

Informe semana 8

**MÓDULO: TALLER DE APLICACIONES MÓVILES**

**SEMANA: Semana 8**

Docente: IVAN AYALA

Estudiante: Andres Lagos

Índice

[Introducción 3](#_Toc210570186)

[Desarrollo 4](#_Toc210570187)

[Evidencias de la Aplicación 6](#_Toc210570188)

[Código Fuente 10](#_Toc210570189)

[MainActivity.kt 10](#_Toc210570190)

[MenuActivity.kt 17](#_Toc210570191)

[LoginActivity.kt 21](#_Toc210570192)

[SettingsActivity.kt 23](#_Toc210570193)

[Ranges.kt 25](#_Toc210570194)

[Codigo tinkercad 27](#_Toc210570195)

[Conclusión 27](#_Toc210570196)

[Anexo 28](#_Toc210570197)

# Introducción

Este informe presenta la aplicación desarrollada en Android Studio para el módulo Taller de Aplicaciones Móviles. La app integra cálculos matemáticos, monitoreo de temperatura desde Firebase Realtime Database, conversión de grados a radianes, cálculo de despachos y registro de ubicación mediante GPS. Además, emite una alarma sonora y vibratoria cuando la temperatura supera los rangos definidos localmente por el usuario.

# Desarrollo

1. **Objetivo del desarrollo**

Se implementó un sistema de monitoreo de temperatura en la aplicación Android desarrollada en semanas anteriores.

Este módulo permite leer valores en tiempo real desde Firebase Realtime Database, convertirlos a grados Celsius y activar una alarma sonora y vibratoria cuando los valores superan el rango definido por el usuario administrador.

1. **Lectura de temperatura desde Firebase**

Se utilizó el servicio Firebase Realtime Database para obtener el valor del campo temperatura alojado en el nodo sensors/truck1/.

La aplicación escucha los cambios de este valor mediante un ValueEventListener, lo que permite actualizar de forma instantánea la temperatura mostrada en pantalla.

Este valor se recibe en grados Fahrenheit y se transforma automáticamente a grados Celsius mediante una conversión matemática implementada en MainActivity.kt.

1. **Configuración de rangos en base de datos local**

Se implementó una base de datos local utilizando Room para almacenar los rangos mínimo y máximo de temperatura configurados por el usuario.

Esta configuración se gestiona desde la pantalla SettingsActivity, la cual permite modificar los valores y guardarlos localmente para que el sistema de monitoreo pueda utilizarlos sin conexión a internet.

1. **Sistema de alerta sonora y vibratoria**

Cuando el valor leído desde Firebase supera el rango permitido, el sistema activa una vibración del dispositivo y un sonido de alarma pregrabado.

Esto se logró utilizando las clases Vibrator y MediaPlayer, integradas en la clase principal MainActivity.kt.

La alarma se detiene automáticamente al regresar el valor de temperatura a un rango aceptable.

1. **Pruebas de funcionamiento**

Se realizaron pruebas en el emulador de Android Studio para verificar la lectura y actualización automática de datos desde Firebase.

Además, se probó la funcionalidad de los rangos configurables y la activación del sistema de alerta, confirmando su correcto comportamiento.

Finalmente, se validó la persistencia de los valores locales en la base de datos Room, asegurando que el sistema mantuviera los límites incluso tras reiniciar la aplicación.

1. **Resultados obtenidos**

La aplicación logró un monitoreo estable de temperatura en tiempo real, mostrando los datos de forma visual y con retroalimentación inmediata al usuario.

Con esta integración, se cumple el objetivo de supervisar la cadena de frío de alimentos, alertando al operador ante cualquier variación fuera del rango definido.

1. **Prototipo de emulación en Tinkercad**

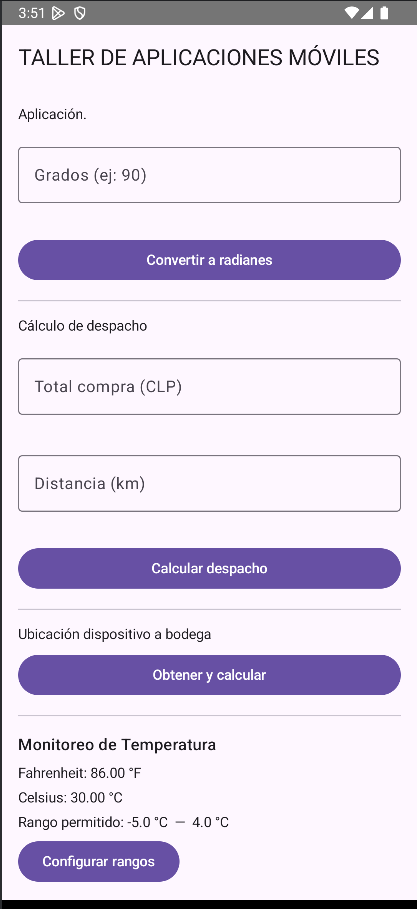
Para complementar el sistema de monitoreo desarrollado en Android Studio, se creó un prototipo virtual en la plataforma Tinkercad utilizando un Arduino UNO y un sensor de temperatura TMP36.

El circuito permite obtener lecturas analógicas del sensor y convertirlas a grados Celsius y Fahrenheit mediante el código cargado en el microcontrolador.

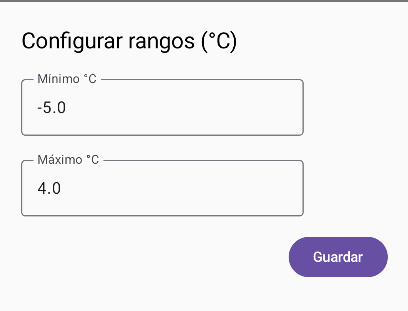
El programa en Arduino emula el envío de datos hacia Firebase, mostrando en el monitor serial una estructura tipo JSON con los valores de temperatura en °F y el tiempo de actualización.

De esta manera, se simula la captura y transmisión del dato que la aplicación móvil recibe para su monitoreo en tiempo real.

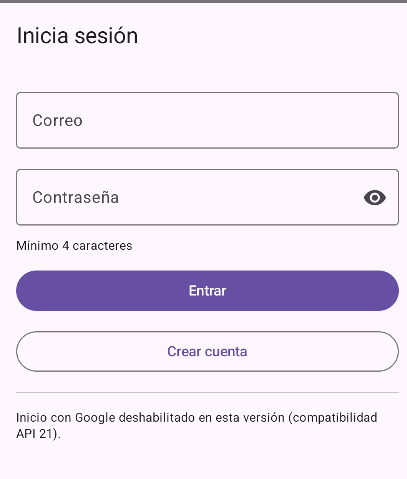
# Evidencias de la Aplicación



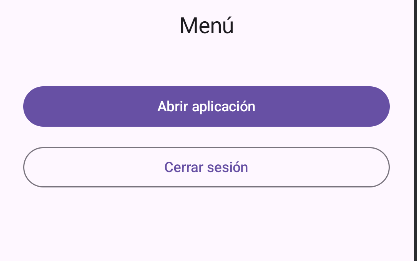
Pantalla principal con monitoreo de temperatura.



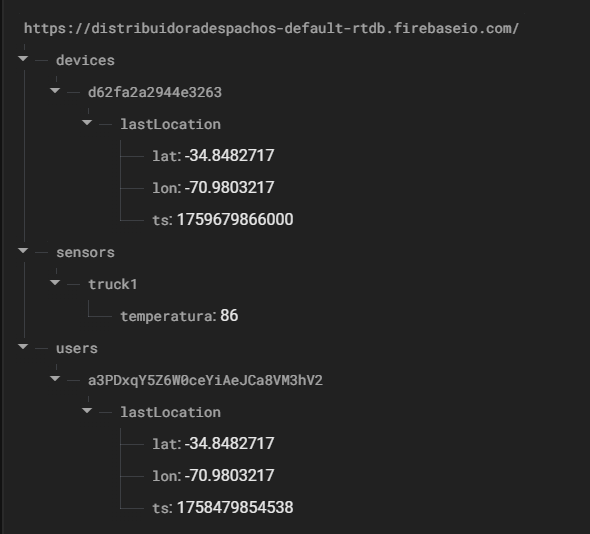
Pantalla para configurar los rangos de temperatura.



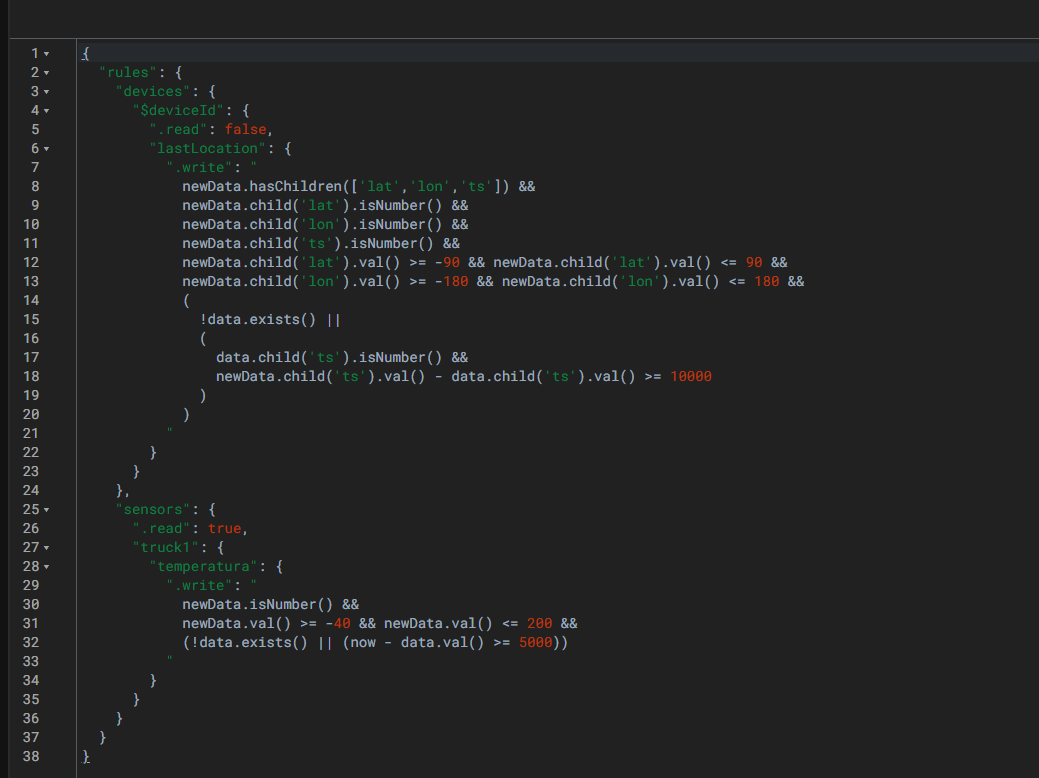
Pantalla de inicio de sesión de usuario.



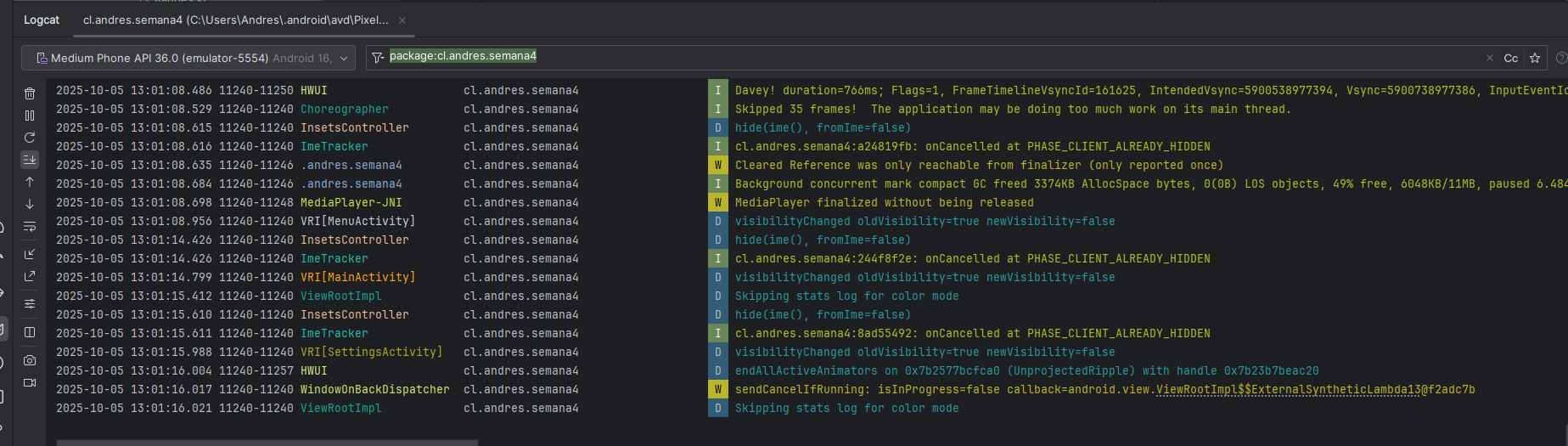
Menú principal con opciones de abrir aplicación y cerrar sesión.



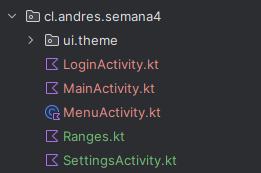
Visualización en Firebase Realtime Database de las ubicaciones y temperatura.



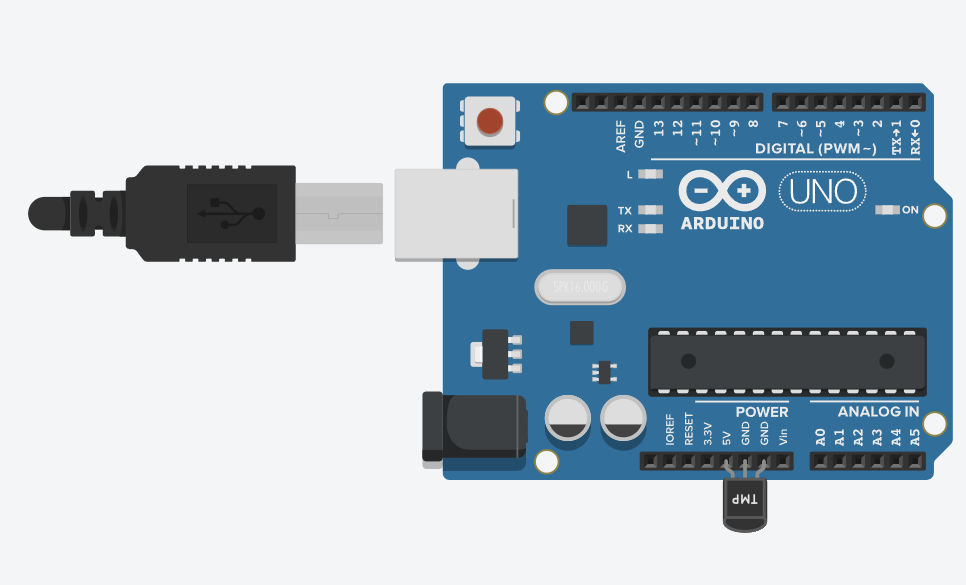
Reglas configuradas en Firebase para control de acceso y validación.



Consola de Logcat mostrando la ejecución correcta sin errores.



Estructura de archivos Kotlin en el proyecto.



**Prototipo de emulación en Tinkercad**

# Código Fuente

A continuación, se presenta el código completo de los principales archivos Kotlin del proyecto:

## MainActivity.kt

package cl.andres.semana4  
  
import android.Manifest  
import android.content.Context  
import android.content.Intent  
import android.content.pm.PackageManager  
import android.media.MediaPlayer  
import android.os.Build  
import android.os.Bundle  
import android.os.VibrationEffect  
import android.os.Vibrator  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.rememberLauncherForActivityResult  
import androidx.activity.compose.setContent  
import androidx.activity.result.contract.ActivityResultContracts  
import androidx.activity.viewModels  
import androidx.compose.foundation.layout.\*  
import androidx.compose.foundation.text.KeyboardOptions  
import androidx.compose.material3.\*  
import androidx.compose.runtime.\*  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.platform.*LocalContext*import androidx.compose.ui.text.input.KeyboardType  
import androidx.compose.ui.unit.dp  
import androidx.core.content.ContextCompat  
import com.google.android.gms.location.LocationServices  
import com.google.android.gms.location.Priority  
import com.google.firebase.database.\*  
import java.text.NumberFormat  
import java.util.Locale  
import kotlin.math.\*  
  
// mis clases/recursos  
import cl.andres.semana4.RangesViewModel  
import cl.andres.semana4.SettingsActivity  
import cl.andres.semana4.R  
  
// --- Constantes ---  
private const val *BODEGA\_LAT* = -35.016  
private const val *BODEGA\_LON* = -71.333  
private const val *RADIO\_GRATIS\_KM* = 20.0  
  
// --- Utilidades ---  
fun gradosARadianes(grados: Double): Double = grados \* Math.*PI* / 180.0  
  
fun haversineKm(lat1: Double, lon1: Double, lat2: Double, lon2: Double): Double {  
 val R = 6371.0  
 val dLat = *gradosARadianes*(lat2 - lat1)  
 val dLon = *gradosARadianes*(lon2 - lon1)  
 val lat1R = *gradosARadianes*(lat1)  
 val lat2R = *gradosARadianes*(lat2)  
 val a = *sin*(dLat / 2).*pow*(2) + *cos*(lat1R) \* *cos*(lat2R) \* *sin*(dLon / 2).*pow*(2)  
 val c = 2 \* *atan2*(*sqrt*(a), *sqrt*(1 - a))  
 return R \* c  
}  
  
data class ShippingInput(val totalCompra: Int, val distanciaKm: Double)  
data class ShippingResult(val costoDespacho: Int, val aplicaGratis: Boolean)  
  
fun calcularDespacho(input: ShippingInput): ShippingResult {  
 val total = input.totalCompra  
 val km = input.distanciaKm  
 return if (total >= 50\_000 && km <= *RADIO\_GRATIS\_KM*) {  
 ShippingResult(0, true)  
 } else if (total >= 25\_000) {  
 ShippingResult((150 \* km).toInt(), false)  
 } else {  
 ShippingResult((300 \* km).toInt(), false)  
 }  
}  
  
fun formatoCLP(valor: Int): String {  
 val nf = NumberFormat.getIntegerInstance(Locale("es", "CL"))  
 return "$" + nf.format(valor)  
}  
  
// =======================================================  
// MainActivity  
// =======================================================  
class MainActivity : ComponentActivity() {  
  
 // Firebase + alarma + ViewModel (Room)  
 private lateinit var db: DatabaseReference  
 private var vib: Vibrator? = null // ahora nullable  
 private var mp: MediaPlayer? = null // ahora nullable  
 private val vm: RangesViewModel by *viewModels* **{** RangesViewModel.factory(this) **}** override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
  
 vib = getSystemService(Context.*VIBRATOR\_SERVICE*) as Vibrator  
 mp = MediaPlayer.create(this, R.raw.*alerta*) // res/raw/alerta.mp3 (minúsculas)  
 db = FirebaseDatabase.getInstance().*reference  
  
 setContent* **{** AppScreen(  
 // Suscripción a RTDB en tiempo real  
 subscribeTemperature = **{** onValue **->** db.child("sensors").child("truck1").child("temperatura")  
 .addValueEventListener(object : ValueEventListener {  
 override fun onDataChange(snapshot: DataSnapshot) {  
 val f = snapshot.getValue(Double::class.*java*) ?: return  
 onValue(f)  
 }  
 override fun onCancelled(error: DatabaseError) {}  
 })  
 **}**,  
 // Rangos desde Room  
 getRanges = **{** cb **->** vm.getRanges(cb) **}**,  
 // Alarma si sale de rango  
 onOutOfRange = **{** triggerAlarm() **}**,  
 // Abrir configuración  
 onOpenSettings = **{** startActivity(Intent(this@MainActivity, SettingsActivity::class.*java*))  
 **}** )  
 **}** // debug rápido  
 val grados = 90.0  
 val rad = *gradosARadianes*(grados)  
 *println*("$grados grados = $rad radianes")  
 }  
  
 private fun triggerAlarm() {  
 // vibración segura  
 vib?.*let* **{** v **->** if (Build.VERSION.*SDK\_INT* >= Build.VERSION\_CODES.*O*) {  
 v.vibrate(VibrationEffect.createOneShot(700, VibrationEffect.*DEFAULT\_AMPLITUDE*))  
 } else {  
 @Suppress("DEPRECATION")  
 v.vibrate(700)  
 }  
 **}** // sonido seguro  
 mp?.*let* **{** player **->** if (!player.*isPlaying*) player.start()  
 **}** }  
  
 override fun onDestroy() {  
 super.onDestroy()  
 // libero recursos para evitar leaks/crashes  
 mp?.release()  
 mp = null  
 vib = null  
 }  
}  
  
// =======================================================  
// UI Compose (lo que ya tenías + monitor de temperatura)  
// =======================================================  
@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
fun AppScreen(  
 subscribeTemperature: ((Double) -> Unit) -> Unit = **{}**,  
 getRanges: ((Double, Double) -> Unit) -> Unit = **{}**,  
 onOutOfRange: () -> Unit = **{}**,  
 onOpenSettings: () -> Unit = **{}**) {  
 var kmDetectado by remember **{** *mutableStateOf*<Double?>(null) **}** Scaffold(  
 topBar = **{** TopAppBar(title = **{** Text("TALLER DE APLICACIONES MÓVILES") **}**) **}** ) **{** padding **->** Column(  
 modifier = Modifier  
 .*padding*(padding)  
 .*padding*(16.*dp*)  
 .*fillMaxWidth*(),  
 verticalArrangement = Arrangement.spacedBy(16.*dp*)  
 ) **{** Text("Aplicación.")  
 ConversorRadianesUI()  
 Divider()  
 CalculadoraDespachoUI(kmDetectado = kmDetectado)  
 Divider()  
 DistanciaBodegaUI(  
 bodegaLat = *BODEGA\_LAT*,  
 bodegaLon = *BODEGA\_LON*,  
 onKmDetectado = **{** km **->** kmDetectado = km **}** )  
 Divider()  
 TemperatureMonitorUI(  
 subscribeTemperature = subscribeTemperature,  
 getRanges = getRanges,  
 onOutOfRange = onOutOfRange,  
 onOpenSettings = onOpenSettings  
 )  
 **}  
 }**}  
  
@Composable  
fun ConversorRadianesUI() {  
 var entrada by remember **{** *mutableStateOf*("") **}** var resultado by remember **{** *mutableStateOf*<Double?>(null) **}** var error by remember **{** *mutableStateOf*<String?>(null) **}** OutlinedTextField(  
 value = entrada,  
 onValueChange = **{** entrada = **it**; error = null **}**,  
 label = **{** Text("Grados (ej: 90)") **}**,  
 singleLine = true,  
 keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),  
 modifier = Modifier.*fillMaxWidth*(),  
 isError = error != null,  
 supportingText = **{** if (error != null) Text(error!!) **}** )  
  
 Button(  
 onClick = **{** val valor = entrada.*replace*(',', '.').*toDoubleOrNull*()  
 if (valor == null) {  
 error = "Ingrese un número válido"  
 resultado = null  
 } else {  
 val rad = *gradosARadianes*(valor)  
 resultado = rad  
 }  
 **}**,  
 modifier = Modifier.*fillMaxWidth*(),  
 shape = MaterialTheme.shapes.extraLarge  
 ) **{** Text("Convertir a radianes") **}** resultado?.*let* **{** rad **->** Text("Resultado: $rad rad") **}**}  
  
@Composable  
fun DistanciaBodegaUI(  
 bodegaLat: Double,  
 bodegaLon: Double,  
 onKmDetectado: (Double) -> Unit  
) {  
 val context = *LocalContext*.current  
 val fused = remember **{** LocationServices.getFusedLocationProviderClient(context) **}** val permisos = rememberLauncherForActivityResult(  
 ActivityResultContracts.RequestMultiplePermissions()  
 ) **{ }** var distancia by remember **{** *mutableStateOf*<Double?>(null) **}** var error by remember **{** *mutableStateOf*<String?>(null) **}** Column(verticalArrangement = Arrangement.spacedBy(8.*dp*)) **{** Text("Ubicación dispositivo a bodega")  
  
 Row(horizontalArrangement = Arrangement.spacedBy(8.*dp*)) **{** Button(  
 onClick = **{** val lacksFine = ContextCompat.checkSelfPermission(  
 context, Manifest.permission.*ACCESS\_FINE\_LOCATION* ) != PackageManager.*PERMISSION\_GRANTED* val lacksCoarse = ContextCompat.checkSelfPermission(  
 context, Manifest.permission.*ACCESS\_COARSE\_LOCATION* ) != PackageManager.*PERMISSION\_GRANTED* if (lacksFine || lacksCoarse) {  
 permisos.launch(  
 *arrayOf*(  
 Manifest.permission.*ACCESS\_FINE\_LOCATION*,  
 Manifest.permission.*ACCESS\_COARSE\_LOCATION* )  
 )  
 return@Button  
 }  
  
 fused.getCurrentLocation(Priority.*PRIORITY\_HIGH\_ACCURACY*, null)  
 .addOnSuccessListener **{** loc **->** if (loc != null) {  
 val km = *haversineKm*(loc.*latitude*, loc.*longitude*, bodegaLat, bodegaLon)  
 distancia = km  
 onKmDetectado(km)  
 } else {  
 error = "No fue posible obtener la ubicación actual."  
 }  
 **}** .addOnFailureListener **{** error = **it**.message ?: "Fallo al obtener ubicación." **}  
 }**,  
 modifier = Modifier.weight(1f),  
 shape = MaterialTheme.shapes.extraLarge  
 ) **{** Text("Obtener y calcular") **}** distancia?.*let* **{** km **->** AssistChip(  
 onClick = **{** onKmDetectado(km) **}**,  
 label = **{** Text("Usar ${"%.2f".*format*(km)} km") **}** )  
 **}  
 }** if (distancia != null) Text("Distancia: ${"%.2f".*format*(distancia)} km")  
 if (error != null) Text(error!!, color = MaterialTheme.colorScheme.error)  
 **}**}  
  
@Composable  
fun CalculadoraDespachoUI(kmDetectado: Double? = null) {  
 var totalTxt by remember **{** *mutableStateOf*("") **}** var kmTxt by remember **{** *mutableStateOf*("") **}** var mensaje by remember **{** *mutableStateOf*("") **}** var errorTotal by remember **{** *mutableStateOf*<String?>(null) **}** var errorKm by remember **{** *mutableStateOf*<String?>(null) **}** LaunchedEffect(kmDetectado) **{** if (kmDetectado != null && kmTxt.*isBlank*()) {  
 kmTxt = "%.2f".*format*(kmDetectado)  
 }  
 **}** Text("Cálculo de despacho")  
  
 OutlinedTextField(  
 value = totalTxt,  
 onValueChange = **{** totalTxt = **it**; errorTotal = null **}**,  
 label = **{** Text("Total compra (CLP)") **}**,  
 singleLine = true,  
 keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),  
 modifier = Modifier.*fillMaxWidth*(),  
 isError = errorTotal != null,  
 supportingText = **{** if (errorTotal != null) Text(errorTotal!!) **}** )  
  
 OutlinedTextField(  
 value = kmTxt,  
 onValueChange = **{** kmTxt = **it**; errorKm = null **}**,  
 label = **{** Text("Distancia (km)") **}**,  
 singleLine = true,  
 keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),  
 modifier = Modifier.*fillMaxWidth*(),  
 isError = errorKm != null,  
 supportingText = **{** if (errorKm != null) Text(errorKm!!) **}** )  
  
 Button(  
 onClick = **{** val total = totalTxt.*replace*(".", "").*replace*(',', '.')  
 .*toDoubleOrNull*()?.toInt()  
 val km = kmTxt.*replace*(',', '.').*toDoubleOrNull*()  
  
 if (total == null) errorTotal = "Ingrese un total válido (entero)"  
 if (km == null) errorKm = "Ingrese kilómetros válidos (número)"  
  
 if (total != null && km != null) {  
 val r = *calcularDespacho*(ShippingInput(total, km))  
 mensaje = if (r.aplicaGratis) {  
 "Despacho GRATIS (total ≥ \$50.000 y ≤ ${*RADIO\_GRATIS\_KM*.toInt()} km)"  
 } else {  
 "Costo despacho: ${*formatoCLP*(r.costoDespacho)}"  
 }  
 }  
 **}**,  
 modifier = Modifier.*fillMaxWidth*(),  
 shape = MaterialTheme.shapes.extraLarge  
 ) **{** Text("Calcular despacho") **}** if (mensaje.*isNotBlank*()) Text(mensaje)  
}  
  
// -------- Monitor de temperatura (Parte B) --------  
@Composable  
fun TemperatureMonitorUI(  
 subscribeTemperature: ((Double) -> Unit) -> Unit,  
 getRanges: ((Double, Double) -> Unit) -> Unit,  
 onOutOfRange: () -> Unit,  
 onOpenSettings: () -> Unit  
) {  
 var f by remember **{** *mutableStateOf*<Double?>(null) **}** var c by remember **{** *mutableStateOf*<Double?>(null) **}** var minC by remember **{** *mutableStateOf*(-5.0) **}** var maxC by remember **{** *mutableStateOf*(4.0) **}** // cargo rangos (Room)  
 LaunchedEffect(Unit) **{** getRanges **{** min, max **->** minC = min; maxC = max **} }** // escucho Firebase  
 LaunchedEffect(Unit) **{** subscribeTemperature **{** fValue **->** f = fValue  
 c = (fValue - 32.0) \* 5.0 / 9.0  
 val cur = c!!  
 if (cur < minC || cur > maxC) onOutOfRange()  
 **}  
 }** Column(verticalArrangement = Arrangement.spacedBy(8.*dp*)) **{** Text("Monitoreo de Temperatura", style = MaterialTheme.typography.titleMedium)  
 Text("Fahrenheit: ${f?.*let* **{** String.*format*("%.2f °F", **it**) **}** ?: "--"}")  
 Text("Celsius: ${c?.*let* **{** String.*format*("%.2f °C", **it**) **}** ?: "--"}")  
 Text("Rango permitido: $minC °C — $maxC °C")  
 Button(onClick = onOpenSettings) **{** Text("Configurar rangos") **}  
 }**}

## MenuActivity.kt

package cl.andres.semana4  
  
import android.Manifest  
import android.annotation.SuppressLint  
import android.content.Intent  
import android.content.pm.PackageManager  
import android.os.Bundle  
import android.provider.Settings  
import android.widget.Toast  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
import androidx.activity.result.contract.ActivityResultContracts  
import androidx.compose.foundation.layout.Arrangement  
import androidx.compose.foundation.layout.Column  
import androidx.compose.foundation.layout.fillMaxWidth  
import androidx.compose.foundation.layout.padding  
import androidx.compose.material3.Button  
import androidx.compose.material3.CenterAlignedTopAppBar  
import androidx.compose.material3.ExperimentalMaterial3Api  
import androidx.compose.material3.MaterialTheme  
import androidx.compose.material3.OutlinedButton  
import androidx.compose.material3.Scaffold  
import androidx.compose.material3.Text  
import androidx.compose.runtime.Composable  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.unit.dp  
import androidx.core.content.ContextCompat  
import com.google.android.gms.location.FusedLocationProviderClient  
import com.google.android.gms.location.LocationServices  
import com.google.android.gms.location.Priority  
import com.google.firebase.database.FirebaseDatabase  
import com.google.firebase.database.ServerValue  
  
class MenuActivity : ComponentActivity() {  
  
 private lateinit var fused: FusedLocationProviderClient  
  
 // pido permisos de ubicación cuando entro al menú  
 private val permisosLauncher = registerForActivityResult(  
 ActivityResultContracts.RequestMultiplePermissions()  
 ) **{** resultados **->** val fine = resultados[Manifest.permission.*ACCESS\_FINE\_LOCATION*] == true  
 val coarse = resultados[Manifest.permission.*ACCESS\_COARSE\_LOCATION*] == true  
 if (fine || coarse) {  
 capturarYGuardarUbicacion()  
 } else {  
 Toast.makeText(  
 this,  
 "Permiso de ubicación denegado: no se guardó la posición.",  
 Toast.*LENGTH\_SHORT* ).show()  
 }  
 **}** override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
  
 fused = LocationServices.getFusedLocationProviderClient(this)  
  
 *setContent* **{** MaterialTheme **{** MenuScreen(  
 onOpenApp = **{** startActivity(Intent(this, MainActivity::class.*java*)) **}**,  
 onLogout = **{** // sin Auth: cierro sesión volviendo al login  
 startActivity(  
 Intent(this, LoginActivity::class.*java*).*apply* **{** addFlags(  
 Intent.*FLAG\_ACTIVITY\_CLEAR\_TOP* or  
 Intent.*FLAG\_ACTIVITY\_NEW\_TASK* or  
 Intent.*FLAG\_ACTIVITY\_CLEAR\_TASK* )  
 **}** )  
 finish()  
 **}** )  
 **}  
 }** }  
  
 override fun onStart() {  
 super.onStart()  
 // guardo la ubicación apenas abro el menú  
 solicitarPermisoYGuardar()  
 }  
  
 private fun solicitarPermisoYGuardar() {  
 val fineOk = ContextCompat.checkSelfPermission(  
 this, Manifest.permission.*ACCESS\_FINE\_LOCATION* ) == PackageManager.*PERMISSION\_GRANTED* val coarseOk = ContextCompat.checkSelfPermission(  
 this, Manifest.permission.*ACCESS\_COARSE\_LOCATION* ) == PackageManager.*PERMISSION\_GRANTED* if (fineOk || coarseOk) {  
 capturarYGuardarUbicacion()  
 } else {  
 permisosLauncher.launch(  
 *arrayOf*(  
 Manifest.permission.*ACCESS\_FINE\_LOCATION*,  
 Manifest.permission.*ACCESS\_COARSE\_LOCATION* )  
 )  
 }  
 }  
  
 // capturo la ubicación y la dejo en RTDB bajo /devices/<ANDROID\_ID>/lastLocation  
 @SuppressLint("MissingPermission")  
 private fun capturarYGuardarUbicacion() {  
 val fineOk = ContextCompat.checkSelfPermission(  
 this, Manifest.permission.*ACCESS\_FINE\_LOCATION* ) == PackageManager.*PERMISSION\_GRANTED* val coarseOk = ContextCompat.checkSelfPermission(  
 this, Manifest.permission.*ACCESS\_COARSE\_LOCATION* ) == PackageManager.*PERMISSION\_GRANTED* if (!fineOk && !coarseOk) {  
 permisosLauncher.launch(  
 *arrayOf*(  
 Manifest.permission.*ACCESS\_FINE\_LOCATION*,  
 Manifest.permission.*ACCESS\_COARSE\_LOCATION* )  
 )  
 return  
 }  
  
 // como no uso FirebaseAuth, identifico el dispositivo con el ANDROID\_ID  
 val deviceId = Settings.Secure.getString(*contentResolver*, Settings.Secure.*ANDROID\_ID*)  
 ?: "unknown"  
  
 try {  
 fused.getCurrentLocation(Priority.*PRIORITY\_HIGH\_ACCURACY*, null)  
 .addOnSuccessListener **{** loc **->** if (loc != null) {  
 val ref = FirebaseDatabase.getInstance()  
 .getReference("devices") // <- aquí cambio a "devices"  
 .child(deviceId)  
 .child("lastLocation")  
  
 val data = *mapOf*(  
 "lat" *to* loc.*latitude*,  
 "lon" *to* loc.*longitude*,  
 "ts" *to* ServerValue.*TIMESTAMP* )  
  
 ref.setValue(data)  
 // .addOnSuccessListener { Toast.makeText(this, "Ubicación guardada", Toast.LENGTH\_SHORT).show() }  
 .addOnFailureListener **{** e **->** Toast.makeText(  
 this,  
 "Error al guardar ubicación: ${e.*localizedMessage*}",  
 Toast.*LENGTH\_SHORT* ).show()  
 **}** } else {  
 Toast.makeText(  
 this,  
 "No fue posible obtener la ubicación.",  
 Toast.*LENGTH\_SHORT* ).show()  
 }  
 **}** .addOnFailureListener **{** e **->** Toast.makeText(  
 this,  
 "Error al obtener ubicación: ${e.*localizedMessage*}",  
 Toast.*LENGTH\_SHORT* ).show()  
 **}** } catch (se: SecurityException) {  
 Toast.makeText(this, "Permiso de ubicación requerido.", Toast.*LENGTH\_SHORT*).show()  
 }  
 }  
}  
  
@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
private fun MenuScreen(  
 onOpenApp: () -> Unit,  
 onLogout: () -> Unit  
) {  
 Scaffold(  
 topBar = **{** CenterAlignedTopAppBar(title = **{** Text("Menú") **}**) **}** ) **{** padding **->** Column(  
 modifier = Modifier  
 .*padding*(padding)  
 .*padding*(24.*dp*)  
 .*fillMaxWidth*(),  
 verticalArrangement = Arrangement.spacedBy(12.*dp*)  
 ) **{** Button(  
 onClick = onOpenApp,  
 modifier = Modifier.*fillMaxWidth*(),  
 shape = MaterialTheme.shapes.extraLarge  
 ) **{** Text("Abrir aplicación") **}** OutlinedButton(  
 onClick = onLogout,  
 modifier = Modifier.*fillMaxWidth*(),  
 shape = MaterialTheme.shapes.extraLarge  
 ) **{** Text("Cerrar sesión") **}  
 }  
 }**}

## LoginActivity.kt

package cl.andres.semana4  
  
import android.content.Intent  
import android.os.Bundle  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
import androidx.compose.foundation.layout.\*  
import androidx.compose.foundation.text.KeyboardOptions  
import androidx.compose.material.icons.Icons  
import androidx.compose.material.icons.filled.*Visibility*import androidx.compose.material.icons.filled.*VisibilityOff*import androidx.compose.material3.\*  
import androidx.compose.runtime.\*  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.text.input.KeyboardType  
import androidx.compose.ui.text.input.PasswordVisualTransformation  
import androidx.compose.ui.text.input.VisualTransformation  
import androidx.compose.ui.unit.dp  
  
*/\*\*  
 \* Versión sin Firebase Auth / Google Sign-In para mantener minSdk 21 (Lollipop).  
 \* Valida formato básico y navega al menú si los datos son plausibles.  
 \*/*class LoginActivity : ComponentActivity() {  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *setContent* **{** MaterialTheme **{** LoginScreen(  
 onEmailLogin = **{** email, pass **->** val ok = email.*contains*("@") && pass.length >= 4  
 if (ok) {  
 startActivity(Intent(this@LoginActivity, MenuActivity::class.*java*))  
 finish()  
 }  
 **}**,  
 onRegister = **{** \_, \_ **->** // Mock de registro: simplemente continúa al menú  
 startActivity(Intent(this@LoginActivity, MenuActivity::class.*java*))  
 finish()  
 **}** )  
 **}  
 }** }  
}  
  
@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
fun LoginScreen(  
 onEmailLogin: (String, String) -> Unit,  
 onRegister: (String, String) -> Unit  
) {  
 var email by remember **{** *mutableStateOf*("") **}** var pass by remember **{** *mutableStateOf*("") **}** var passVisible by remember **{** *mutableStateOf*(false) **}** var error by remember **{** *mutableStateOf*<String?>(null) **}** Scaffold(  
 topBar = **{** TopAppBar(title = **{** Text("Inicia sesión") **}**) **}** ) **{** padding **->** Column(  
 modifier = Modifier  
 .*padding*(padding)  
 .*padding*(16.*dp*),  
 verticalArrangement = Arrangement.spacedBy(12.*dp*)  
 ) **{** OutlinedTextField(  
 value = email,  
 onValueChange = **{** email = **it**; error = null **}**,  
 label = **{** Text("Correo") **}**,  
 singleLine = true,  
 keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Email),  
 modifier = Modifier.*fillMaxWidth*()  
 )  
  
 OutlinedTextField(  
 value = pass,  
 onValueChange = **{** pass = **it**; error = null **}**,  
 label = **{** Text("Contraseña") **}**,  
 singleLine = true,  
 visualTransformation = if (passVisible) VisualTransformation.None else PasswordVisualTransformation(),  
 trailingIcon = **{** IconButton(onClick = **{** passVisible = !passVisible **}**) **{** Icon(  
 imageVector = if (passVisible) Icons.Filled.*VisibilityOff* else Icons.Filled.*Visibility*,  
 contentDescription = null  
 )  
 **}  
 }**,  
 keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),  
 modifier = Modifier.*fillMaxWidth*()  
 )  
  
 if (error != null) {  
 Text(error!!, color = MaterialTheme.colorScheme.error)  
 } else {  
 Text("Mínimo 4 caracteres", style = MaterialTheme.typography.bodySmall)  
 }  
  
 Button(  
 onClick = **{** if (email.*contains*("@") && pass.length >= 4) {  
 onEmailLogin(email.*trim*(), pass)  
 } else {  
 error = "Correo o contraseña inválidos."  
 }  
 **}**,  
 modifier = Modifier.*fillMaxWidth*(),  
 shape = MaterialTheme.shapes.extraLarge  
 ) **{** Text("Entrar") **}** OutlinedButton(  
 onClick = **{** if (email.*contains*("@") && pass.length >= 4) {  
 onRegister(email.*trim*(), pass)  
 } else {  
 error = "Datos inválidos para registro."  
 }  
 **}**,  
 modifier = Modifier.*fillMaxWidth*(),  
 shape = MaterialTheme.shapes.extraLarge  
 ) **{** Text("Crear cuenta") **}** Divider(Modifier.*padding*(vertical = 4.*dp*))  
  
 Text(  
 "Inicio con Google deshabilitado en esta versión (compatibilidad API 21).",  
 style = MaterialTheme.typography.bodySmall  
 )  
 **}  
 }**}

## SettingsActivity.kt

package cl.andres.semana4  
  
import android.os.Bundle  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
import androidx.activity.viewModels  
import androidx.compose.foundation.layout.\*  
import androidx.compose.material3.\*  
import androidx.compose.runtime.\*  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.text.input.KeyboardType  
import androidx.compose.ui.unit.dp  
import androidx.compose.ui.platform.*LocalContext*import androidx.compose.foundation.text.KeyboardOptions  
  
class SettingsActivity : ComponentActivity() {  
 private val vm: RangesViewModel by *viewModels* **{** RangesViewModel.factory(this) **}** override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *setContent* **{** MaterialTheme **{** SettingsScreen(  
 load = **{** cb **->** vm.getRanges(cb) **}**,  
 save = **{** minC, maxC **->** vm.setRanges(minC, maxC); finish() **}** )  
 **}  
 }** }  
}  
  
@Composable  
fun SettingsScreen(  
 load: ((Double, Double) -> Unit) -> Unit,  
 save: (Double, Double) -> Unit  
) {  
 var min by remember **{** *mutableStateOf*("-5.0") **}** var max by remember **{** *mutableStateOf*("4.0") **}** // Carga inicial desde Room  
 LaunchedEffect(Unit) **{** load **{** minC, maxC **->** min = minC.toString()  
 max = maxC.toString()  
 **}  
 }** Column(  
 modifier = Modifier.*fillMaxSize*().*padding*(24.*dp*),  
 verticalArrangement = Arrangement.spacedBy(16.*dp*)  
 ) **{** Text("Configurar rangos (°C)", style = MaterialTheme.typography.titleLarge)  
  
 OutlinedTextField(  
 value = min,  
 onValueChange = **{** min = **it }**,  
 label = **{** Text("Mínimo °C") **}**,  
 keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),  
 singleLine = true  
 )  
 OutlinedTextField(  
 value = max,  
 onValueChange = **{** max = **it }**,  
 label = **{** Text("Máximo °C") **}**,  
 keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),  
 singleLine = true  
 )  
  
 Button(  
 onClick = **{** val minC = min.*toDoubleOrNull*() ?: -5.0  
 val maxC = max.*toDoubleOrNull*() ?: 4.0  
 save(minC, maxC)  
 **}**,  
 modifier = Modifier.align(Alignment.End)  
 ) **{** Text("Guardar") **}  
 }**}

## Ranges.kt

package cl.andres.semana4  
  
import android.content.Context  
import androidx.lifecycle.ViewModel  
import androidx.lifecycle.ViewModelProvider  
import androidx.room.\*  
import kotlinx.coroutines.CoroutineScope  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.launch  
  
@Entity(tableName = "ranges")  
data class RangeConfig(  
 @PrimaryKey val id: Int = 1,  
 val minC: Double,  
 val maxC: Double  
)  
  
@Dao  
interface RangeDao {  
 @Query("SELECT \* FROM ranges WHERE id = 1")  
 suspend fun get(): RangeConfig?  
  
 @Insert(onConflict = OnConflictStrategy.REPLACE)  
 suspend fun save(cfg: RangeConfig)  
}  
  
@Database(entities = [RangeConfig::class], version = 1)  
abstract class AppDb : RoomDatabase() {  
 abstract fun dao(): RangeDao  
 companion object {  
 @Volatile private var I: AppDb? = null  
 fun get(ctx: Context): AppDb =  
 I ?: *synchronized*(this) **{** I ?: Room.databaseBuilder(ctx, AppDb::class.*java*, "ranges.db").build().*also* **{** I = **it }  
 }** }  
}  
  
class RangesViewModel(private val ctx: Context): ViewModel() {  
 private val dao = AppDb.get(ctx).dao()  
  
 fun getRanges(cb: (Double, Double) -> Unit) {  
 *CoroutineScope*(Dispatchers.IO).*launch* **{** val cfg = dao.get() ?: RangeConfig(minC = -5.0, maxC = 4.0).*also* **{** dao.save(**it**) **}** cb(cfg.minC, cfg.maxC)  
 **}** }  
  
 fun setRanges(minC: Double, maxC: Double) {  
 *CoroutineScope*(Dispatchers.IO).*launch* **{** dao.save(RangeConfig(minC = minC, maxC = maxC))  
 **}** }  
  
 companion object {  
 val factory = **{** ctx: Context **->** object : ViewModelProvider.Factory {  
 override fun <T : ViewModel> create(modelClass: Class<T>): T =  
 RangesViewModel(ctx.*applicationContext*) as T  
 }  
 **}** }  
}

## Codigo tinkercad

// Arduino UNO + TMP36 — Emulación "guardado" en Firebase (JSON por Serial)

const int TMP36\_PIN = A0;

void setup() {

Serial.begin(9600);

}

void loop() {

int adc = analogRead(TMP36\_PIN);

float V = adc \* (5.0 / 1023.0); // Voltaje leído (0-5V)

float c = (V - 0.5) \* 100.0; // Conversión a °C

float f = c \* 9.0 / 5.0 + 32.0; // Conversión a °F

unsigned long t = millis() / 1000; // Tiempo en segundos

// Emulación de envío a Firebase (muestra JSON)

Serial.print("{\"temperatura\":");

Serial.print(f, 2);

Serial.print(",\"temperaturaC\":");

Serial.print(c, 2);

Serial.print(",\"unidad\":\"F\",\"updatedAt\":");

Serial.print(t);

Serial.println("}");

delay(2000);

}

# Conclusión

La aplicación cumple con todos los requisitos solicitados: lectura de temperatura desde Firebase, conversión a grados Celsius, configuración de rangos mediante base de datos local y generación de alarma en caso de valores fuera del rango. El proyecto fue ejecutado correctamente sin errores y validado mediante las pruebas en el emulador Android Studio.

# Anexo

https://github.com/Gorilacl/Actividad6