Toggly Kotlin / Android

Kotlin coroutines · Jetpack Compose · Views · Room / DataStore storage

Install (Gradle KTS)

```
dependencies {
  implementation("io.toggly:toggly-android-core")
  // pick UI module
  implementation("io.toggly:toggly-compose")
  implementation("io.toggly:toggly-views")
  // pick storage module
  implementation("io.toggly:toggly-room")
  implementation("io.toggly:toggly-datastore")
}
```

Pin the same release on every `io.toggly` module (Maven Central or `docs.toggly.io/sdks/android`).

Initialize (Application)

```
class MyApp : Application() {
  override fun onCreate() {
    super.onCreate()
    Toggly.configure(
      config = TogglyConfig(
        appKey      = BuildConfig.TOGGLY_APP_KEY,
        environment = "Production",
        defaultFlags = mapOf(
          "NewCheckout" to false),
        refreshInterval = 60_000L,
        enableAutoRefresh = true,
        enableLogging = BuildConfig.DEBUG,
      ),
      storage = SharedPreferencesStorage(this)
    )
    ProcessLifecycleOwner.get()
      .lifecycleScope.launch {
        Toggly.shared.init()
      }
  }
}
```

Register `MyApp` in `AndroidManifest.xml` `android:name`.

Compose — setup

```
// wrap app screens in TogglyProvider
TogglyProvider {
  Dashboard()
}
```

Compose — composables

```
import io.toggly.compose.*

@Composable
fun Dashboard() {
  val newDash by rememberFeatureFlag(
    "new-dashboard")
  if (newDash) NewDashboard()
  else         LegacyDashboard()
}

@Composable fun Welcome() {
  FeatureFlag("welcome-banner") { WelcomeBanner() }

  FeatureFlag(
    featureKey = "new-checkout",
    enabled  = { NewCheckoutFlow() },
    disabled = { LegacyCheckoutFlow() })
}
```

Compose — feature gate

```
@Composable fun AdminSection() {
  FeatureGate(
    featureKeys = listOf(
      "admin-access","premium-tier"),
    requirement = FeatureRequirement.ALL,
  ) { AdminPanel() }
}
```

Toggly Kotlin / Android

Kotlin coroutines · Jetpack Compose · Views · Room / DataStore storage

Views — bind visibility

```
import io.toggly.views.*

class HomeActivity : AppCompatActivity() {
  override fun onCreate(s: Bundle?) {
    super.onCreate(s)
    setContentView(R.layout.activity_home)

    newFeatureBtn
      .bindToFeatureFlag("new-feature", this)

    toggleViews(
      featureKey = "new-checkout",
      lifecycleOwner = this,
      enabledView  = newCheckoutView,
      disabledView = legacyCheckoutView)
  }
}
```

ViewModel + LiveData

```
class HomeVM : FeatureFlagViewModel() {
  val newDash =
    featureFlagLiveData("new-dashboard")
}

// in Activity / Fragment
vm.newDash.observe(this) { on →
  newDashView.isVisible = on
}
```

Coroutines / Flow

```
// suspend — call from coroutine scope
lifecycleScope.launch {
  val on = Toggly.shared
    .isFeatureOn("my-feature")
}

// reactive
Toggly.shared.featureFlagFlow("my-feature")
  .collect { on → render(on) }

// multi-flag (suspend)
val ok = Toggly.shared.evaluateFeatureGate(
  featureKeys = listOf("a","b"),
  requirement = FeatureRequirement.ALL)
```

Identity

```
lifecycleScope.launch {
  Toggly.shared.setIdentity("user-123")  // login
  Toggly.shared.setIdentity(null)        // logout
  Toggly.shared.refresh()                // manual
}
```

Compose previews

```
@Preview @Composable
fun HomePreview() {
  val mock = MockTogglyService(
    flags = mapOf("new-dashboard" to true))
  TogglyProvider(service = mock) {
    Home()
  }
}
```

Feature variants

```
// TogglyConfig(enableVariants = true, ...)
lifecycleScope.launch {
  val variant = Toggly.shared
    .getVariant("checkout-button-color")
  when (variant?.name) {
    "blue"  → showBlueButton()
    "green" → showGreenButton()
    else    → showDefaultButton() // null
  }

  val value = Toggly.shared
    .getVariantValue("checkout-button-color")
}
```

`getVariant` / `getVariantValue` return `null` unless `enableVariants = true` and the feature is enabled with an assignment.

Storage options

```
val storage =
  SharedPreferencesStorage(context)

val storage = createRoomStorage(context)
val storage = createDataStoreStorage(context)
val storage = MemoryStorage()  // tests
```

DataStore is preferred for new apps; SharedPreferences works on every API level; Room helps if you already use it.

Events

```
lifecycleScope.launch {
  Toggly.shared.events.collect { e →
    when (e) {
      is TogglyEvent.Initialized → ready()
      is TogglyEvent.Refreshed   → rebind()
      is TogglyEvent.FeatureChanged → {
        Log.i("toggly",
          "${e.featureKey} → ${e.newValue}")
      }
      is TogglyEvent.Error → log(e.error)
    }
  }
}
```

Multi-module config

```
TogglyConfig(
  appKey      = appKey,
  environment = "Production",
  defaultFlags = mapOf(
    "feature-a" to true,
    "feature-b" to false),
  refreshInterval = 60_000L,
  enableAutoRefresh = true,
)
```

Testing

```
@Before fun setUp() {
  Toggly.reset()
  Toggly.configure(config = TogglyConfig(
    defaultFlags = mapOf("new-feature" to true)),
    storage = MemoryStorage())
  runBlocking { Toggly.shared.init() }
}
```

ProGuard / R8

PROGUARD-RULES.PRO

```
-keep class io.toggly.** { *; }
-keepclassmembers class io.toggly.** { *; }
-dontwarn io.toggly.**
```

Pitfalls

- Initialize Toggly in `Application.onCreate` — don't re-init per Activity; the SDK is a singleton.
- Always set `defaultFlags` — Android phones lose connectivity often (subway, lifts, captive Wi-Fi).
- Use `BuildConfig.TOGGLY_APP_KEY` via Gradle `buildConfigField` — never hard-code the key in source.
- `rememberFeatureFlag` is recomposition-safe; `collectAsState` on the flow yields the same effect.

Resources

- Docs: `docs.toggly.io/sdks/android`
- Maven Central: `io.toggly:toggly-android-*`
- GitHub: `github.com/ops-ai/Toggly.FeatureManagement`
- Android 7.0+ (API 24+) · Kotlin 2.0+ · JDK 17