app:assembleDebug --profile --scan

# En Android Studio:
# View → Tool Windows → Build Analyzer
```

2. Parallel Execution

```
# gradle.properties (ya configurado)
org.gradle.parallel=true
org.gradle.workers.max=8  # Ajustar según CPU
```

3. Incremental Compilation

```
# gradle.properties (ya configurado)
kotlin.incremental=true
ksp.incremental=true
```

4. Avoid Full Rebuilds

```
// En lugar de:
./gradlew clean build

// Hacer:
./gradlew :app:assembleDebug

// Clean solo cuando sea necesario (errores extraños)
```

📱 Compatibilidad minSdk 24

APIs a tener cuidado

Java Time API (requiere desugaring):

```
// Si usas:
import java.time.LocalDateTime
import java.time.ZonedDateTime

// Agregar en app/build.gradle.kts:
android {
```

```
    compileOptions {
        isCoreLibraryDesugaringEnabled = true
    }
}
dependencies {
    coreLibraryDesugaring("com.android.tools:desugar_jdk_libs:2.1.4")
}
```

Notification Channels (API 26+):

```
// Verificar versión antes de usar
if (Build.VERSION.SDK_INT ⩾ Build.VERSION_CODES.O) {
    createNotificationChannel()
}
```

Adaptive Icons (API 26+):

```

```

🔍 Debugging Tips

AGP 9 Changes

```
# Ver tareas disponibles
./gradlew tasks --all | grep app

# Dependencias tree
./gradlew :app:dependencies

# Ver configuración aplicada
./gradlew :app:properties
```

KSP Generated Code

```
# Regenerar código
./gradlew clean :app:kspDebugKotlin

# Ver logs de generación
./gradlew :app:kspDebugKotlin --info | grep KSP
```

Compose Debugging

```
// Ver recompositions en Logcat
import androidx.compose.runtime.SideEffect
```

```
@Composable
fun MyComposable() {
    SideEffect {
        Log.d("Compose", "MyComposable recomposed")
    }
    // ...
}
```

📚 Recursos Adicionales

- Kotlin 2.0 Migration Guide
- AGP 9 Known Issues
- KSP vs KAPT Performance
- Android Keystore Best Practices

Última actualización: 2026-01-23

Contacto: Ver AGENTS.md para convenciones del proyecto