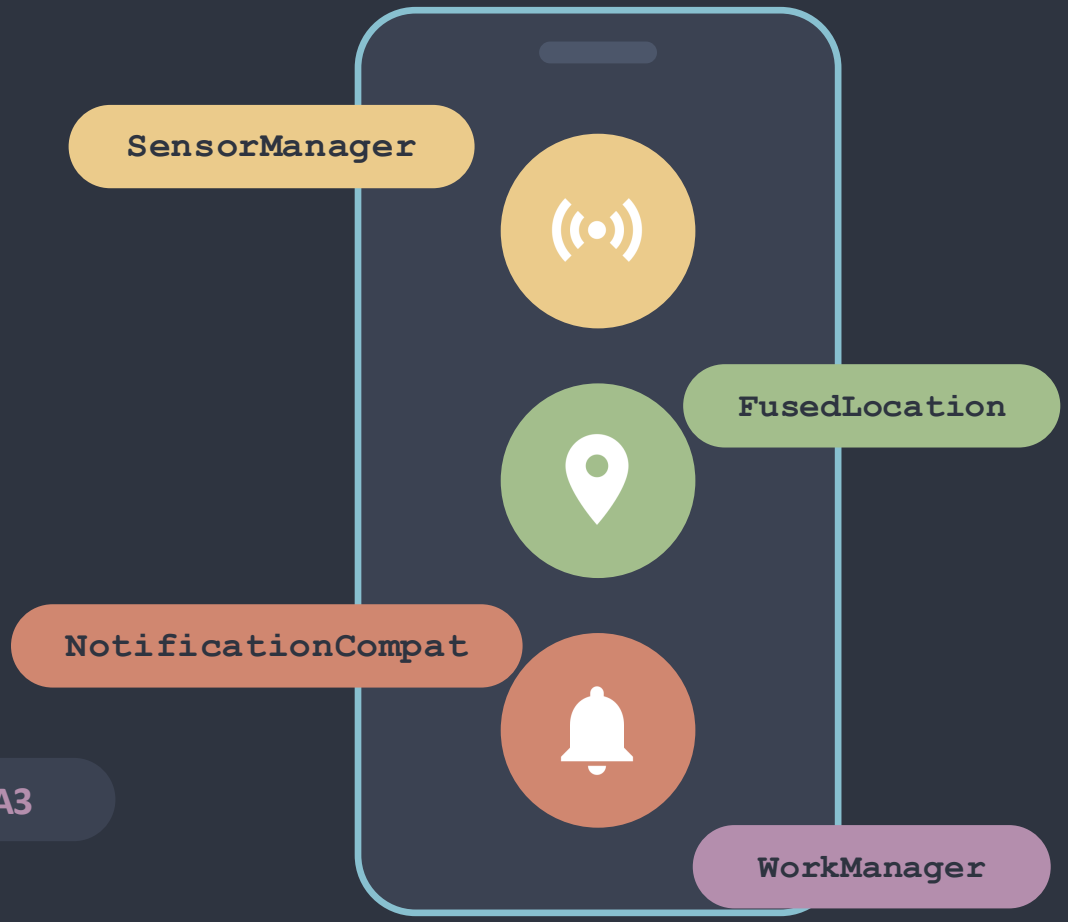
Bloque 4 · Tema 8

Sensores, GPS y Notificaciones

El dispositivo que siente, se localiza y avisa: hardware y segundo plano con Kotlin y Jetpack Compose

6 horas | Excursia · cuaderno de campo | RA2 · RA3



Programación Multimedia y Dispositivos Móviles (0489) · 2º CFGS DAM

Hoja de ruta del tema

6 apartados + práctica

1 **Excursia**
Pantallas, arquitectura y DTOs

2 **Dependencias y permisos**
Gradle y AndroidManifest

3 **Sensores**
SensorManager y callbackFlow

4 **GPS y localización**
FusedLocationProviderClient

5 **Permisos en tiempo de ejecución**
Ubicación fina o aproximada

6 **Notificaciones locales**
Canales, Builder, PendingIntent

7 **WorkManager**
Avisos con la app cerrada

8 **Ensamblaje**
AppContainer y navegación

9 **Práctica y reto**
Desarrollo guiado y Excursia+

¿Qué vamos a evaluar?

RA2 · RA3

RA2

Desarrolla aplicaciones para dispositivos móviles analizando y empleando las tecnologías y librerías específicas.

RA3

Desarrolla programas que integran contenidos multimedia analizando y empleando las tecnologías y librerías específicas.



RA2-d Uso de los sensores del dispositivo



RA3-a Entornos de desarrollo para aplicaciones multimedia



RA3-e Clases para eventos, tipos de medios y excepciones



RA3-f Animaciones (AnimatedVisibility en Entorno)



RA3-g Reproducción y presentación de contenidos multimedia

★ *RA2-d vertebra el tema: sensores, GPS y servicios del sistema*

00

Excursia

Un cuaderno de campo digital construido desde cero

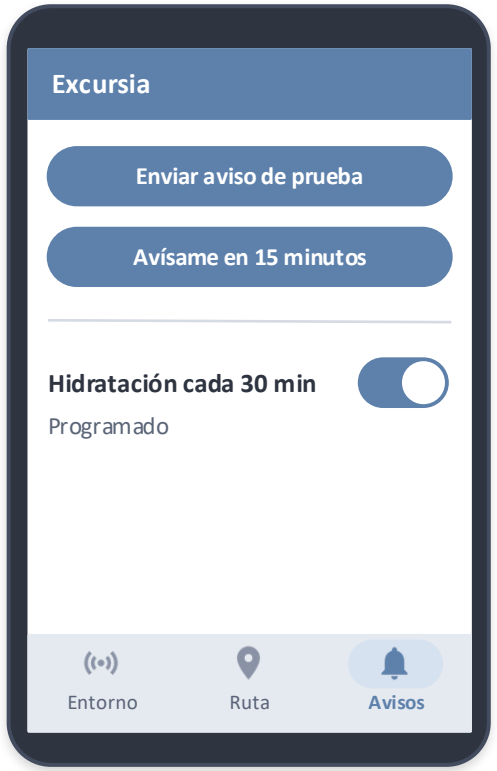


Tres pantallas, tres bloques técnicos

Excursia







Entorno

`SensorManager`

Acelerómetro, agitación, luz (lux) y proximidad

Ruta

`FusedLocationProviderClient`

Latitud, longitud, precisión, altitud y mapa

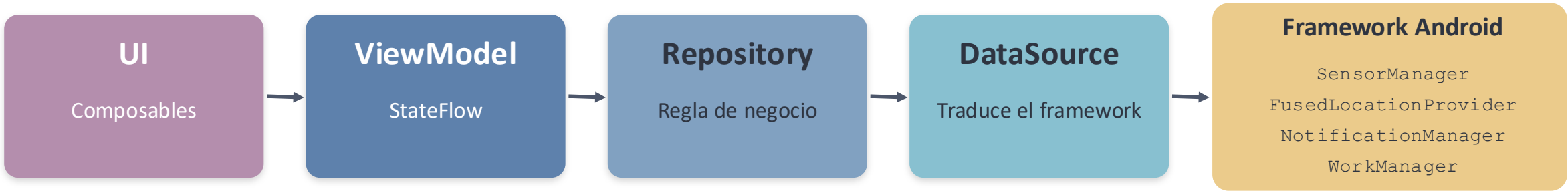
Avisos

`NotificationCompat + WorkManager`

Aviso de prueba, recordatorio y hidratación

La misma arquitectura de siempre

MVVM



Sin capa de dominio · Sin Hilt ni Koin · Contenedor escrito a mano

El framework nunca sube de capa

Solo el DataSource conoce SensorEvent, Location o WorkInfo. Hacia arriba viajan modelos propios: Vector3 y Posicion.

La regla de negocio vive en el repositorio

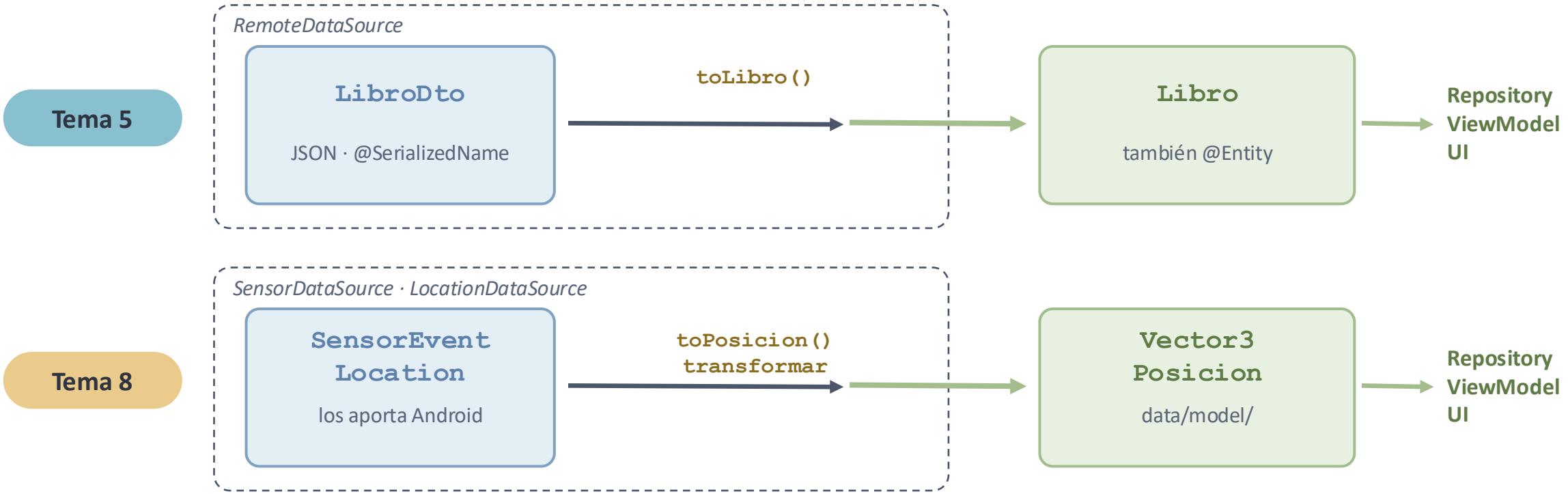
¿Cuándo se agita el móvil? ¿Qué es «penumbra»? Se decide en el Repository, no en el ViewModel ni en la UI.

Separación DTO ↔ modelo (Tema 5)

SensorEvent y Location hacen de DTO. No hay que escribirlos: los aporta el sistema.

¿Quién hace de DTO en este tema?

`DTO ↔ modelo`



Regla de oro: el DTO no sale del DataSource. Ningún ViewModel importa android.hardware.* ni android.location.*

01

Dependencias y permisos

Proyecto nuevo, Gradle y AndroidManifest



Crear el proyecto y añadir dependencias

Gradle KTS

Plantilla	Package	minSdk	Build
Empty Activity (Compose)	com.ejemplo.excursia	26 · canales de notificación	Kotlin DSL

play-services-location	21.4.0	FusedLocationProviderClient
kotlinx-coroutines-play-services	1.10.2	Task<T>.await() en funciones suspend
work-runtime	2.11.2	WorkManager y CoroutineWorker
core	1.18.0	NotificationCompat y ContextCompat
navigation-compose	2.9.8	Las tres pestañas, rutas type-safe
lifecycle-*-compose	2.10.0	collectAsStateWithLifecycle y viewModel()
compose-bom + material-icons-extended	2026.06.01	Iconos Sensors, Place, Map…

Artefactos vacíos: work-runtime-ktx (desde 2.9) y core-ktx (desde 1.19). Usar **work-runtime** y **core**.

Permisos: solo los imprescindibles

AndroidManifest

ACCESS_COARSE_LOCATION

Red · ~100-300 m

Sí

ACCESS_FINE_LOCATION

GPS · ~10 m

Sí

ACCESS_BACKGROUND_LOCATION

Segundo plano · comentado

No

POST_NOTIFICATIONS

Obligatorio desde API 33

Sí

ACTIVITY_RECOGNITION

Contador de pasos · API 29+

Reto

Sensores: sin permiso

Acelerómetro, luz y proximidad no requieren ningún permiso.

BODY_SENSORS, obsoleto

Con targetSdk 36 (Android 16) se sustituye por permisos granulares de android.permissions.health:

READ_HEART_RATE

READ_HEALTH_DATA_IN_BACKGROUND

02

Sensores del dispositivo

SensorManager, callbackFlow y la pantalla Entorno



Tres familias de sensores

`SensorManager`

Movimiento

Agitación, giros, pasos

- `TYPE_ACCELEROMETER`
- `TYPE_LINEAR_ACCELERATION`
- `TYPE_GRAVITY`
- `TYPE_GYROSCOPE`
- `TYPE_STEP_COUNTER`



Posición

Brújula, móvil en el bolsillo

- `TYPE_MAGNETIC_FIELD`
- `TYPE_ROTATION_VECTOR`
- `TYPE_PROXIMITY`



Entorno

Brillo automático, altímetro

- `TYPE_LIGHT`
- presión
- temperatura



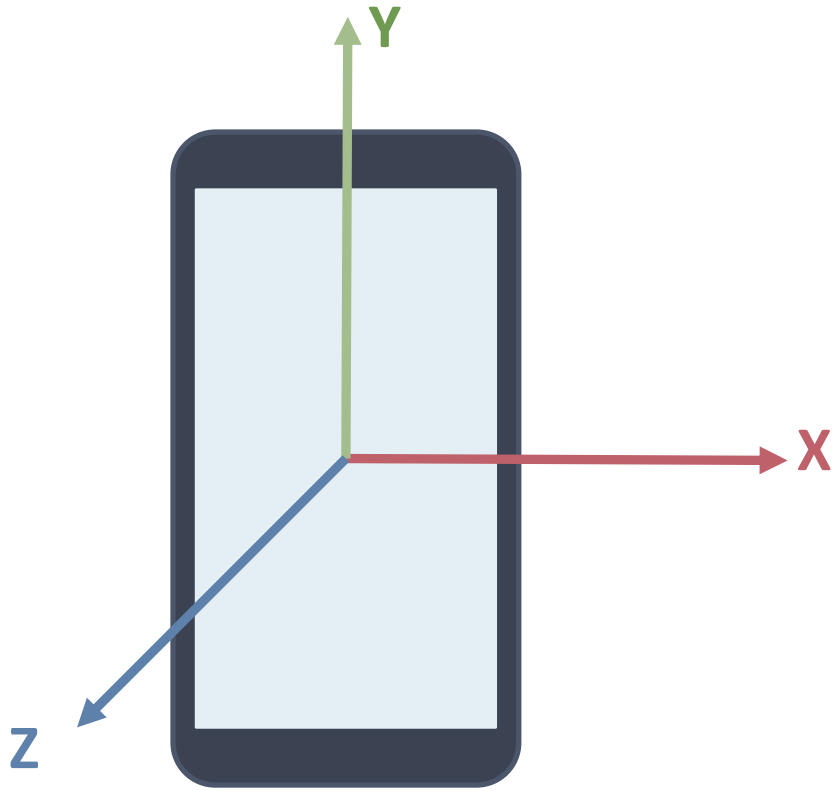
`getDefaultSensor()` puede devolver null: mostrar un mensaje, nunca lanzar una excepción.



Emulador: Extended Controls → Virtual sensors

Ejes del acelerómetro y el modelo Vector3

Vector3



Z sale de la pantalla hacia el usuario

$$|a| = \sqrt{(x^2 + y^2 + z^2)}$$

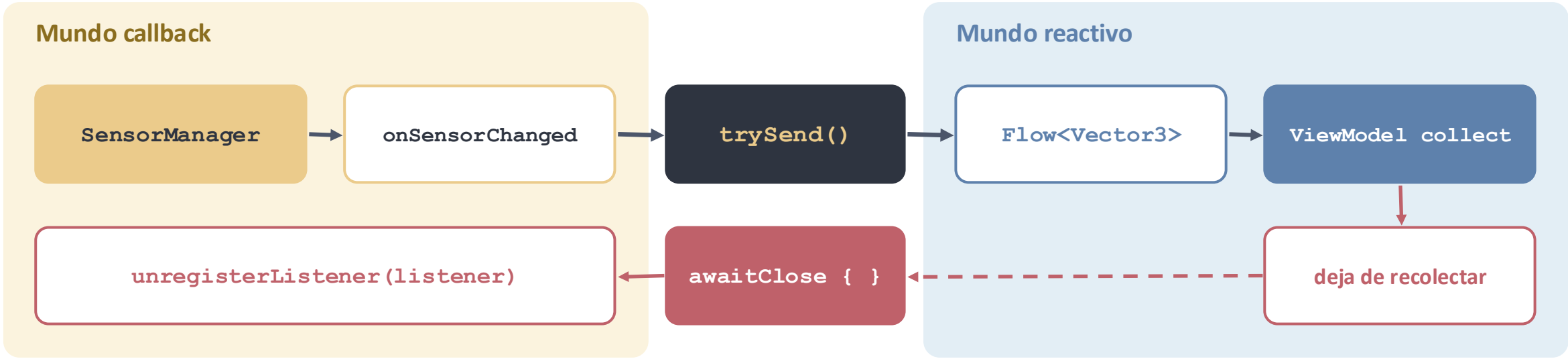
$\approx 9{,}81\ m/s^2$ en reposo: es la gravedad

data/model/Vector3.kt

```
data class Vector3(
    val x: Float = 0f,
    val y: Float = 0f,
    val z: Float = 0f
) {
    /** Módulo del vector */
    val magnitud: Float
        get() = sqrt(x * x + y * y + z * z)
}
```

callbackFlow: del callback al Flow

`callbackFlow`



Sin awaitClose

El listener sigue registrado: el sensor continúa activo y consume batería indefinidamente. Es el error más frecuente del tema.

¿Por qué no flow { }?

onSensorChanged() se ejecuta en otro hilo. callbackFlow crea un canal que admite trySend() externos y ofrece awaitClose.

SensorDataSource paso a paso

DataSource

data/datasource/local/SensorDataSource.kt

```kotlin
private fun <T> observarSensor(
    tipo: Int,
    frecuencia: Int = SensorManager.SENSOR_DELAY_NORMAL,
    transformar: (SensorEvent, Sensor) -> T
): Flow<T> = callbackFlow {
    val sensor = sensorManager.getDefaultSensor(tipo)
    if (sensor == null) {
        close(SensorNoDisponibleException(tipo))
        return@callbackFlow
    }
    val listener = object : SensorEventListener {
        override fun onSensorChanged(event: SensorEvent) {
            trySend(transformar(event, sensor))
        }
        override fun onAccuracyChanged(s: Sensor?, a: Int) {}
    }
    sensorManager.registerListener(listener, sensor, frecuencia)
    awaitClose { sensorManager.unregisterListener(listener) }
}
```

1

Sensor ausente

Se cierra el Flow con SensorNoDisponibleException.

2

Mapeo DTO → modelo

transformar convierte el SensorEvent en el modelo propio.

3

Registro

registerListener con la frecuencia elegida (NORMAL ≈ 200 ms).

4

Liberación

awaitClose, siempre al final del bloque.

Públicas: `observarAcelerometro()` · `observarLuzAmbiental()` · `observarProximidad()`

EntornoRepository: dar significado al dato

Repository

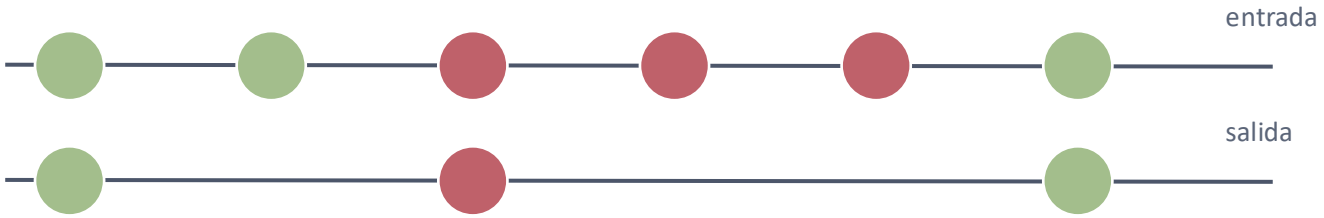
Nivel de luz (lux)

OSCURIDAD	PENUMBRA	INTERIOR	EXTERIOR NUBLADO	SOL DIRECTO
< 10	10 – 100	100 – 1 000	1 000 – 10 000	≥ 10 000

Agitación: magnitud > 15 m/s²



`distinctUntilChanged()`




```
EntornoRepositoryImpl

override val agitacion: Flow<Boolean> =
    aceleracion
        .map { it.magnitud >
UMBRAL_AGITACION }
        .distinctUntilChanged()

override val nivelLuz: Flow<NivelLuz> =
    luzAmbiental
        .map { lux -> when { ... } }
        .distinctUntilChanged()
```

EntornoViewModel: Flow → StateFlow

ViewModel

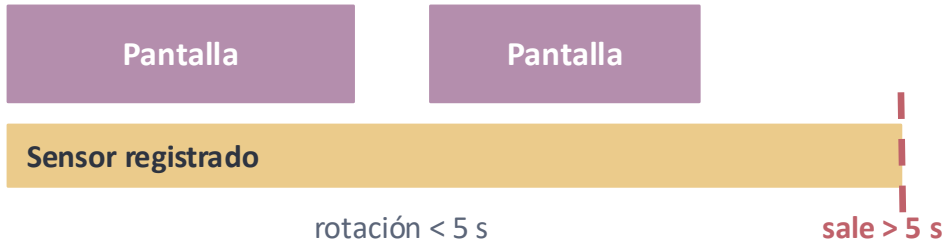
ui/entorno/EntornoViewModel.kt

```
val aceleracion: StateFlow<Vector3> =
    entornoRepository.aceleracion
        .catch { emit(Vector3()) }
        .stateIn(
            viewModelScope,
            SharingStarted.WhileSubscribed(5_000),
            Vector3()
        )
```

.catch → valor neutro si el sensor falta

aceleracion	Vector3()	luzAmbiental	0f
nivelLuz	INTERIOR	agitacion	false
objetoCerca	false		

WhileSubscribed(5_000) en el tiempo

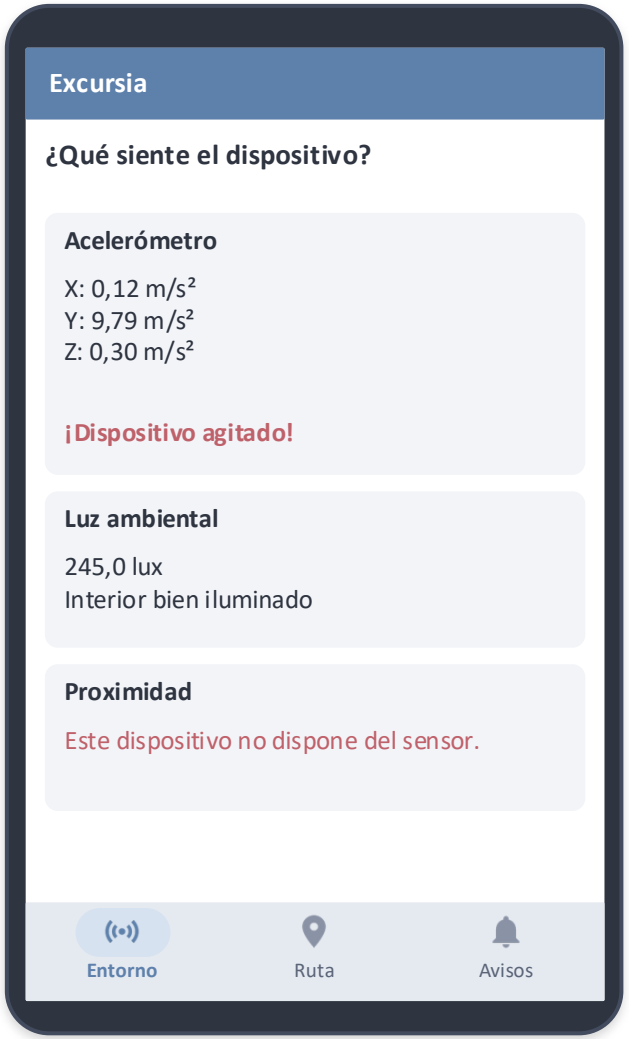


ViewModel, no AndroidViewModel. El Context lo necesita el SensorManager, y está encapsulado en el DataSource.

Factory manual: viewModelFactory { initializer { … } } toma el repositorio de **app.container**

PantallaEntorno: nunca suponer el hardware

Compose



`collectAsStateWithLifecycle()`

Detiene la recolección en STOPPED: junto a WhileSubscribed, libera el sensor.

`TarjetaSensor(titulo, disponible, contenido)`

Composable privado reutilizable: si falta el sensor, muestra un aviso.

`AnimatedVisibility(visible = agitacion)`

El aviso de agitación aparece y desaparece animado (RA3-f).

Probar los tres caminos

Sensor presente, sensor ausente y rotación de pantalla.

03

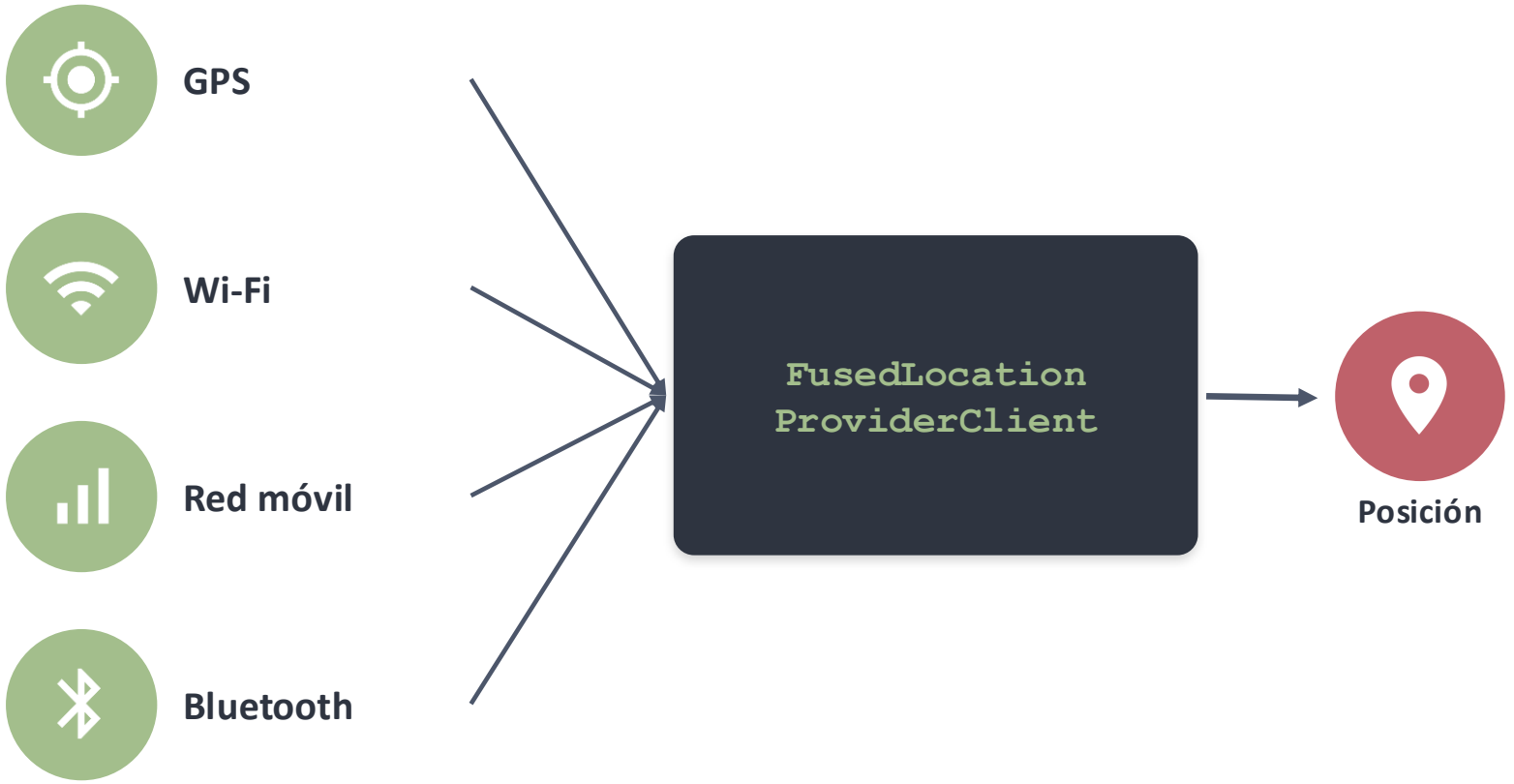
GPS y localización

FusedLocationProviderClient, permisos y la pantalla Ruta



Fused Location: varias fuentes, una posición

Play Services



Fused (recomendado)

Fusiona proveedores y gestiona la batería. Necesita Google Play Services.

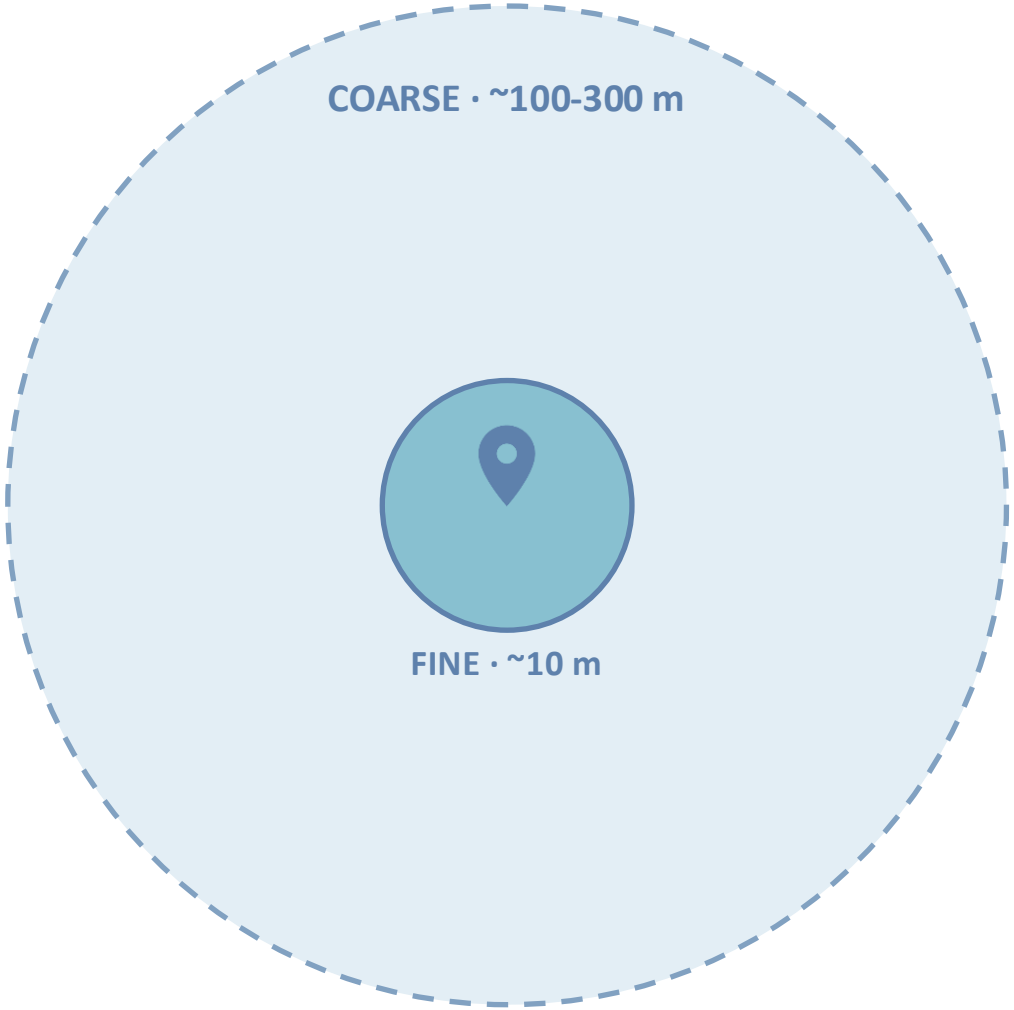
LocationManager

API nativa. Alternativa sin Play Services: algunos emuladores, Android TV, ciertos fabricantes.

Emulador: imagen «Google APIs» o «Google Play» y Extended Controls → Location (punto, GPX/KML o ruta reproducida).

Permisos de ubicación: fina o aproximada

Permisos



Android 12+ (API 31)

Aunque se pida FINE, el usuario puede conceder solo la aproximada. La app debe comprobar cuál obtuvo: precisionAproximada.

`RequestMultiplePermissions`

Se piden FINE y COARSE a la vez: un único diálogo con las dos opciones de precisión.

`ACCESS_BACKGROUND_LOCATION`

Exige justificación ante Google Play y se pide después de FINE. Excursia no lo usa.

LocationDataSource: ubicaciones como Flow

DataSource

data/datasource/local/LocationDataSource.kt

```
fun observarPosicion(intervaloMs: Long = 5_000L)
    : Flow<Posicion> = callbackFlow {
    val solicitud = LocationRequest.Builder(
        Priority.PRIORITY_HIGH_ACCURACY, intervaloMs)
        .setMinUpdateIntervalMillis(intervaloMs / 2)
        .setMinUpdateDistanceMeters(5f)
        .setWaitForAccurateLocation(true)
        .build()
    val callback = object : LocationCallback() {
        override fun onLocationResult(r: LocationResult) {
            r.lastLocation?.let { trySend(it.toPosicion()) }
        }
    }
    fusedClient.requestLocationUpdates(
        solicitud, callback, Looper.getMainLooper())
        .addOnFailureListener { e -> close(e) }
    awaitClose { fusedClient.removeLocationUpdates(callback) }
}
```

PRIORITY_HIGH_ACCURACY

Usa GPS: máxima precisión, más consumo

intervalo / 2

Ritmo máximo aceptado

5 m

Ignora cambios menores

waitForAccurateLocation

Espera un primer fix bueno

toPosicion()

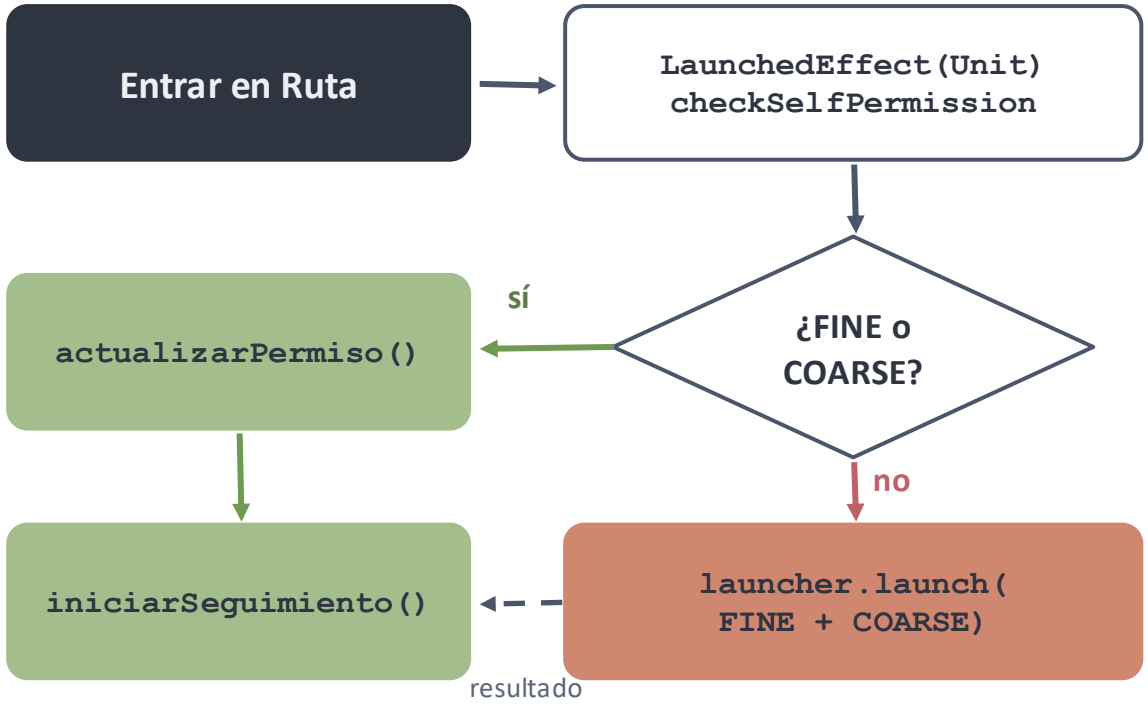
Location → Posicion (privada)

removeLocationUpdates

Sin él, el GPS vacía la batería

PantallaRuta: el permiso decide qué se ve

Compose



`DisposableEffect` → `onDispose { detenerSeguimiento() }`
Al salir de la pantalla se apaga el GPS.

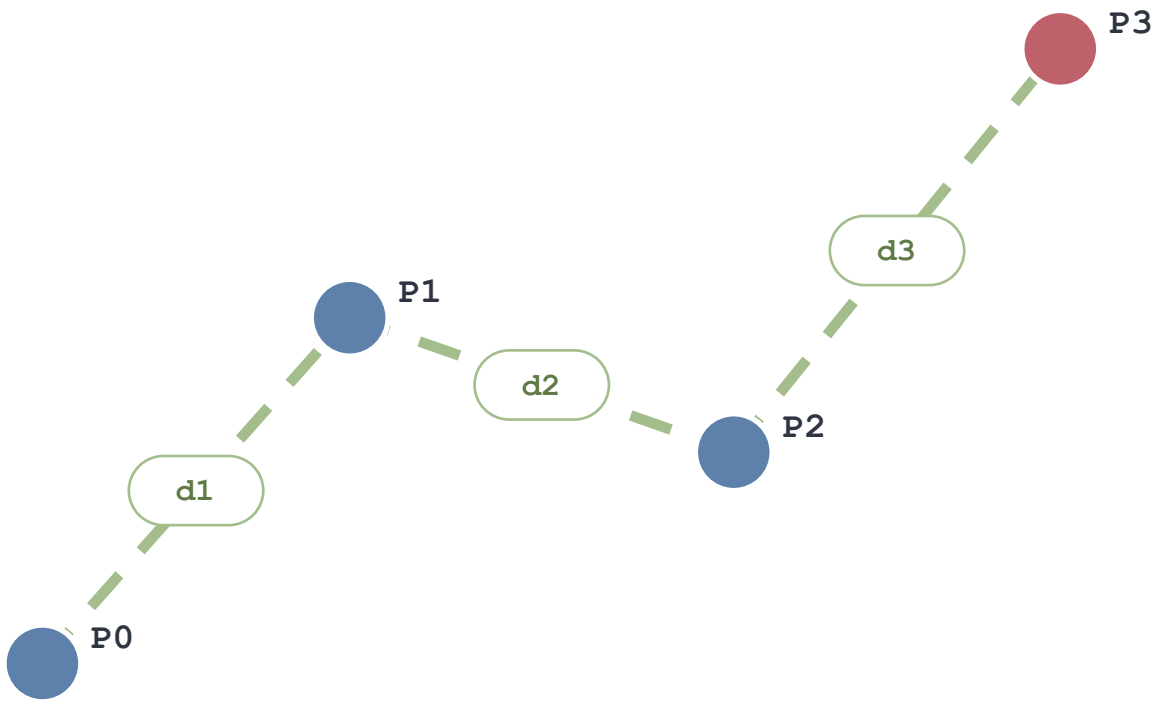
Tres estados de la UI

1. **Sin permiso**
   Texto + botón «Conceder permiso»
2. **Buscando señal**
   CircularProgressIndicator
3. **Datos**
   Coordenadas, altitud, precisión y distancia

RutaViewModel: acumular la distancia

ViewModel

Location.distanceBetween(…) en RutaRepository: estática, sin hardware ni permisos




```
distanciaRecorrida = d1 + d2 + d3
```

ui/ruta/RutaViewModel.kt

```kotlin
data class RutaUiState(
    val permisoConcedido: Boolean = false,
    val precisionAproximada: Boolean = false,
    val posicion: Posicion? = null,
    val distanciaRecorrida: Float = 0f,
    val error: String? = null
)
```

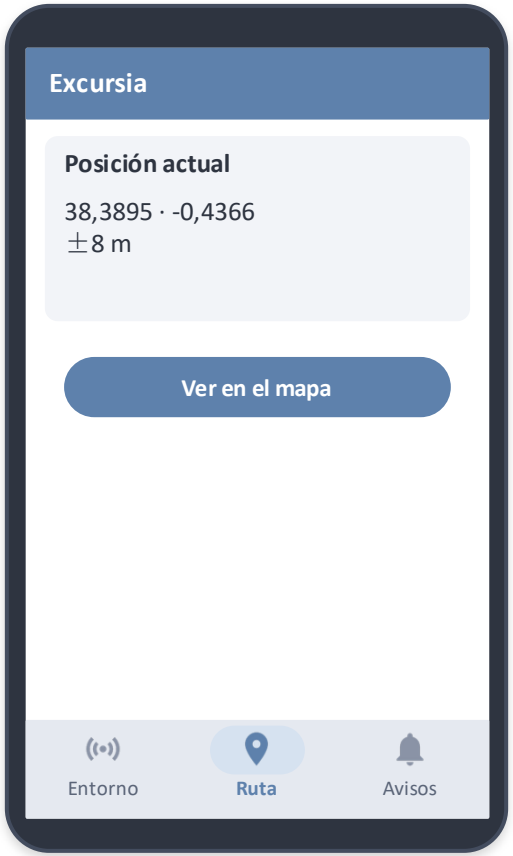
`iniciarSeguimiento()` — Idempotente: si el Job está activo, no hace nada

`detenerSeguimiento()` — Cancela el Job → dispara awaitClose

`ultimaPosicionConocida()` — Se pinta al instante mientras llega el GPS

«Ver en el mapa»: Intent implícito geo:

Intent




```
ACTION_VIEW
geo:lat,lon?q=…
```

Cualquier app de mapas

Sin setPackage(): no se fuerza ninguna.

Ninguna app instalada

```
catch (ActivityNotFoundException)
```

¿Por qué no resolveActivity()? Desde API 30 (visibilidad de paquetes) devuelve null sin un bloque <queries>.

04

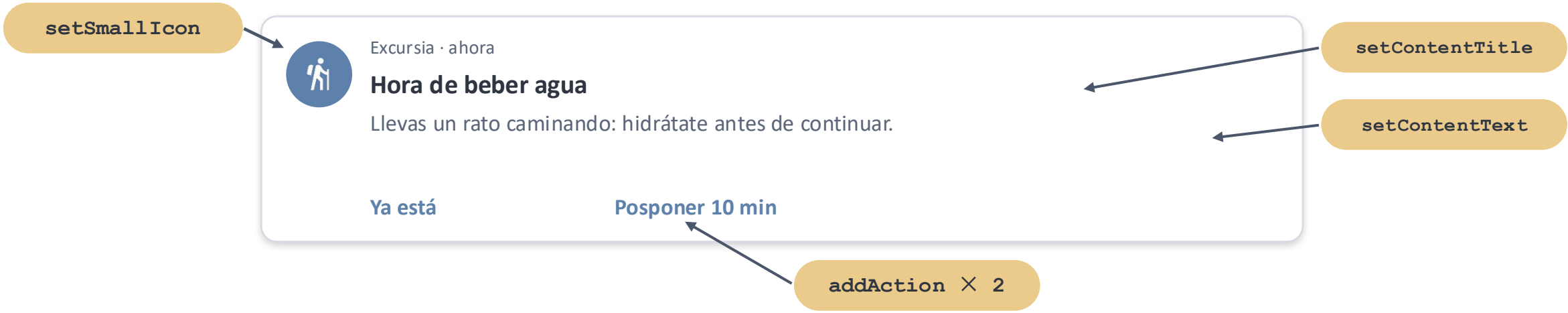
Notificaciones locales

Canales, NotificationCompat, acciones y permisos



Anatomía de una notificación

`NotificationCompat`



`NotificationChannel`

Agrupa por categoría. Obligatorio desde API 26.

`NotificationCompat.Builder`

Icono, título, texto, estilo y prioridad.

`PendingIntent`

Acción diferida que el sistema ejecuta en nombre de la app.

`NotificationManagerCompat`

Publica, actualiza y cancela.

Canales: el usuario decide qué silenciar

`NotificationChannel`



Alertas de ruta

`CANAL_ALERTAS`

`IMPORTANCE_DEFAULT`

Sonido y posible heads-up.
Hidratación, descanso y avisos.



Ruta en curso

`CANAL_RUTA`

`IMPORTANCE_LOW`

Silenciosa: solo en la barra de estado.

IDs de notificación

`ID_AVISO`	`1001`
`ID_RUTA_EN_CURSO`	`1002`
`ID_HIDRATACION`	`1003`

Un mismo ID actualiza o cancela la notificación existente.

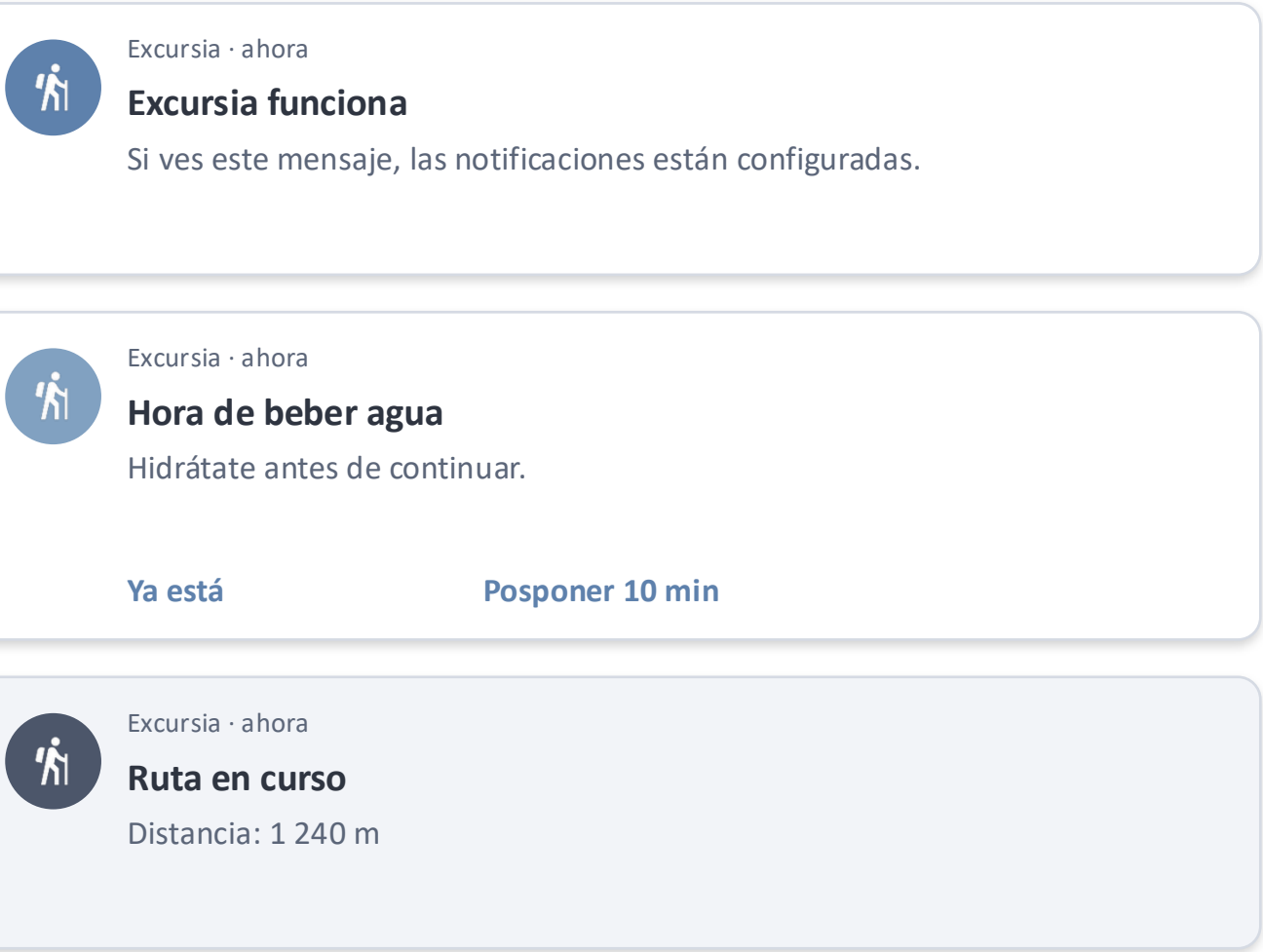
```
ExcursiaApplication.kt

override fun onCreate() {
    super.onCreate()
    container = DefaultAppContainer(this)
    container.notificador.crearCanales()    // idempotente
}
```

Los canales deben existir antes de publicar la primera notificación.

Tres notificaciones de NotificadorExcursia

`Notificador`



`mostrarAviso()`

BigTextStyle · setAutoCancel(true)

`mostrarRecordatorioHidratacion()`

Dos acciones con getBroadcast

`mostrarRutaEnCurso()`

setOngoing · setOnlyAlertOnce · CANAL_RUTA

La de ruta en curso aún no se usa: es el punto de partida del reto.

Botones de acción: BroadcastReceiver

Receiver



onReceive dura pocos segundos

Se ejecuta en el hilo principal: el trabajo diferido se delega en WorkManager.

ACCION_HECHO

```
cancelar(ID_HIDRATACION)
```

«Ya está»

ACCION_POSPONER

```
1. cancelar(ID)

2. repository
   .posponerHidratacion(10)
```

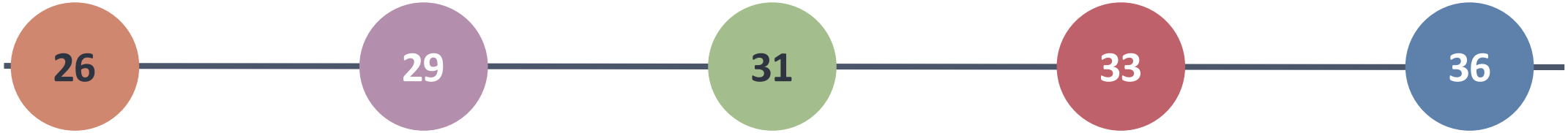
WorkManager

AndroidManifest.xml

```
<receiver
    android:name=".data.notificaciones.AccionesNotificacionReceiver"
    android:exported="false" />
```

Cambios de Android que afectan al tema

API



Android 8

Canales de notificación obligatorios

Android 10

ACTIVITY_RECOGNITION para contar pasos

Android 12

FLAG_IMMUTABLE obligatorio · ubicación aproximada

Android 13

POST_NOTIFICATIONS en tiempo de ejecución

Android 16

BODY_SENSORS → permisos de salud

ui/avisos/PermisoNotificaciones.kt (simplificado)

```kotlin
@Composable
fun recordarPermisoNotificaciones(): Pair<Boolean, () -> Unit> {
    var concedido by remember { mutableStateOf(/* API < 33 → true */) }
    val launcher = rememberLauncherForActivityResult(
        ActivityResultContracts.RequestPermission()
    ) { resultado -> concedido = resultado }
    return concedido to { launcher.launch(POST_NOTIFICATIONS) }
}
```

Sin permiso no hay excepción

Desde API 33 la notificación simplemente no aparece. puedeNotificar() lo comprueba antes para no «perseguir fantasmas».

05

WorkManager

Recordatorios que llegan aunque la app esté cerrada



¿Qué herramienta uso en cada caso?

Decisión

Recordatorio dentro de X minutos	→ `OneTimeWorkRequest`
Recordatorio cada 30 min durante la ruta	→ `PeriodicWorkRequest`
Sincronización periódica con una API	→ `PeriodicWorkRequest`
Tarea con la app abierta y visible	→ `viewModelScope.launch`
Seguimiento GPS continuo y visible	→ `Foreground Service`
Aviso a una hora exacta	→ `AlarmManager`

15 min

intervalo mínimo de un PeriodicWorkRequest

Garantiza que se ejecuta, no cuándo

El sistema puede agrupar tareas para ahorrar batería.

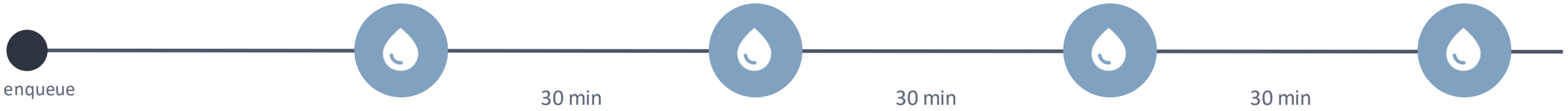
Puntual frente a periódico

`WorkRequest`

`OneTimeWorkRequest`



`PeriodicWorkRequest`




```
OneTimeWorkRequestBuilder<RecordatorioWorker>()
    .setInputData(datos)
    .setInitialDelay(minutos, TimeUnit.MINUTES)
    .addTag(ETIQUETA_RECORDATORIO)
    .build()
```

```
PeriodicWorkRequestBuilder<HidratacionWorker>(
    30, TimeUnit.MINUTES)
    .setConstraints(Constraints.Builder()
        .setRequiresBatteryNotLow(true).build())
    .build()
```

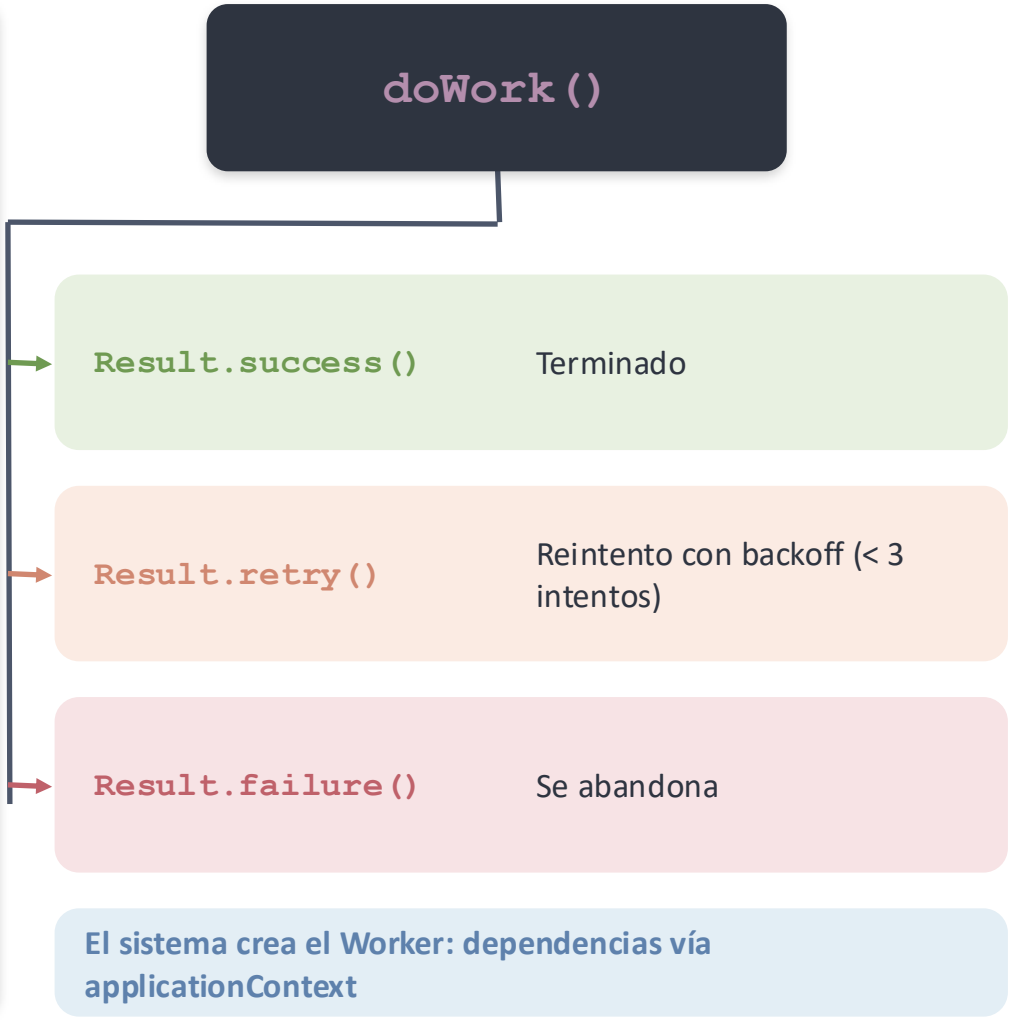
CoroutineWorker y el resultado del trabajo

Worker

```
data/workers/RecordatorioWorker.kt

class RecordatorioWorker(
    context: Context, params: WorkerParameters
) : CoroutineWorker(context, params) {

    override suspend fun doWork(): Result {
        val titulo = inputData.getString(CLAVE_TITULO)
            ?: "Recordatorio de Excursia"
        val notificador = (applicationContext
            as ExcursiaApplication).container.notificador
        return try {
            notificador.mostrarAviso(titulo, mensaje)
            Result.success()
        } catch (e: Exception) {
            if (runAttemptCount < 3) Result.retry()
            else Result.failure()
        }
    }
}
```



Del estado del trabajo al Switch

TrabajosDataSource

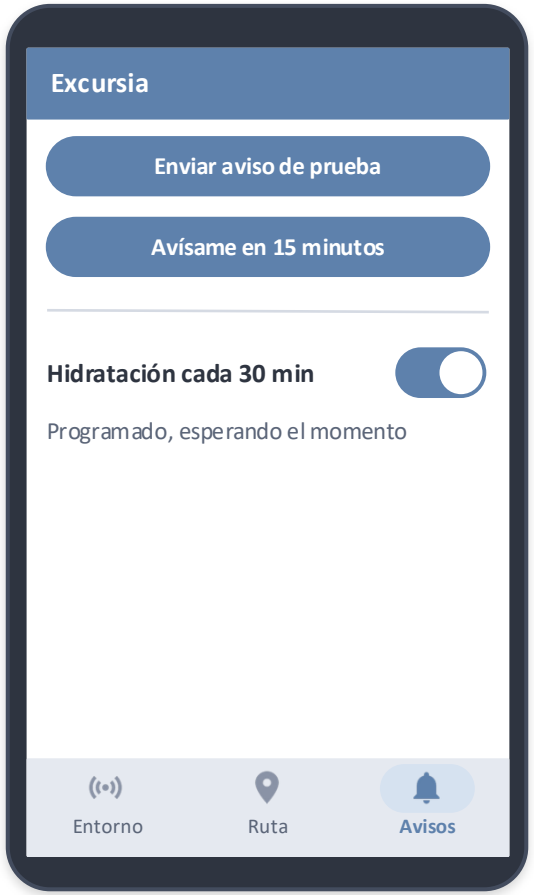
```
getWorkInfosForUniqueWorkFlow
.map { it.firstOrNull()?.state }
```

RecordatoriosRepository

```
estadoHidratacion
INTERVALO_HIDRATACION_MIN = 30
```

AvisosViewModel

```
stateIn(… WhileSubscribed(5_000), null)
```



`WorkInfo`

`ENQUEUED`
Programado · Switch ON

`RUNNING`
Ejecutándose · Switch ON

`CANCELLED · null`
Desactivado · Switch OFF

Sin permiso de notificaciones, PantallaAvisos muestra solo una tarjeta con «Activar notificaciones» y oculta el resto.

Prueba: cerrar la app desde recientes y esperar el aviso

06

Juntar las piezas

AppContainer, Application, MainActivity y navegación



AppContainer: inyección manual de dependencias

Sin Hilt ni Koin



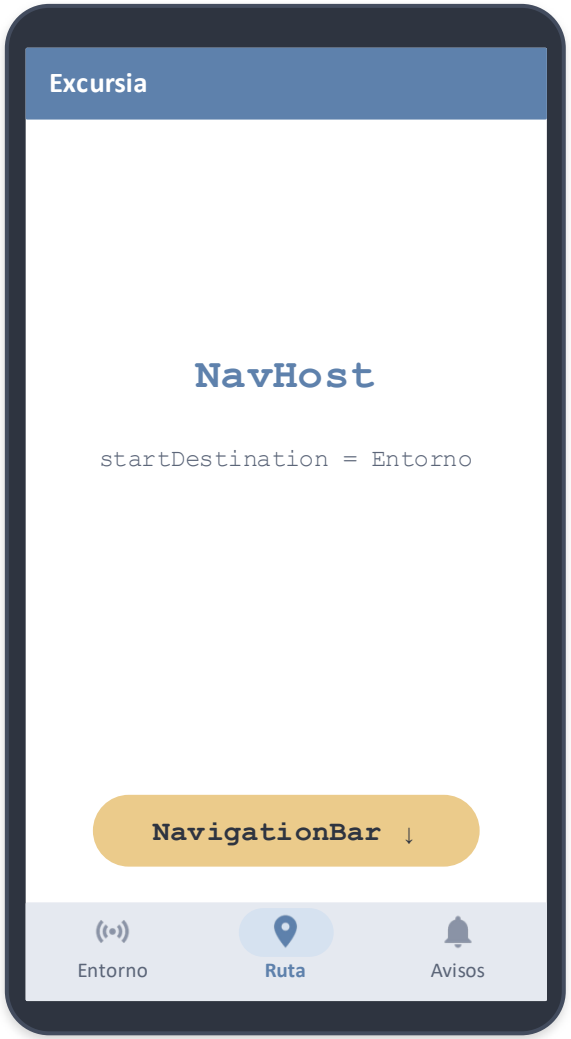
data/AppContainer.kt

```
interface AppContainer {
    val entornoRepository: EntornoRepository
    val rutaRepository: RutaRepository
    val recordatoriosRepository:
        RecordatoriosRepository
    val notificador: NotificadorExcursia
}
// DefaultAppContainer: cada una "by lazy"
```

android:name=".ExcursiaApplication"
Si falta en el manifiesto: ClassCastException al abrir una pantalla.

Navegación entre las tres pantallas

Navigation Compose



ui/navegacion/NavegacionExcursia.kt

```
@Serializable object Entorno
@Serializable object Ruta
@Serializable object Avisos

NavigationBarItem(
    selected = destinoActual?.hierarchy?.any {
        it.hasRoute(destino.ruta::class) } == true,
    onClick = { navController.navigate(destino.ruta) {
        launchSingleTop = true; restoreState = true
        popUpTo(Entorno) { saveState = true } } }, ...)
```

Entorno
`Icons.Default.Sensors`

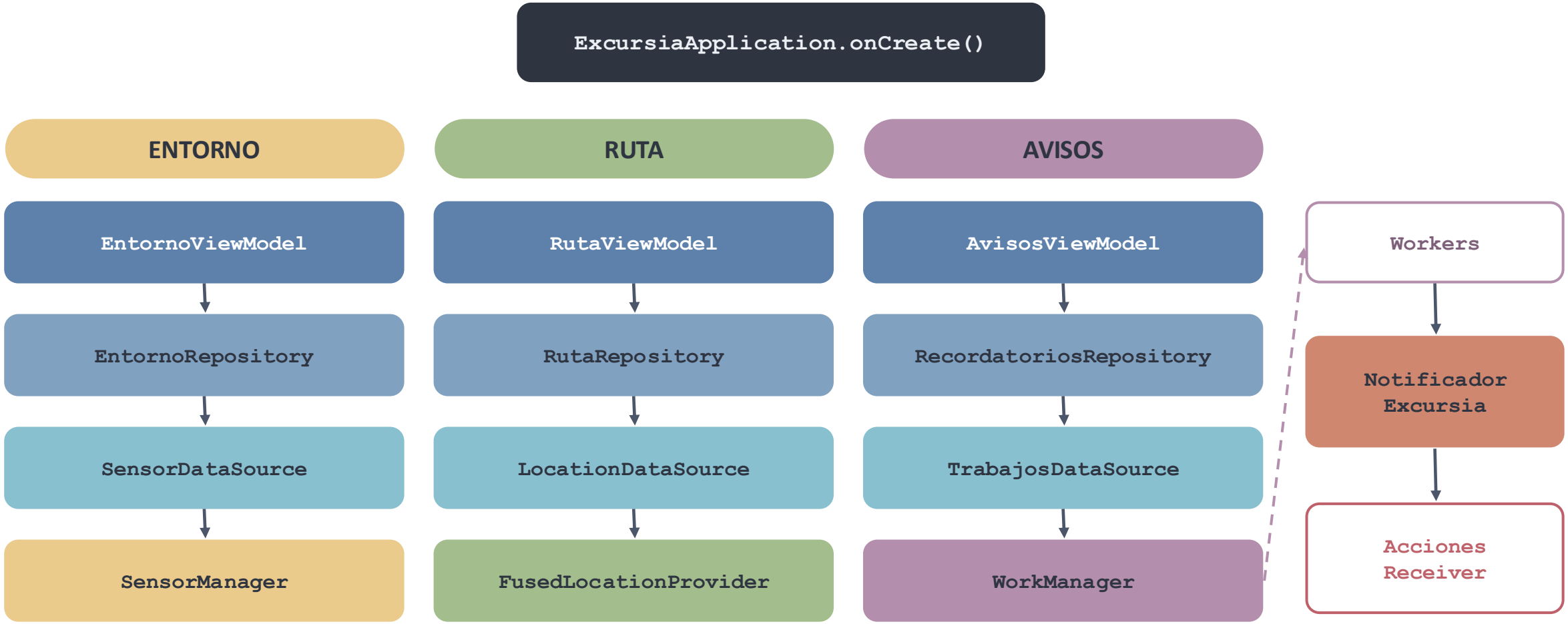
Ruta
`Icons.Default.Place`

Avisos
`Icons.Default.Notifications`

Rutas type-safe: necesitan el plugin kotlin("plugin.serialization")

Diagrama de flujo de Excursia

Visión global



DataSources y repositorios: DefaultAppContainer · canales: crearCanales()

Estructura de paquetes

Excursia

```
com.ejemplo.excursia/
├─ ExcursiaApplication.kt  · MainActivity.kt
├─ data/
│  ├─ AppContainer.kt
│  ├─ model/  Vector3.kt  · Posicion.kt
│  ├─ datasource/local/
│  │  ├─ SensorDataSource  · LocationDataSource  · TrabajosDataSource
│  ├─ repository/  Entorno  · Ruta  · Recordatorios
│  ├─ notificaciones/  Notificador  · AccionesReceiver
│  ├─ workers/  RecordatorioWorker.kt
├─ ui/
│  ├─ navegacion/  entorno/  ruta/  avisos/  theme/
```

model/

Modelos propios que suben de capa

datasource/local/

Única zona que importa el framework

repository/

Reglas de negocio

notificaciones/ · workers/

Piezas que crea el sistema

07

Practicar

Desarrollo guiado por etapas y reto Excursia+



Desarrollo práctico guiado: cinco etapas

`Git tags`



Proyecto base

Dependencias, permisos, paquetes y clases vacías

Entorno

Vector3, SensorDataSource, repositorio, ViewModel y pantalla

Ruta

Posicion, LocationDataSource, permisos y distancia

Notificaciones

Drawables, canales, receiver y aviso de prueba

WorkManager y navegación

Workers, contenedor completo y NavigationBar

Antes de dar el tema por terminado

- Compila sin código deprecated
- Todo callbackFlow libera en awaitClose
- Ningún ViewModel importa el framework
- Funciona sin permisos, sensores ni GPS

Reto final: Excursia+

Trabajo autónomo

Brújula

Obligatorio

TYPE_ROTATION_VECTOR, getOrientation(), Modifier.rotate y animateFloatAsState

Alerta de alejamiento

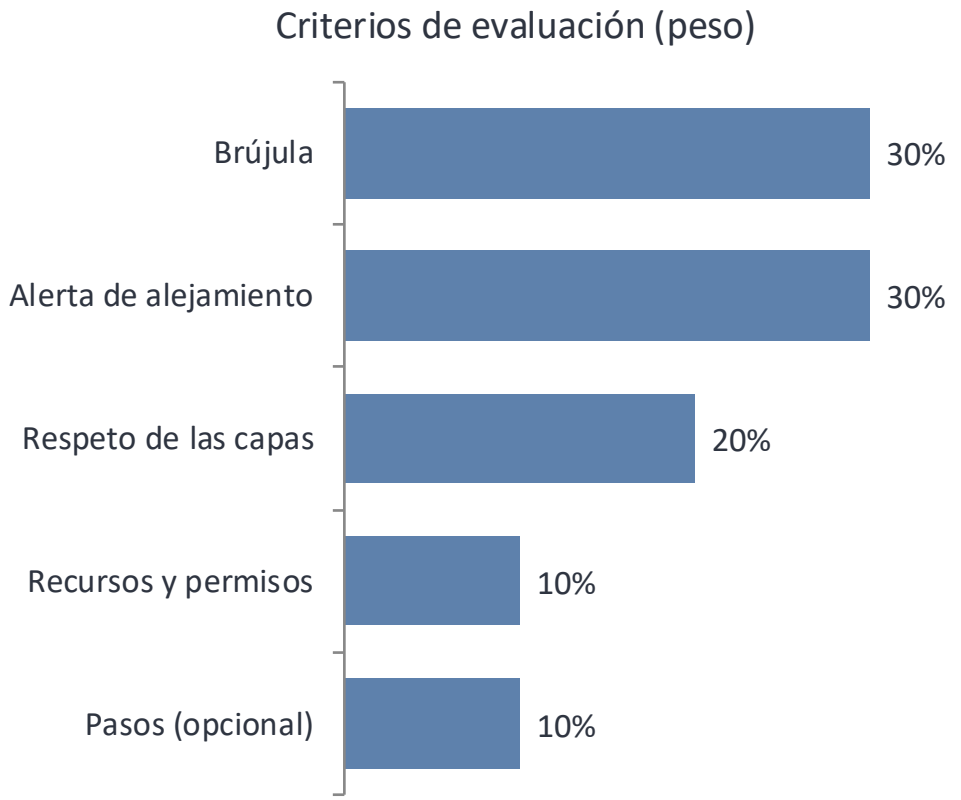
Obligatorio

Una sola notificación por cruce del umbral, con botón «Volver al inicio»

Contador de pasos

Opcional

TYPE_STEP_COUNTER + ACTIVITY_RECOGNITION; restar el valor inicial



Entrega: repositorio Git + README.md + vídeo de 2 minutos

Ideas clave



El DTO no sale del DataSource

SensorEvent y Location se mapean a Vector3 y Posicion.



callbackFlow + awaitClose

Del listener al Flow, y liberar siempre el recurso.



Nunca suponer el hardware

Sensor nulo, sin GPS o sin permiso: mensaje, no excepción.



Fina o aproximada

Pedir ambas y comprobar cuál se concedió.



Canal + permiso + inmutable

API 26, API 33 y FLAG_IMMUTABLE (API 31).



WorkManager en diferido

Llega con la app cerrada; periódico ≥ 15 min.

Referencias

Fuentes

Descripción general de sensores
developer.android.com/guide/topics/sensors/sensors_overview

Sensores de movimiento
developer.android.com/guide/topics/sensors/sensors_motion

Solicitar actualizaciones de ubicación
developer.android.com/training/location/request-updates

Notificaciones y canales
developer.android.com/develop/ui/views/notifications/channels

Descripción general de WorkManager
developer.android.com/topic/libraries/architecture/workmanager

Cambios de comportamiento en Android 16
developer.android.com/about/versions/16/behavior-changes-16

Tema 8. Sensores, GPS y Notificaciones (apuntes)
documentation.javiercarrasco.es/pmdm/B4/B4-T8_Sensores_GPS_Notificaciones/