TEMA 3

Temas Avanzados de Android

Programación de Dispositivos Móviles

Datos y BD | Mapas y GPS | Sensores | Multimedia | Widgets | Servicios Web | Bluetooth

CONTENIDOS

Tema 3

01 **Datos y Base de Datos**
SharedPrefs, DataStore, Room, Firebase

02 **Mapas y GPS**
Google Maps, FusedLocation, Geofencing

03 **Sensores**
Acelerómetro, giroscopio, casos de uso

04 **Multimedia**
Audio, vídeo, ExoPlayer, CameraX

05 **Widgets**
AppWidgets, RemoteViews, actualización

06 **Servicios Web**
Retrofit, REST, JWT, JSON/XML

07 **Bluetooth**
BT clásico, BLE, GATT, IoT

Almacenamiento Local: SharedPreferences y DataStore

SharedPreferences

Clásico — Simple

- Pares clave-valor
- String, Int, Boolean, Float
- Modo privado (MODE_PRIVATE)
- Síncrono (commit) o asíncrono (apply)
- Aún válido para preferencias simples

Deprecated para uso complejo

DataStore

Moderno — Recomendado

- Basado en corrutinas y Flow
- Preferences DataStore → clave-valor
- Proto DataStore → tipos estructurados
- Manejo de errores con Result
- Seguro para operaciones asíncronas

Reemplaza SharedPreferences

Archivos y SQLite

Avanzado — Control total

- Internal Storage → solo la app
- External Storage → compartido
- SQLite nativo → con DatabaseHelper
- openOrCreateDatabase()
- Consultas SQL directas

Room es la abstracción recomendada

Room Database: ORM Oficial de Android

Arquitectura Room

UI / ViewModel
Observa LiveData / Flow

▼

Repository
Capa de abstracción

▼

DAO
@Query @Insert @Delete

▼

Room Database
@Database — SQLite

▼

SQLite
Motor de base de datos

Componentes Room

```
// ENTITY
@Entity(tableName = "usuarios")
data class Usuario(
    @PrimaryKey(autoGenerate = true)
    val id: Int = 0,
    val nombre: String,
    val email: String
)

// DAO
@Dao
interface UsuarioDao {
    @Query("SELECT * FROM usuarios")
    fun getAll(): Flow<List<Usuario>>
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(u: Usuario)
    @Delete
    suspend fun delete(u: Usuario)
}

// DATABASE
@Database(entities = [Usuario::class], version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun usuarioDao(): UsuarioDao
    companion object {
```

Firebase: Base de Datos en la Nube

RT Realtime Database

- ✓ JSON en tiempo real
- ✓ Sincronización offline automática
- ✓ Ideal para chats y presencia
- ✓ Estructura de árbol JSON simple

```
val db = Firebase.database.reference
db.child("usuarios").child(uid)
  .setValue(userData)
  .addOnSuccessListener { /* OK */ }

// Escuchar cambios en tiempo real
db.child("mensajes")
  .addValueEventListener(object: ValueEventListener
{
    override fun onDataChange(snapshot:
DataSnapshot) {
      val data =
snapshot.getValue(Mensaje::class.java)
    }
    override fun onCancelled(error: DatabaseError)
```

FS Cloud Firestore

- ✓ Colecciones y documentos
- ✓ Consultas avanzadas con filtros
- ✓ Escalabilidad superior
- ✓ Ideal para apps de producción

```
val db = Firebase.firestore

// Escribir documento
db.collection("usuarios").document(uid)
  .set(hashMapOf("nombre" to "Juan", "edad" to 30))

// Leer con Flow
db.collection("productos")
  .whereEqualTo("activo", true)
  .orderBy("precio")
  .addSnapshotListener { snap, error ->
    val lista =
snap?.toObjects(Producto::class.java)
  }
```

Google Maps y Geolocalización

Google Maps SDK

📍 Marcadores (Markers)

addMarker() con latLng, title e icono personalizado

🗺️ Tipos de mapa

MAP_TYPE_NORMAL, SATELLITE, TERRAIN, HYBRID

✏️ Polilíneas y polígonos

Rutas, áreas y formas vectoriales sobre el mapa

📷 Camera y zoom

CameraUpdateFactory.newLatLngZoom(latlng, zoom)

🔘 Círculos y shapes

CircleOptions con centro, radio, color y stroke

FusedLocationProviderClient

```
// Permiso: ACCESS_FINE_LOCATION
val fusedClient = LocationServices
    .getFusedLocationProviderClient(this)

// Última ubicación conocida
fusedClient.lastLocation
    .addOnSuccessListener { location ->
        location?.let {
            val lat = it.latitude
            val lng = it.longitude
        }
    }

// Actualizaciones periódicas
val request = LocationRequest.Builder(
    Priority.PRIORITY_HIGH_ACCURACY, 5000)
    .setMinUpdateIntervalMillis(2000)
    .build()

fusedClient.requestLocationUpdates(
    request,
    locationCallback,
    Looper.getMainLooper()
)
```

Sensores Android: Tipos y Acceso

M Sensores de Movimiento

- Acelerómetro
- Giroscopio
- Gravedad
- Rotación lineal

Uso:

Juegos, gestos, orientación

P Sensores de Posición

- Magnetómetro
- Proximidad
- GPS (no sensor)

Uso:

Brújula, detectar llamadas

E Sensores de Entorno

- Luz ambiental
- Barómetro
- Temperatura
- Humedad

Uso:

Brillo auto, altitud, clima

B Sensores de Biométrico

- Ritmo cardíaco
- SpO2
- Huella dactilar

Uso:

Salud, seguridad biométrica

```
Acceso: SensorManager → getDefaultSensor(TYPE) → registerListener(callback, sensor, SENSOR_DELAY_NORMAL) → onSensorChanged(event)
```

Implementación del SensorEventListener

Acceso al sensor

```
class SensorActivity : AppCompatActivity(),
    SensorEventListener {

  private lateinit var sensorManager: SensorManager
  private var accelerometer: Sensor? = null

  override fun onCreate(b: Bundle?) {
    super.onCreate(b)
    sensorManager = getSystemService(
        Context.SENSOR_SERVICE) as SensorManager
    accelerometer = sensorManager
        .getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
  }

  override fun onResume() {
    super.onResume()
    sensorManager.registerListener(
        this, accelerometer,
        SensorManager.SENSOR_DELAY_NORMAL)
  }

  override fun onPause() {
    super.onPause()
    sensorManager.unregisterListener(this)
  }
```

Casos de uso prácticos

SC — **Detector de agitación**

Calcular magnitud del vector aceleración > umbral → evento

SP — **Contador de pasos**

TYPE_STEP_COUNTER
Acumula pasos desde reinicio

MG — **Brújula digital**

TYPE_MAGNETIC_FIELD + getOrientation() → azimuth

RV — **Orientación pantalla**

TYPE_ROTATION_VECTOR
Cuaternión → matriz rotación

Audio, Vídeo y Cámara

AUDIO

MediaPlayer

Audio básico

- Reproducir audio local o URL
- setDataSource + prepare() + start()
- Control: pause, stop, seekTo
- Liberar con release() en onDestroy

VIDEO

ExoPlayer

Vídeo y streaming (recomendado)

- HLS, DASH, SmoothStreaming
- Altamente personalizable
- PlayerView → UI integrada
- Media3 (AndroidX)

CAMARA

CameraX

Cámara moderna

- API simplificada y estable
- Use Cases: Preview, Image Capture, Video Capture, Analysis
- Compatible con todos los dispositivos
- ProcessCameraProvider

PERMISOS

Permisos

Necesarios siempre

- CAMERA → cámara
- RECORD_AUDIO → micrófono
- READ_MEDIA_IMAGES (API 33+)
- Solicitar en runtime con registerForActivityResult

ExoPlayer y CameraX — Código Esencial

ExoPlayer (Media3)

```
// Dependencia
// implementation("androidx.media3:media3-
exoplayer:1.x")

// Layout: <PlayerView
android:id="@+id/player_view"/>

private var player: ExoPlayer? = null

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val player = ExoPlayer.Builder(this).build()
    binding.playerView.player = player

    val mediaItem = MediaItem.fromUri(
        "https://example.com/video.mp4")
    player.setMediaItem(mediaItem)
    player.prepare()
    player.play()
}

override fun onDestroy() {
    super.onDestroy()
    player?.release()
    player = null
}
```

CameraX — Preview + Captura

```
// Dependencia
// implementation("androidx.camera:camera-camera2:1.x")

private lateinit var cameraProvider:
ProcessCameraProvider
private lateinit var imageCapture: ImageCapture

fun startCamera() {
    val cameraProviderFuture =
        ProcessCameraProvider.getInstance(this)

    cameraProviderFuture.addListener({
        cameraProvider = cameraProviderFuture.get()
        val preview = Preview.Builder().build().also {
            it.setSurfaceProvider(
                binding.viewFinder.surfaceProvider)
        }
        imageCapture = ImageCapture.Builder().build()

        cameraProvider.bindToLifecycle(
            this,
            CameraSelector.DEFAULT_BACK_CAMERA,
            preview, imageCapture)
    }, ContextCompat.getMainExecutor(this))
}
```

AppWidgets: Componentes en el Escritorio

¿Qué es un AppWidget?

WD — **Widget Layout XML**

Layout con RemoteViews
(vistas limitadas disponibles)

WI — **Widget Info XML**

Tamaño, período actualización,
vista de preview

WP — **AppWidgetProvider**

BroadcastReceiver → onUpdate()
onEnabled(), onDisabled()

WM — **AndroidManifest**

Receptor con action
ACTION_APPWIDGET_UPDATE

AppWidgetProvider — Código

```
// res/xml/widget_info.xml
<appwidget-provider
    android:minWidth="110dp"
    android:minHeight="40dp"
    android:updatePeriodMillis="1800000"
    android:previewImage="@drawable/widget_preview"
    android:initialLayout="@layout/widget_layout"/>

// AppWidgetProvider
class MiWidget : AppWidgetProvider() {

    override fun onUpdate(
        context: Context,
        manager: AppWidgetManager,
        ids: IntArray
    ) {
        ids.forEach { appWidgetId ->
            val views = RemoteViews(
                context.packageName,
                R.layout.widget_layout)

            // Actualizar texto
            views.setTextViewText(
                R.id.tv_widget, "Actualizado!")
```

Retrofit: Consumo de APIs REST

Stack recomendado

ViewModel
Llama al Repository

▼

Repository
Abstrae la fuente

▼

Retrofit
HTTP Client

▼

OkHttp
Interceptores, logs

▼

Gson / Moshi
Serialización JSON

▼

API REST
Servidor remoto

Configuración y uso de Retrofit

```
// Interface de la API
interface ApiService {
    @GET("users/{id}")
    suspend fun getUser(@Path("id") id: Int): Response<User>
    @POST("users")
    suspend fun createUser(@Body user: User): Response<User>
    @GET("products")
    suspend fun getProducts(
        @Query("page") page: Int,
        @Query("category") cat: String
    ): Response<List<Product>>
}

// Singleton Retrofit
object RetrofitClient {
    private const val BASE_URL = "https://api.example.com/"

    val instance: ApiService by lazy {
        val logging = HttpLoggingInterceptor()
            .apply { level = HttpLoggingInterceptor.Level.BODY }
        val okHttp = OkHttpClient.Builder()
            .addInterceptor(logging)
            .addInterceptor { chain ->
                val req = chain.request().newBuilder()
                    .addHeader("Authorization", "Bearer $token")
```

Bluetooth Clásico y BLE

BT Bluetooth Clásico

- Transferencia de datos alta velocidad
- Rango: hasta 100 metros
- Audio (A2DP, HSP, HFP)
- SPP (Serial Port Profile)
- Conexión más lenta de establecer

LE Bluetooth Low Energy (BLE)

- Consumo de energía muy bajo
- Ideal para sensores e IoT
- Arquitectura GATT (Server/Client)
- Profiles, Services, Characteristics
- Advertising y scanning continuo

BLE: Scan + GATT Connection

```
// Permisos: BLUETOOTH_SCAN, BLUETOOTH_CONNECT (API 31+), ACCESS_FINE_LOCATION

// Scanner BLE
val scanner = bluetoothAdapter.bluetoothLeScanner
val scanCallback = object : ScanCallback() {
    override fun onScanResult(callbackType: Int, result: ScanResult) {
        val device = result.device    // BluetoothDevice
        val rssi = result.rssi        // Señal (dBm)
        Log.d("BLE", "Encontrado: ${device.name} / ${device.address}")
    }
}
```

Resumen del Tema 3

Puntos Clave

01	Datos: Room (ORM SQLite), DataStore (preferencias), Firebase (cloud)
02	Mapas: FusedLocationProvider, Google Maps SDK, Geofencing para zonas
03	Sensores: SensorManager + SensorEventListener → movimiento, posición, entorno
04	Multimedia: ExoPlayer (vídeo/streaming), CameraX (cámara), permisos runtime
05	Widgets: AppWidgetProvider + RemoteViews → contenido en el escritorio
06	Servicios Web: Retrofit + OkHttp + Gson → consumo de APIs REST con corrutinas
07	Bluetooth: BT clásico para audio/datos, BLE con GATT para IoT y sensores

ACTIVIDADES PRÁCTICAS

Ejercicios del Tema 3

1 Room + ViewModel

CRUD completo con Room, ViewModel y Flow. Mostrar lista con RecyclerView.

2 Mapa + Ubicación

Mostrar mapa con Google Maps, obtener ubicación actual y añadir marcador.

3 Detector de agitación

Implementar Shake Detector con SensorEventListener. Vibrar al detectar.

4 Reproductor de vídeo

Integrar ExoPlayer con una URL de streaming HLS. Controles play/pause/seek.

5 Widget de reloj

AppWidget que muestra la hora actual. Se actualiza cada 30 minutos.

6 Consulta REST + Room

Descargar lista de usuarios con Retrofit y almacenarlos en Room (offline-first).

7 BLE Scanner

Escanear dispositivos BLE cercanos y mostrarlos en una lista con RSSI.