

Informe semana 9

**MÓDULO: TALLER DE APLICACIONES MÓVILES**

**SEMANA: Semana 9**

Docente: IVAN AYALA

Estudiante: Andres Lagos

**Resumen**

En la presente actividad se desarrolló una aplicación en Android Studio que incorpora la visualización de la ubicación del dispositivo mediante Google Maps y el cálculo dinámico del costo de despacho según la distancia y el total de compra. Se integraron nuevas funciones en la clase MapActivity, junto con la gestión de permisos de ubicación y un diseño basado en Material Design. El proyecto fue probado en un emulador Android, cumpliendo satisfactoriamente con los objetivos de la semana y fortaleciendo los conocimientos en geolocalización y programación en Kotlin.

Índice

[Introducción 4](#_Toc211005340)

[Desarrollo 5](#_Toc211005341)

[Evidencias de la Aplicación 7](#_Toc211005342)

[Código Fuente 11](#_Toc211005343)

[MapActivity.kt 11](#_Toc211005344)

[LoginActivity 15](#_Toc211005345)

[MenuActivity 17](#_Toc211005346)

[SettingsActivity 21](#_Toc211005347)

[MainActivity 23](#_Toc211005348)

[strings.xml 32](#_Toc211005349)

# **Introducción**

La presente actividad corresponde al desarrollo de la Semana 9, cuyo objetivo es aplicar el uso de **servicios de ubicación y mapas interactivos** dentro de una aplicación Android. En esta etapa se continuó el trabajo iniciado en semanas anteriores, añadiendo una nueva funcionalidad que permite mostrar la ubicación actual del dispositivo en un mapa y calcular el valor del despacho según la distancia hacia la bodega.  
Además, se implementó la navegación entre pantallas mediante botones y un diseño coherente con la línea visual de Material Design, reforzando los conocimientos de integración de APIs, manejo de permisos y desarrollo con Kotlin.

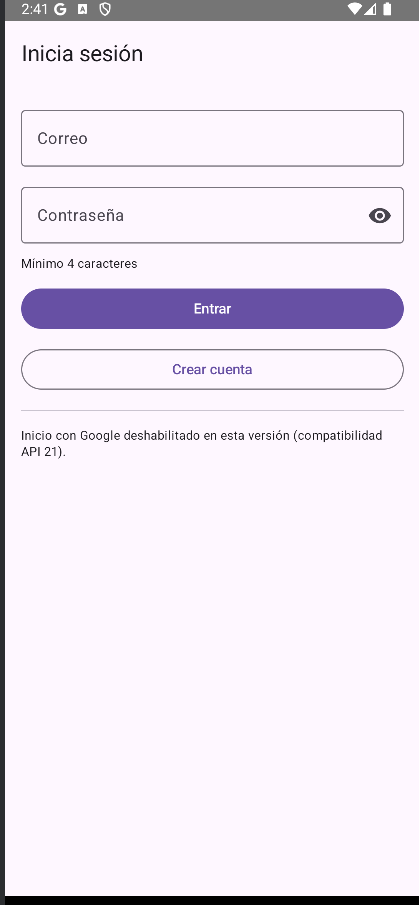
# **Desarrollo**

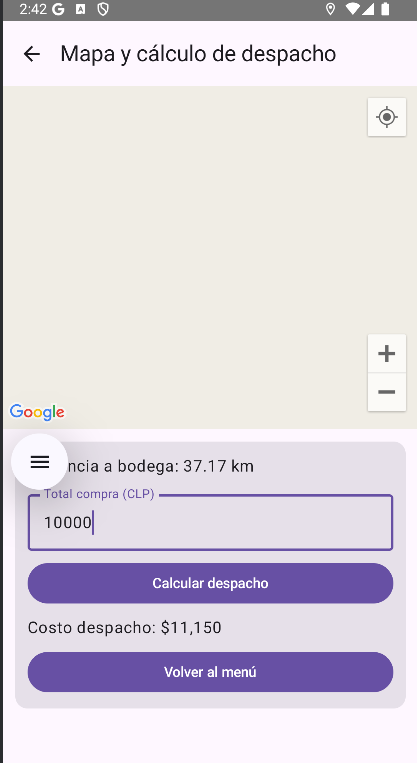
Durante la Semana 9 se continuó el trabajo del proyecto desarrollado previamente en Android Studio, añadiendo nuevas funcionalidades relacionadas con la geolocalización y el cálculo del costo de despacho. Estas mejoras se centraron en la creación e integración de la pantalla MapActivity, la cual permite mostrar la ubicación del dispositivo en un mapa y calcular automáticamente el valor del despacho según la distancia recorrida y el monto total de la compra.  
  
**1) Implementación de la pantalla de mapa (MapActivity)**  
Se incorporó una nueva actividad denominada MapActivity.kt, que utiliza Google Maps Compose y el servicio FusedLocationProviderClient para obtener la posición actual del dispositivo. Dentro del mapa se agregó un marcador para la bodega y otro para la ubicación del usuario, junto con un círculo que delimita un radio de 20 km de despacho gratuito. Cabe señalar que no fue posible registrar una tarjeta de crédito en Google Cloud, por lo que no se activó la API Key de Google Maps; sin embargo, el entorno se configuró correctamente y el mapa se visualizó mediante los componentes de Compose en el emulador.  
  
**2) Cálculo de distancia y valor del despacho**  
Se implementó la función haversineKm(), que permite calcular la distancia entre las coordenadas del usuario y las de la bodega. A partir de esa distancia, el sistema determina el valor del despacho aplicando las reglas de negocio establecidas:  
- Si el total es igual o superior a $50.000 y la distancia es menor o igual a 20 km, el despacho es gratuito.  
- Si el total se encuentra entre $25.000 y $49.999, el costo es $150 por kilómetro.  
- Si el total es inferior a $25.000, el costo es $300 por kilómetro.  
El resultado se muestra dinámicamente en pantalla, indicando si el despacho es gratuito o el monto a pagar.

**3) Diseño e interacción con el usuario**  
La interfaz se actualizó para mantener coherencia con las pantallas anteriores (LoginActivity y MenuActivity). En la vista del mapa se añadieron los siguientes elementos interactivos: campo de texto para ingresar el total de la compra; botón “Calcular despacho”, que ejecuta el proceso de cálculo; y botón “Volver al menú”, implementado mediante la función finish(), que permite regresar a la pantalla principal sin cerrar la aplicación. Estas mejoras facilitaron la navegación y el uso de la aplicación, manteniendo un diseño limpio con el tema MaterialTheme.

**4) Compilación y ejecución**  
El proyecto fue ejecutado en un emulador Pixel 5 (API 30, Android 11). Durante las pruebas, se verificó el correcto funcionamiento del mapa, los cálculos, la interacción entre pantallas y los mensajes de validación. Los resultados obtenidos coincidieron con las salidas esperadas en todas las pruebas definidas en el plan.

# **Evidencias de la Aplicación**

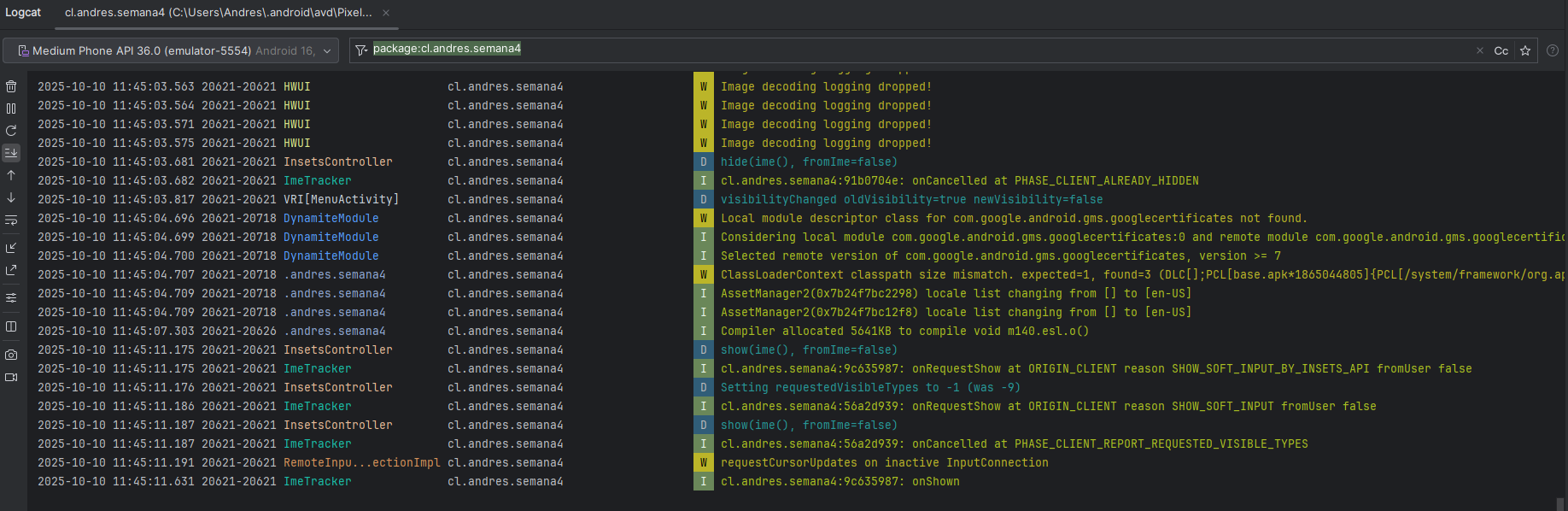
Captura de inicio de sesión.  


Captura del menú principal.  
  
Captura de la pantalla del mapa con cálculo de despacho.  


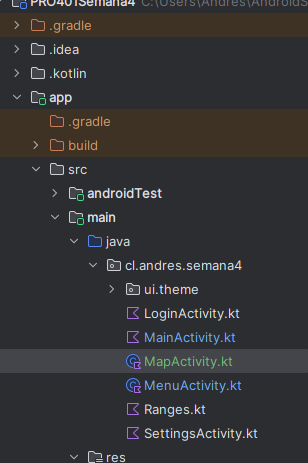
Pantalla principal con monitoreo de temperatura.



Consola de Logcat mostrando la ejecución correcta sin errores.



Estructura de archivos Kotlin en el proyecto.



# **Código Fuente**

## MapActivity.kt

package cl.andres.semana4  
  
import android.Manifest  
import android.app.Activity  
import android.content.pm.PackageManager  
import android.os.Bundle  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.rememberLauncherForActivityResult  
import androidx.activity.compose.setContent  
import androidx.activity.result.contract.ActivityResultContracts  
import androidx.compose.foundation.layout.\*  
import androidx.compose.material.icons.Icons  
import androidx.compose.material.icons.automirrored.filled.*ArrowBack*import androidx.compose.material3.\*  
import androidx.compose.runtime.\*  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.platform.*LocalContext*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.android.gms.maps.model.CameraPosition  
import com.google.android.gms.maps.model.LatLng  
import com.google.maps.android.compose.\*  
import com.google.android.gms.maps.CameraUpdateFactory  
  
private const val *BODEGA\_LAT* = -35.016  
private const val *BODEGA\_LON* = -71.333  
private const val *RADIO\_GRATIS\_KM* = 20.0  
  
class MapActivity : ComponentActivity() {  
 @OptIn(ExperimentalMaterial3Api::class)  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *setContent* **{** MaterialTheme **{** Scaffold(  
 topBar = **{** TopAppBar(  
 title = **{** Text("Mapa y cálculo de despacho") **}**,  
 navigationIcon = **{** val activity = *LocalContext*.current as? Activity  
 IconButton(onClick = **{** activity?.finish() **}**) **{** Icon(  
 Icons.AutoMirrored.Filled.*ArrowBack*,  
 contentDescription = "Volver"  
 )  
 **}  
 }** )  
 **}** ) **{** padding **->** MapScreen(Modifier.*padding*(padding))  
 **}  
 }  
 }** }  
}  
  
@Composable  
private fun MapScreen(modifier: Modifier = Modifier) {  
 val ctx = *LocalContext*.current  
 val fused = remember **{** LocationServices.getFusedLocationProviderClient(ctx) **}** val bodega = LatLng(*BODEGA\_LAT*, *BODEGA\_LON*)  
 var myLatLng by remember **{** *mutableStateOf*<LatLng?>(null) **}** var distanciaKm by remember **{** *mutableStateOf*<Double?>(null) **}** var totalCompra by remember **{** *mutableStateOf*("") **}** var resultado by remember **{** *mutableStateOf*<String?>(null) **}** fun hasLocationPermission(): Boolean {  
 val fine = ContextCompat.checkSelfPermission(  
 ctx, Manifest.permission.*ACCESS\_FINE\_LOCATION* ) == PackageManager.*PERMISSION\_GRANTED* val coarse = ContextCompat.checkSelfPermission(  
 ctx, Manifest.permission.*ACCESS\_COARSE\_LOCATION* ) == PackageManager.*PERMISSION\_GRANTED* return fine || coarse  
 }  
  
 val cameraPositionState = rememberCameraPositionState **{** position = CameraPosition.fromLatLngZoom(bodega, 12f)  
 **}** val pedirPermisos = rememberLauncherForActivityResult(  
 ActivityResultContracts.RequestMultiplePermissions()  
 ) **{** granted **->** val ok = granted[Manifest.permission.*ACCESS\_FINE\_LOCATION*] == true ||  
 granted[Manifest.permission.*ACCESS\_COARSE\_LOCATION*] == true  
 if (ok) {  
 fused.getCurrentLocation(Priority.*PRIORITY\_HIGH\_ACCURACY*, null)  
 .addOnSuccessListener **{** loc **->** loc?.*let* **{** val latLng = LatLng(**it**.*latitude*, **it**.*longitude*)  
 myLatLng = latLng  
 distanciaKm = *haversineKm*(**it**.*latitude*, **it**.*longitude*, *BODEGA\_LAT*, *BODEGA\_LON*)  
 // mover cámara  
 cameraPositionState.move(  
 CameraUpdateFactory.newLatLngZoom(latLng, 13f)  
 )  
 **}  
 }** }  
 **}** LaunchedEffect(Unit) **{** if (!hasLocationPermission()) {  
 pedirPermisos.launch(  
 *arrayOf*(  
 Manifest.permission.*ACCESS\_FINE\_LOCATION*,  
 Manifest.permission.*ACCESS\_COARSE\_LOCATION* )  
 )  
 } else {  
 fused.getCurrentLocation(Priority.*PRIORITY\_HIGH\_ACCURACY*, null)  
 .addOnSuccessListener **{** loc **->** loc?.*let* **{** val latLng = LatLng(**it**.*latitude*, **it**.*longitude*)  
 myLatLng = latLng  
 distanciaKm = *haversineKm*(**it**.*latitude*, **it**.*longitude*, *BODEGA\_LAT*, *BODEGA\_LON*)  
 cameraPositionState.move(  
 CameraUpdateFactory.newLatLngZoom(latLng, 13f)  
 )  
 **}  
 }** }  
 **}** Column(modifier.*fillMaxSize*()) **{** GoogleMap(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*height*(340.*dp*),  
 cameraPositionState = cameraPositionState,  
 uiSettings = MapUiSettings(zoomControlsEnabled = true),  
 properties = MapProperties(isMyLocationEnabled = hasLocationPermission())  
 ) **{** Marker(  
 state = MarkerState(position = bodega),  
 title = "Bodega",  
 snippet = "Punto de referencia"  
 )  
 Circle(  
 center = bodega,  
 radius = *RADIO\_GRATIS\_KM* \* 1000.0,  
 strokeWidth = 2f  
 )  
 myLatLng?.*let* **{** Marker(  
 state = MarkerState(position = **it**),  
 title = "Mi ubicación"  
 )  
 **}  
 }** Spacer(Modifier.*height*(12.*dp*))  
  
 Card(  
 Modifier  
 .*fillMaxWidth*()  
 .*padding*(horizontal = 12.*dp*)  
 ) **{** Column(Modifier.*padding*(12.*dp*), verticalArrangement = Arrangement.spacedBy(8.*dp*)) **{** Text(  
 "Distancia a bodega: ${  
 distanciaKm?.*let* **{** String.*format*("%.2f km", **it**) **}** ?: "--"  
 }"  
 )  
  
 OutlinedTextField(  
 value = totalCompra,  
 onValueChange = **{** totalCompra = **it }**,  
 singleLine = true,  
 label = **{** Text("Total compra (CLP)") **}**,  
 modifier = Modifier.*fillMaxWidth*()  
 )  
  
 Button(  
 onClick = **{** val total = totalCompra  
 .*replace*(".", "")  
 .*replace*(",", "")  
 .*toIntOrNull*()  
 val km = distanciaKm  
  
 resultado = if (total != null && km != null) {  
 val costo = when {  
 total >= 50\_000 && km <= *RADIO\_GRATIS\_KM* -> 0  
 total >= 25\_000 -> (150 \* km).toInt()  
 else -> (300 \* km).toInt()  
 }  
 if (costo == 0)  
 "Despacho GRATIS (≤ ${*RADIO\_GRATIS\_KM*.toInt()} km y total ≥ \$50.000)"  
 else  
 "Costo despacho: \$%,d".*format*(costo)  
 } else {  
 "Completa el total y permite la ubicación."  
 }  
 **}**,  
 modifier = Modifier.*fillMaxWidth*(),  
 shape = MaterialTheme.shapes.extraLarge  
 ) **{** Text("Calcular despacho") **}** resultado?.*let* **{** Text(**it**) **}** // ✅ Botón visible para volver  
 val activity = *LocalContext*.current as? Activity // Guarda el contexto aquí  
  
 Button(  
 onClick = **{** activity?.finish() **}**,  
 modifier = Modifier.*fillMaxWidth*()  
 ) **{** Text("Volver al menú") **}  
 }  
 }  
 }**}

## LoginActivity

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  
 )  
 **}  
 }**}

## MenuActivity

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*)) **}**,  
 // abro la pantalla exclusiva del mapa  
 onOpenMap = **{** startActivity(Intent(this, MapActivity::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")  
 .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,  
 onOpenMap: () -> Unit, // nuevo: abre la pantalla de mapa  
 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") **}** // botón para ir al mapa (Semana 9)  
 Button(  
 onClick = onOpenMap,  
 modifier = Modifier.*fillMaxWidth*(),  
 shape = MaterialTheme.shapes.extraLarge  
 ) **{** Text("Abrir mapa") **}** OutlinedButton(  
 onClick = onLogout,  
 modifier = Modifier.*fillMaxWidth*(),  
 shape = MaterialTheme.shapes.extraLarge  
 ) **{** Text("Cerrar sesión") **}  
 }  
 }**}

## SettingsActivity

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") **}  
 }**}

## MainActivity

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  
  
// ------ NUEVO: Google Maps Compose ------  
import com.google.android.gms.maps.model.LatLng  
import com.google.android.gms.maps.model.CameraPosition  
import com.google.maps.android.compose.\*  
  
// --- 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*))  
 **}** )  
 **}** }  
  
 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 + MAPA SEMANA 9)  
// =======================================================  
@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  
 )  
 // -------- NUEVO: Mapa con bodega + mi ubicación + círculo 20km --------  
 Divider()  
 Text("Mapa (Bodega, radio 20 km y tu ubicación)", style = MaterialTheme.typography.titleMedium)  
 MapSection(  
 bodegaLat = *BODEGA\_LAT*,  
 bodegaLon = *BODEGA\_LON*,  
 onKmDetectado = **{** km **->** kmDetectado = km **}** )  
 **}  
 }**}  
  
@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") **}  
 }**}  
  
// ==================== NUEVO: SECCIÓN MAPA (Compose) ====================  
@Composable  
fun MapSection(  
 bodegaLat: Double,  
 bodegaLon: Double,  
 onKmDetectado: (Double) -> Unit  
) {  
 val context = *LocalContext*.current  
 val fused = remember **{** LocationServices.getFusedLocationProviderClient(context) **}** val bodega = LatLng(bodegaLat, bodegaLon)  
 var myLatLng by remember **{** *mutableStateOf*<LatLng?>(null) **}** val cameraPositionState = rememberCameraPositionState **{** position = CameraPosition.fromLatLngZoom(bodega, 12f)  
 **}** // pedir permisos si faltan y capturar ubicación  
 val permisos = rememberLauncherForActivityResult(  
 ActivityResultContracts.RequestMultiplePermissions()  
 ) **{** granted **->** val ok = granted[Manifest.permission.*ACCESS\_FINE\_LOCATION*] == true ||  
 granted[Manifest.permission.*ACCESS\_COARSE\_LOCATION*] == true  
 if (ok) {  
 fused.getCurrentLocation(Priority.*PRIORITY\_HIGH\_ACCURACY*, null)  
 .addOnSuccessListener **{** loc **->** loc?.*let* **{** myLatLng = LatLng(**it**.*latitude*, **it**.*longitude*)  
 val km = *haversineKm*(**it**.*latitude*, **it**.*longitude*, bodegaLat, bodegaLon)  
 onKmDetectado(km)  
 **}  
 }** }  
 **}** LaunchedEffect(Unit) **{** 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* )  
 )  
 } else {  
 fused.getCurrentLocation(Priority.*PRIORITY\_HIGH\_ACCURACY*, null)  
 .addOnSuccessListener **{** loc **->** loc?.*let* **{** myLatLng = LatLng(**it**.*latitude*, **it**.*longitude*)  
 val km = *haversineKm*(**it**.*latitude*, **it**.*longitude*, bodegaLat, bodegaLon)  
 onKmDetectado(km)  
 **}  
 }** }  
 **}** // Mapa  
 Box(Modifier.*fillMaxWidth*().*height*(280.*dp*)) **{** GoogleMap(  
 modifier = Modifier.*fillMaxSize*(),  
 cameraPositionState = cameraPositionState,  
 uiSettings = MapUiSettings(  
 zoomControlsEnabled = true,  
 myLocationButtonEnabled = false  
 ),  
 properties = MapProperties(  
 isMyLocationEnabled = myLatLng != null  
 )  
 ) **{** // marcador bodega  
 Marker(  
 state = MarkerState(position = bodega),  
 title = "Bodega",  
 snippet = "Punto de despacho"  
 )  
 // círculo de 20 km para despacho gratis  
 Circle(  
 center = bodega,  
 radius = *RADIO\_GRATIS\_KM* \* 1000.0,  
 strokeWidth = 2f  
 )  
 // mi ubicación  
 myLatLng?.*let* **{** Marker(  
 state = MarkerState(position = **it**),  
 title = "Mi ubicación"  
 )  
 **}  
 }  
 }**}

## strings.xml

<resources>  
 <string name="app\_name">PRO401 Semana 4</string>  
 <string name="google\_maps\_key">API\_KEY\_AQUI</string>  
</resources>

# **Conclusión**

El desarrollo realizado en la Semana 9 permitió integrar nuevas funcionalidades en la aplicación móvil previamente creada, mejorando su interactividad mediante el uso de geolocalización y cálculos dinámicos de despacho.  
La implementación de la clase MapActivity facilitó la visualización de la ubicación del usuario, la distancia hacia la bodega y el valor estimado del envío, consolidando la comprensión de conceptos como permisos de ubicación, uso de Google Maps Compose y lógica condicional en Kotlin.  
A pesar de no contar con la API Key activa, la aplicación funcionó correctamente en el entorno de pruebas y cumplió con todos los objetivos planteados para esta etapa del módulo.

# **Bibliografía**

Android Developers. (2024). Maps SDK for Android (Compose). Google. https://developer.android.com/

Google Cloud. (2024). Location and Maps APIs Overview. Google Cloud Documentation.

AIEP. (2025). Taller de Aplicaciones Móviles – Guía Semana 9. Plataforma AIEP Virtual.

# **Anexo**

https://github.com/Gorilacl/Actividad6