Bloque 3 · Tema 6

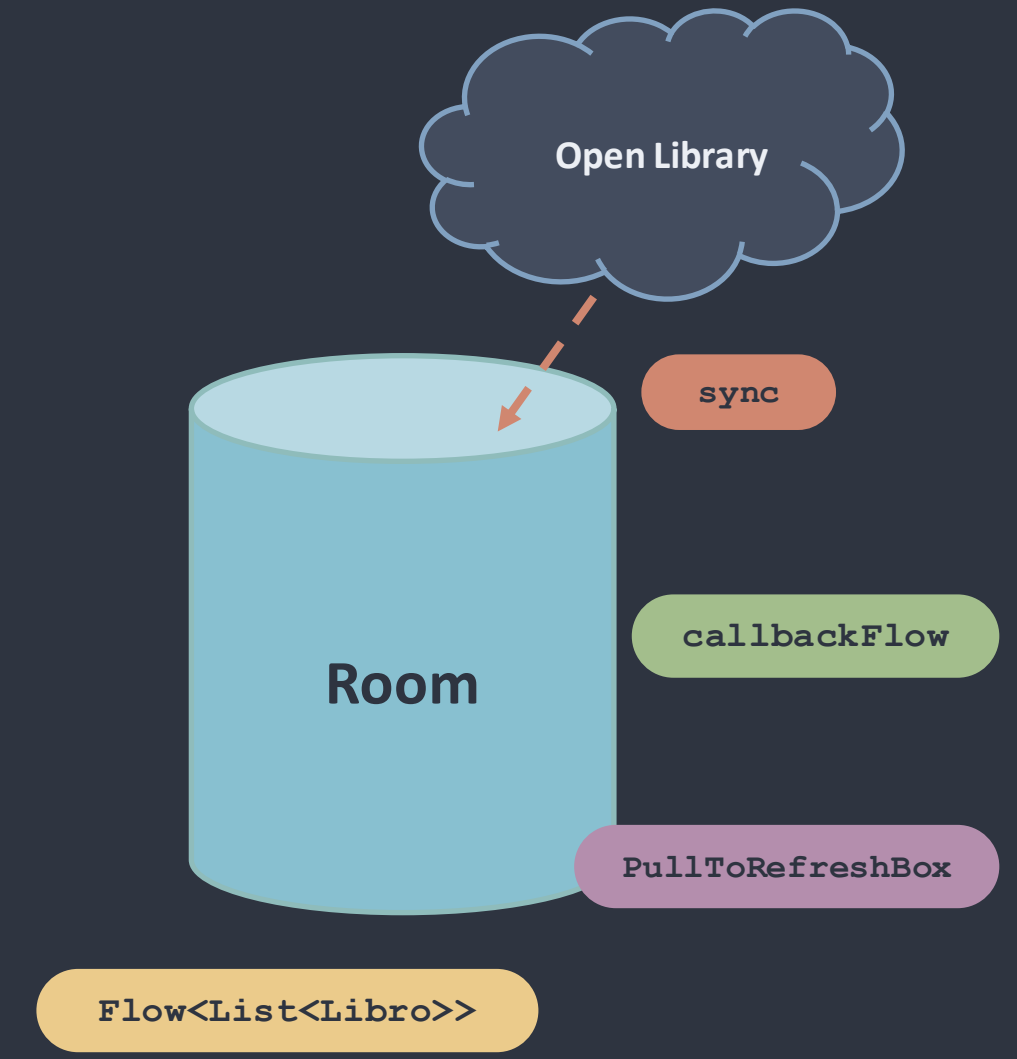
Offline-first

Integración de Room + Retrofit2: Room como única fuente de verdad

4 horas

AppDummy · parte de v7

RA2 · RA3



Hoja de ruta del tema

9 apartados

1 **El problema**
Dos fuentes, ninguna autoridad

2 **Offline-first**
Room, única fuente de verdad

3 **Entidad y DAO**
actualizado_en y UPDATE selectivo

4 **Repository**
sincronizarBiblioteca()

5 **ViewModel**
Estado de sync y búsqueda en SQL

6 **Interfaz**
Pull-to-refresh y aviso de conexión

7 **Conectividad**
Observarla con callbackFlow

8 **Caché HTTP**
Room ya es la caché

9 **Recorrido completo**
Qué pasa al abrir la app

¿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-a	Estructura de clases de la app
RA2-b	Ventanas, menús, alertas y controles
RA2-e	*Consultar enunciado en el RD 405/2023*
RA2-f	Almacenar y recuperar datos con persistencia
RA2-g	Pruebas de interacción usuario-app
RA2-i	Documentar el proceso de desarrollo
RA3-c	Conversión de datos entre formatos
RA3-d	Procesamiento de datos
RA3-e	Eventos, tipos y excepciones
RA3-h	Depurar y documentar

Punto de partida y cambios en el proyecto

`AppDummy v7`

0

bibliotecas nuevas

Todo llega de T4 y T5:

- Room
- Retrofit
- OkHttp
- Gson
- Coil
- Material 3

AndroidManifest.xml

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
    android:name="android.permission.ACCESS_NETWORK_STATE" />
```

`material3 ≥ 1.3.0`

Versión mínima necesaria para usar PullToRefreshBox.

`ACCESS_NETWORK_STATE`

Permiso normal: no se pide en tiempo de ejecución.

AppDatabase → version = 2

Nueva columna. Con fallbackToDestructiveMigration se pierden los datos de prueba.

01

La estrategia

Dos fuentes de datos, una sola verdad



Dos fuentes de datos y ninguna autoridad

Antipatrón

Room (T4)

`Flow<List<Libro>>`

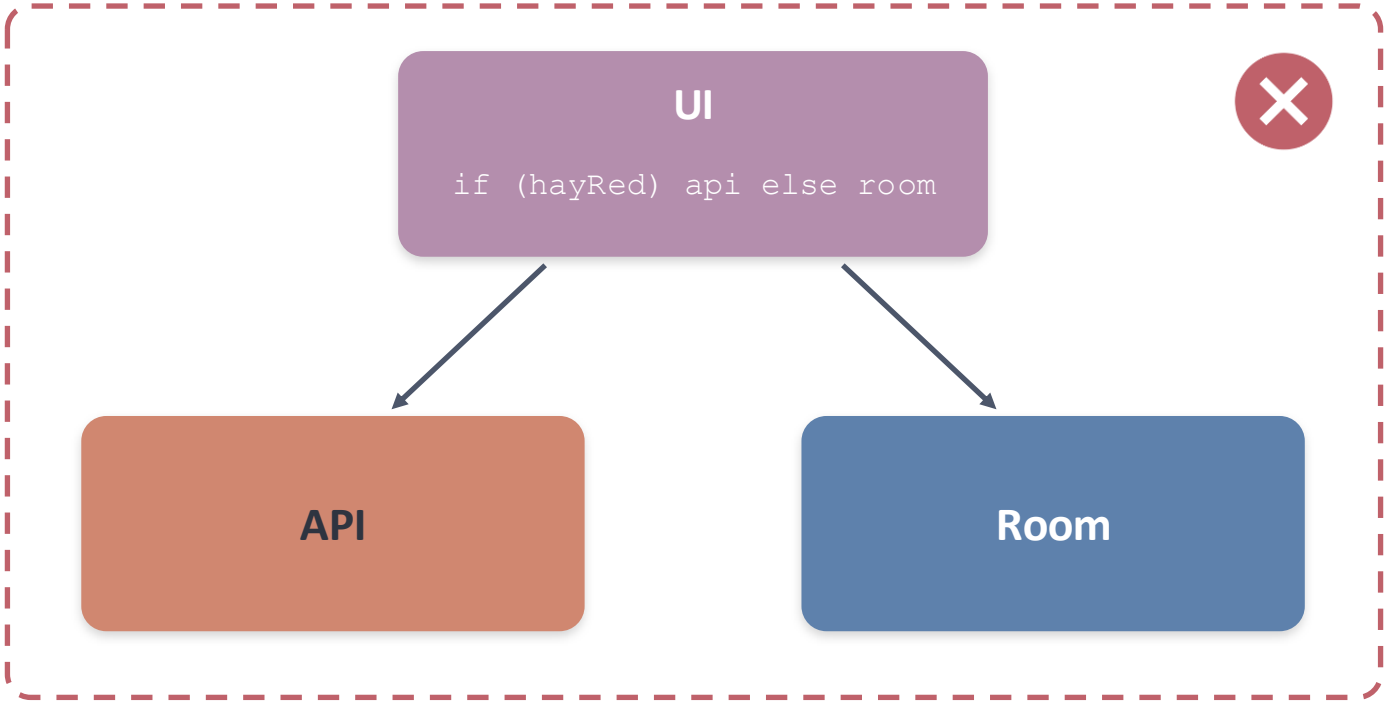
Guarda la biblioteca y avisa de cada cambio.

Retrofit2 (T5)

`Result<T>`

Consulta Open Library. Sus datos cambian y puede no haber red.

¿Quién manda?



Lógica duplicada

La UI decide el origen.

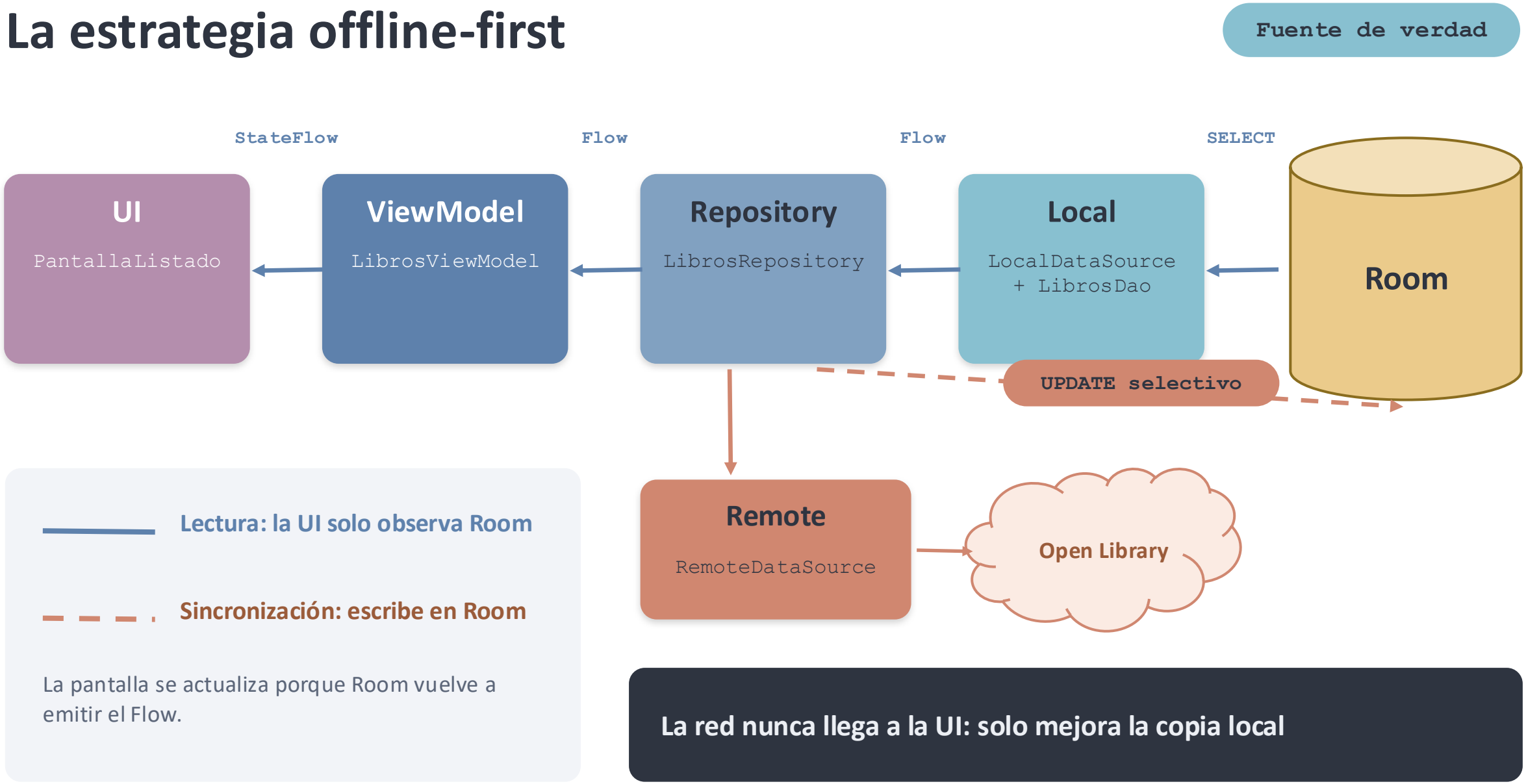
Imposible de probar

La decisión está en un composable.

Parpadeos

La lista cambia de origen al vuelo.

La estrategia offline-first



Tres principios

Offline-first



1

Room es la única fuente de verdad

La UI nunca recibe datos directamente de la red.



2

Sincronizar es un efecto lateral

sincronizarBiblioteca() escribe en Room y devuelve solo un resumen.



3

Un fallo de red no destruye nada

El estado local sigue intacto; el error se muestra en un Snackbar.

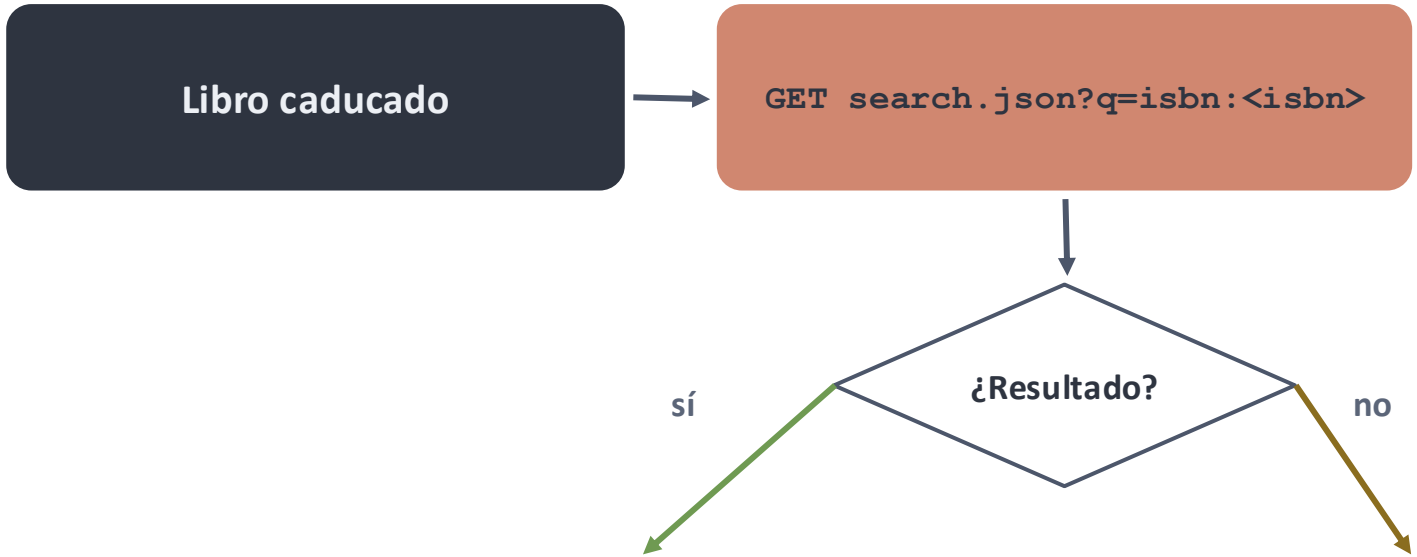


Offline-first no es una caché. En una caché la red manda y la copia local es un apoyo. Aquí es al revés: la copia local es la fuente principal y la red solo la mejora.

¿Qué significa sincronizar en AppDummy?

Open Library

Para cada libro guardado con ISBN caducado:



Actualiza metadatos

título, autor, año, ISBN, portada
+ fecha de comprobación

Solo anota la fecha

`marcarComprobado()`

Se refresca

- `titulo`
- `autor`
- `year`
- `isbn`
- `cover`
- `actualizado_en`

Se protege (solo local)

- `es_favorito`
- `leido`

1,1 s
de pausa entre peticiones

02

Capa de datos

Entidad, DAO y Repository



La marca de tiempo de sincronización

`@ColumnInfo`

data/model/Libro.kt

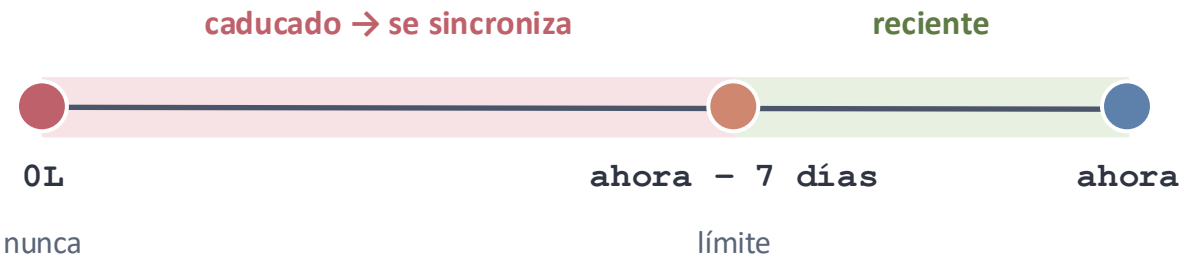
```kotlin
@Entity(tableName = "libros")
data class Libro(
    // … campos de T4 y T5 (titulo, autor,
    //   isbn, cover, esFavorito, leido…)

    // NUEVO en T6 · epoch en ms (0 = nunca)
    @ColumnInfo(name = "actualizado_en")
    val actualizadoEn: Long = 0
)
```

AppDatabase.kt

```kotlin
@Database(entities = [Libro::class],
    version = 2, exportSchema = true)
abstract class AppDatabase : RoomDatabase()
```

¿Qué libros están caducados?



¿Por qué Long y no Date?

Sin TypeConverter

SQLite guarda Long como INTEGER directamente.

Comparable en SQL

actualizado_en < :anteriorA funciona tal cual.

Sin zona horaria

Epoch en ms es igual en cualquier dispositivo.

El DAO: consultas de sincronización

@Dao

obtenerCaducados — `suspend · List<Libro>`

Candidatos con ISBN, del más antiguo al más reciente, con tope.

observarUltimaSincronizacion — `Flow<Long?>`

SELECT MAX(actualizado_en) para la barra superior.

actualizarDesdeRed — `@Query UPDATE`

Solo columnas de la API + actualizadoEn.

marcarComprobado — `@Query UPDATE`

Anota la fecha si la API no conoce el ISBN.

upsertConservandoFavorito — `@Transaction`

Inserta ignorando conflictos y actualiza las existentes.

LibrosDao.kt

```
@Query("SELECT * FROM libros " +
    "WHERE isbn != '' " +
    "AND actualizado_en < :anterioriorA " +
    "ORDER BY actualizado_en ASC " +
    "LIMIT :maximo")
suspend fun obtenerCaducados(
    anteriorA: Long,
    maximo: Int
): List<Libro>

@Query("SELECT MAX(actualizado_en) " +
    "FROM libros")
fun observarUltimaSincronizacion():
    Flow<Long?>
```

Tres reglas para escribir lo que llega de la red

UPDATE selectivo

UPDATE selectivo

Solo las columnas que vienen de la API. Nunca @Upsert.

Por el id local

Un libro creado a mano conserva su clave.

Siempre anotar la fecha

Todo libro comprobado, haya cambios o no.

Fila en Room

```
titulo = "Hail Mery"
esFavorito = true
leido = true
```

+ llega de la API:

```
titulo = "Hail Mary"
```

@Upsert

```
esFavorito = false
leido = false
```

actualizarDesdeRed

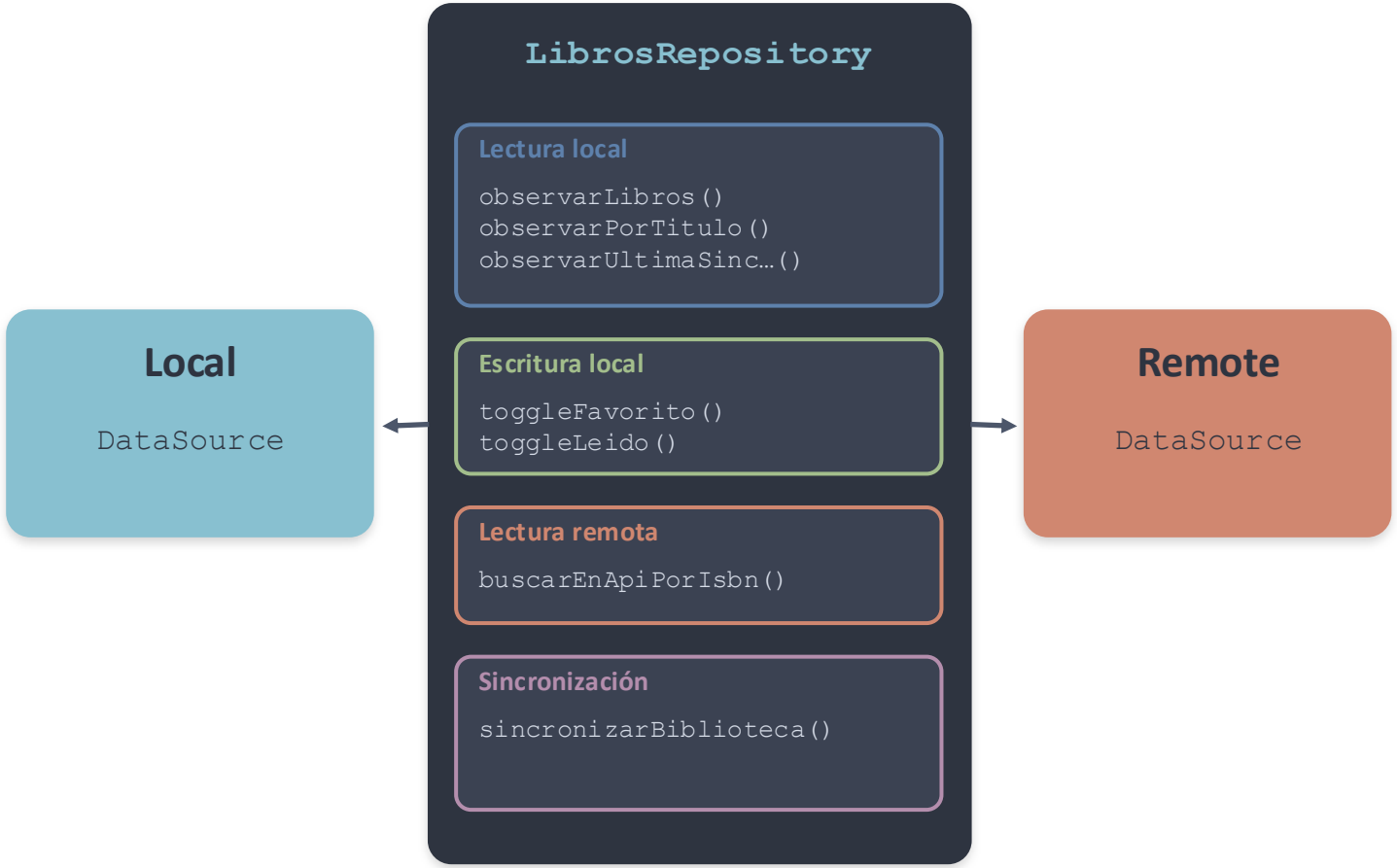
```
esFavorito = true
leido = true
```

LibrosDao.kt

```
@Query("UPDATE libros SET " +
  "titulo = :titulo, " +
  "autor = :autor, " +
  "year = :year, isbn = :isbn, " +
  "cover = :cover, " +
  "actualizado_en = " +
  ":actualizadoEn " +
  "WHERE id = :id")
suspend fun actualizarDesdeRed(…)
```

LibrosRepository conoce ambas fuentes

Repository



La UI y el ViewModel no saben que existe la red.

ResultadoSincronizacion.kt

```kotlin
data class ResultadoSincronizacion(
    val actualizados: Int = 0,
    val sinCambios: Int = 0,
    val error: String? = null
) {
    val comprobados: Int
        get() = actualizados + sinCambios
    val haFallado: Boolean
        get() = error != null
}
```

No contiene libros
Si los devolviera, la UI volvería a pintar datos de la red.

sincronizarBiblioteca(forzar) paso a paso

suspend

1 Calcular el límite

```
forzar ? ahora
: ahora − 7 días
```

2 Obtener pendientes

```
obtenerCaducados(
límite, 20)
```

3 Para cada libro

```
delay(1100)
antes de la siguiente
```

4 Consultar la API

```
buscarEnApiPorIsbn
(isbn)
```

siguiente libro

¿Respuesta?

Libro encontrado

`refrescarDesdeRed(id, datos, ahora)`

Si el ISBN llega vacío, se conserva el del usuario.

Sin resultados

`marcarComprobado(id, ahora)`

La API no conoce el ISBN: solo se anota la fecha.

Error de red

`return ResultadoSincronizacion(…)`

El primer fallo aborta y devuelve el resultado parcial.

El código y sus decisiones de diseño

Repository

LibrosRepository.kt (simplificado)

```
suspend fun sincronizarBiblioteca(
    forzar: Boolean = false
): ResultadoSincronizacion {
    val ahora = System.currentTimeMillis()
    val limite = if (forzar) ahora else ahora - SIETE_DIAS
    val pendientes = local.obtenerCaducados(limite, 20)
    var actualizados = 0
    var sinCambios = 0
    pendientes.forEachIndexed { i, libro ->
        if (i > 0) delay(1100)  // respetar la API
        val remoto = buscarEnApiPorIsbn(libro.isbn)
            .getOrElse { e ->    // return no local
                return ResultadoSincronizacion(
                    actualizados, sinCambios, e.message)
            }
        if (remoto == null) {
            local.marcarComprobado(libro.id, ahora); sinCambios++
        } else {
            local.refrescarDesdeRed(libro.id, remoto, ahora)
            actualizados++
        }
    }
    return ResultadoSincronizacion(actualizados, sinCambios)
}
```

delay() y no paralelo

Peticiones en serie para no recibir HTTP 429.

El primer fallo aborta

Sin red, seguir intentándolo no tiene sentido.

getOrElse + return

La lambda es inline: return sale de la función.

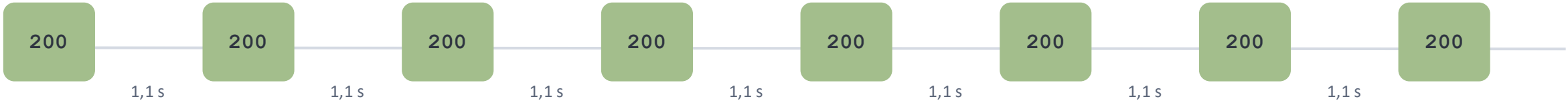
Devuelve un resumen

Los libros llegan a la UI desde Room.

Respetar la API y traducir los errores

HTTP 429

Con delay(1_100)



Sin delay (en paralelo)



Too Many Requests

ejecutarPeticion traduce los fallos a Result.failure:

IOException	403	404	429	5xx
Sin conexión	Acceso denegado	No encontrado	Demasiadas peticiones	Error del servidor

03

ViewModel e interfaz

Estado de sincronización, búsqueda en SQL y pull-to-refresh



LibrosViewModel: estados y eventos

ViewModel

Estado · StateFlow

Siempre tiene un valor; la UI lo dibuja.

`uiState`	lista (Cargando · Exito · Error)
`busqueda`	texto del buscador
`estaSincronizando`	indicador de refresco
`autores / autorSeleccionado`	chips de filtro
`hayConexion`	desde ObservadorConectividad
`ultimaSincronizacion`	fecha de la barra superior

Evento · SharedFlow

Se consume una sola vez: un Snackbar.

`replay = 0`	no se repite al rotar
`extraBufferCapacity = 1`	tryEmit no se pierde

LibrosViewModel.kt

```
private val _eventos = MutableSharedFlow<
    LibrosEvento>(replay = 0,
    extraBufferCapacity = 1)
val eventos = _eventos.asSharedFlow()

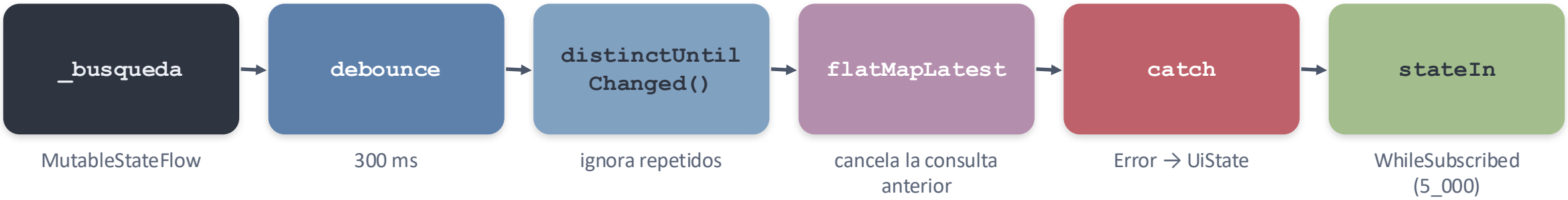
init { sincronizar(forzar = false) }
```

La búsqueda se resuelve en SQL

`flatMapLatest`

El usuario escribe…

`"h"` `"ha"` `"hai"` **`"hail"`** *tras 300 ms sin teclear solo pasa "hail"*



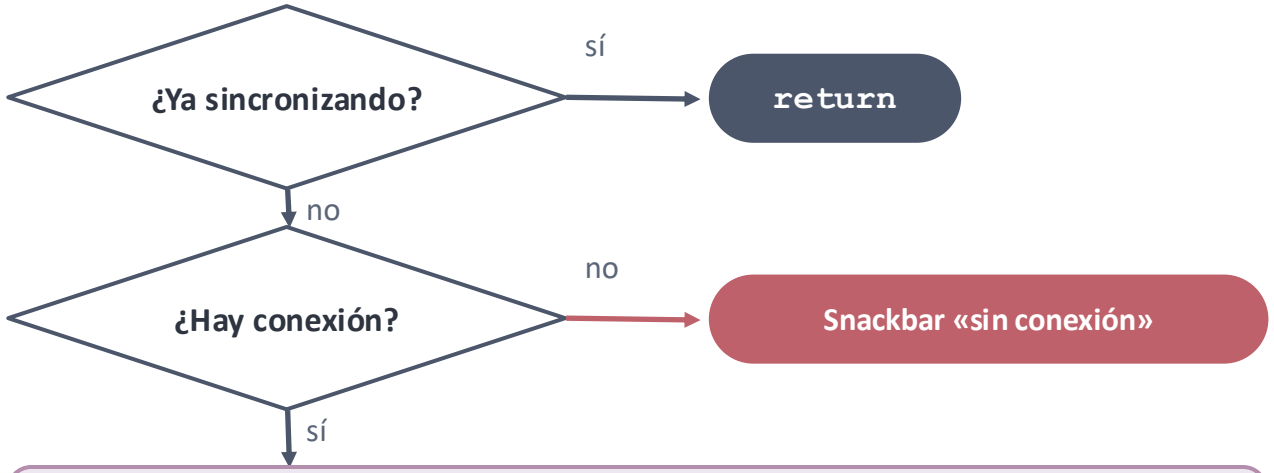

```
LibrosViewModel.kt · uiState
.flatMapLatest { texto ->
    if (texto.isBlank()) repository.observarLibros()
    else repository.observarPorTitulo(texto)
}
```

Título → Room (aprovecha el índice)

Autor → UI (filtro con chips)

sincronizar(forzar): guardas y try/finally

ViewModel



viewModelScope.launch

```
estaSincronizando = true

try  → repository.sincronizarBiblioteca(forzar)

     → evento con mensajeDe(resultado)

finally → estaSincronizando = false
```

finally siempre se ejecuta

Aunque falle o se cancele, el indicador de refresco se apaga.

forzar = false al abrir

init sincroniza solo lo caducado; forzar = true lo revisa todo.

mensajeDe() es pura

Recibe un ResultadoSincronizacion y devuelve texto: fácil de probar con un test unitario.

PantallaListado de un vistazo

Compose

TopAppBar
última sync + botón

AnimatedVisibility
BannerSinConexion()

OutlinedTextField
búsqueda

SnackbarHost
eventos del ViewModel



LazyRow + FilterChip

PullToRefreshBox
isRefreshing · onRefresh

LazyColumn
lista de Room

Solo el banner y el botón dependen de hayConexion.

La lista nunca cambia de origen.

PullToRefreshBox y consumo de eventos

Material 3

PantallaListado.kt

```
PullToRefreshBox(
    isRefreshing = estaSincronizando,
    onRefresh = { viewModel.sincronizar(forzar = true) },
    modifier = Modifier.fillMaxSize()
) {
    LazyColumn(Modifier.fillMaxSize()) {
        if (libros.isEmpty()) item { EstadoVacio() }
        else items(libros, key = { it.id }) { libro ->
            // tarjeta del libro
        }
    }
}
```

PantallaListado.kt

```
LaunchedEffect(Unit) {
    viewModel.eventos.collect { evento ->
        snackbarHostState.showSnackbar(evento.mensaje)
    }
}
```



El contenido debe ser desplazable

Si no hay scroll, el gesto de arrastrar no se detecta.



EstadoVacio dentro de LazyColumn

Una lista vacía sigue permitiendo tirar para refrescar.



LaunchedEffect(Unit)

Se recolecta una vez mientras la pantalla está en composición.

04

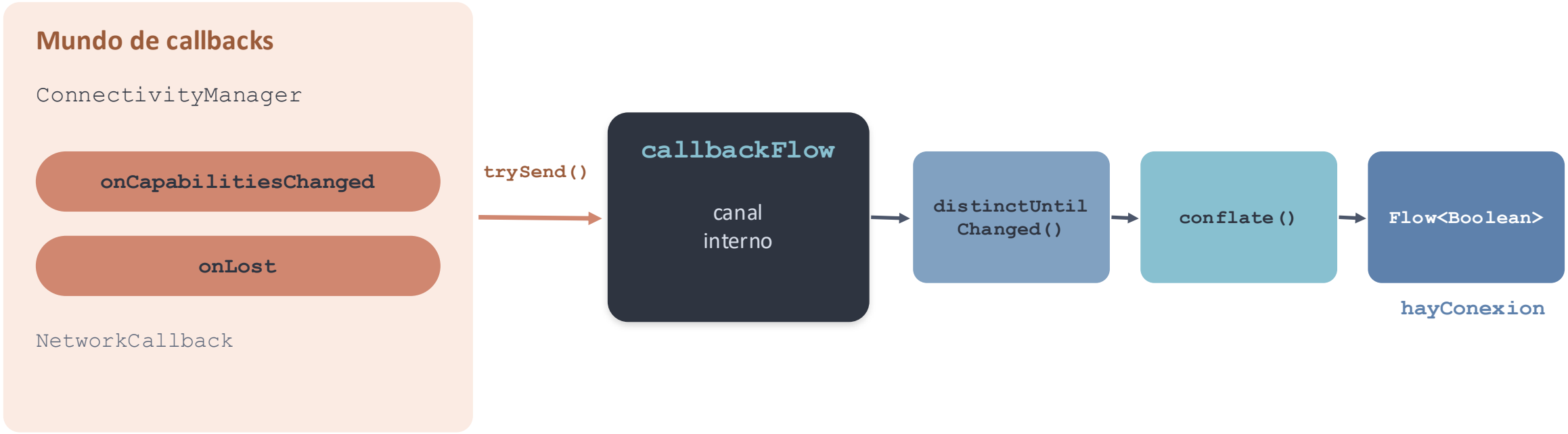
Conectividad y caché

callbackFlow, ConnectivityManager y Coil

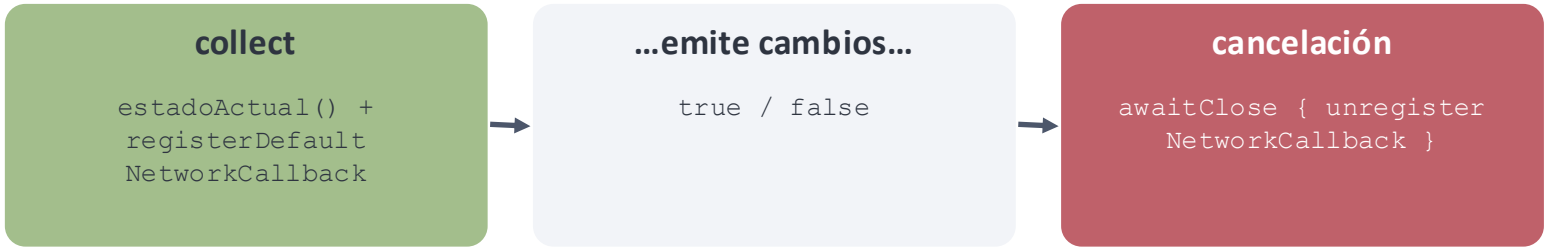


De callbacks a Flow con callbackFlow

`callbackFlow`



Ciclo de vida del callback



`NET_CAPABILITY_VALIDATED`

Wi-Fi conectada sin Internet real no cuenta como conexión.

ObservadorConectividad

utils/

```
utils/ObservadorConectividad.kt (simplificado)

class ObservadorConectividad(context: Context) {
    private val cm = context
        .getSystemService(ConnectivityManager::class.java)

    val hayConexion: Flow<Boolean> = callbackFlow {
        val callback = object : ConnectivityManager.NetworkCallback() {
            override fun onCapabilitiesChanged(
                network: Network, caps: NetworkCapabilities) {
                trySend(caps.hasCapability(
                    NetworkCapabilities.NET_CAPABILITY_VALIDATED))
            }
            override fun onLost(network: Network) { trySend(false) }
        }
        trySend(estadoActual())
        cm.registerDefaultNetworkCallback(callback)
        awaitClose { cm.unregisterNetworkCallback(callback) }
    }.distinctUntilChanged().conflate()

    private fun estadoActual(): Boolean {
        val caps = cm.getNetworkCapabilities(cm.activeNetwork)
        return caps?.hasCapability(
            NetworkCapabilities.NET_CAPABILITY_VALIDATED) == true
    }
}
```

`callbackFlow { }`

Constructor del Flow

`trySend()`

Envía sin suspender

`awaitClose { }`

Libera al cancelar

`applicationContext`

En AppContainer: sin fugas

Prueba: modo avión ✈

Caché HTTP: Room ya es la caché

Coil 3

JSON de search.json → Room

Ya está en la tabla. Una caché HTTP sería redundante y podría falsear actualizado_en.

Carátulas → Coil

Binarios grandes de covers.openlibrary.org. Coil 3 activa por defecto la caché de memoria y de disco.




```
AppDummyApplication.kt (simplificado)

class AppDummyApplication : Application(),
    SingletonImageLoader.Factory {
  override fun newImageLoader(
      context: PlatformContext) =
    ImageLoader.Builder(context)
      .components {
        add(OkHttpNetworkFetcherFactory(
          callFactory = { container.okHttpClient }))
      }
      .crossfade(true)
      .build()
}
```

05

Todo junto

Recorrido completo, estructura y práctica guiada



Recorrido completo al abrir la app

Flujo



Sin red: los pasos 1-4 funcionan igual. Solo la sincronización termina con un aviso.

Estructura de paquetes al final de T6

AppDummy

```
com.ejemplo.appdummy/
├─ AppDummyApplication.kt  (ImageLoader)
├─ data/
│  ├─ datasource/local/
│  │  ├─ AppDatabase.kt (v2) · LibrosDao.kt
│  │  ├─ LocalDataSource.kt · Converters.kt
│  ├─ datasource/remote/  (DTOs, Retrofit)
│  ├─ di/AppContainer.kt
│  ├─ model/Libro.kt
│  ├─ repository/LibrosRepository.kt
│  ├─ repository/ResultadoSincronizacion.kt
├─ screens/listado/  (ViewModel, Pantalla, UiState, Evento)
├─ utils/ObservadorConectividad.kt
```

Nuevo en T6

Resultado de sync y observador de red

Modificado

Casi todo está en data/

Sin cambios

Capa remota de T5

Los cambios se concentran en la capa data/: la interfaz apenas cambia.

Desarrollo práctico guiado

12 pasos

1. Columna, versión 2 y permiso
2. Ampliar DAO y LocalDataSource
3. sincronizarBiblioteca()
4. Quitar delay() → ver HTTP 429
5. Búsqueda con flatMapLatest
6. ObservadorConectividad
7. PullToRefreshBox
8. App Inspection: favoritos intactos
9. Usar dao.upsert → favoritos perdidos
10. Libro manual con ISBN-13
11. Toda la app en modo avión
12. Test unitario de mensajeDe()

Construir · Romper y observar · Verificar

Ideas clave

Room manda

La red solo actualiza Room; la UI solo observa Room.

Sync = efecto lateral

Escribe y devuelve un resumen; la UI se recompone sola.

UPDATE selectivo

Protege favoritos y leídos frente a @Upsert.

Respeta la API

Pausas de 1,1 s y tope de 20 libros por sync.

Búsqueda en SQL

debounce + flatMapLatest sobre observarPorTitulo().

callbackFlow

Conectividad observable y sin fugas con awaitClose.

Referencias

Fuentes

Capa de datos · Guía de arquitectura de Android
developer.android.com/topic/architecture/data-layer

Crear una app offline-first
developer.android.com/topic/architecture/data-layer/offline-first

Pull to refresh en Compose
developer.android.com/develop/ui/compose/components/pull-to-refresh

Leer el estado de la red
developer.android.com/training/basics/network-ops/reading-network-state

callbackFlow · kotlinx.coroutines
kotlinlang.org/api/kotlinx.coroutines

Open Library · API para desarrolladores
openlibrary.org/developers/api

Coil · Red y caché
coil-kt.github.io/coil/network

Tema 6. Integración Room + Retrofit2 y offline-first (apuntes)
documentation.javiercarrasco.es/pmdm/B3/B3-T6_Integracion_OfflineFirst