

N1-U2 Actividad práctica sumativa - Semana 6

**MÓDULO: TALLER DE APLICACIONES MÓVILES**

**SEMANA: Semana 6**

Docente: IVAN AYALA

Estudiante: Andres Lagos

**Resumen**

Se desarrolló una aplicación móvil Android que calcula la distancia entre el dispositivo del cliente y la bodega utilizando la fórmula de Haversine, aplica reglas de negocio para determinar el costo de despacho y utiliza Firebase Authentication (correo/contraseña y Google) como SSO. Al validar credenciales, la app captura la ubicación GPS y la guarda en Firebase Realtime Database, permitiendo trazabilidad básica del despacho. El proyecto es compatible con Android Lollipop (minSdk 21) y se probó principalmente en Android Oreo y superior. Se documentó la configuración, el proceso de compilación, las pruebas y se publica el código en GitHub, con respaldo en OneDrive.

Índice

[Introducción 4](#_Toc209467430)

[Desarrollo 5](#_Toc209467431)

[Conclusión 12](#_Toc209467432)

[Anexo 13](#_Toc209467433)

[Bibliografía 29](#_Toc209467434)

# **Introducción**

El documento describe el proceso de construcción de una app de distribución de alimentos que incluye cálculo de distancia, costo de envío y autenticación con Google y correo/clave. Se busca integrar conceptos de geolocalización, reglas de negocio y servicios en la nube (Firebase) en una solución funcional y reproducible, apta para equipos con Android Lollipop (administrador) y versiones más recientes (clientes).

# **Desarrollo**

**a) Pantalla Login con autenticación Firebase (correo/contraseña)**

Diseñe una **LoginActivity** en Jetpack Compose con:

Campo correo y contraseña (con botón de mostrar/ocultar).

Botones **Entrar**, **Crear cuenta** y **Continuar con Google** (opcional).

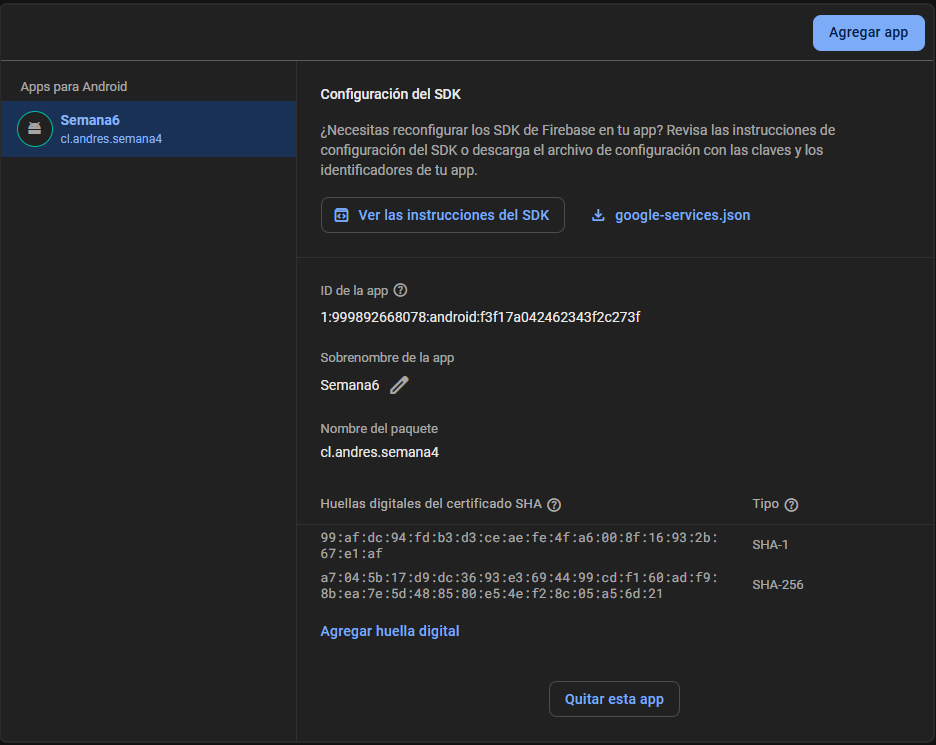
Validaciones: no se permite invocar Firebase si los campos están vacíos; se muestra un **snackbar/toast** indicando que deben completarse (evita el crash por parámetros nulos).

Flujo implementado:

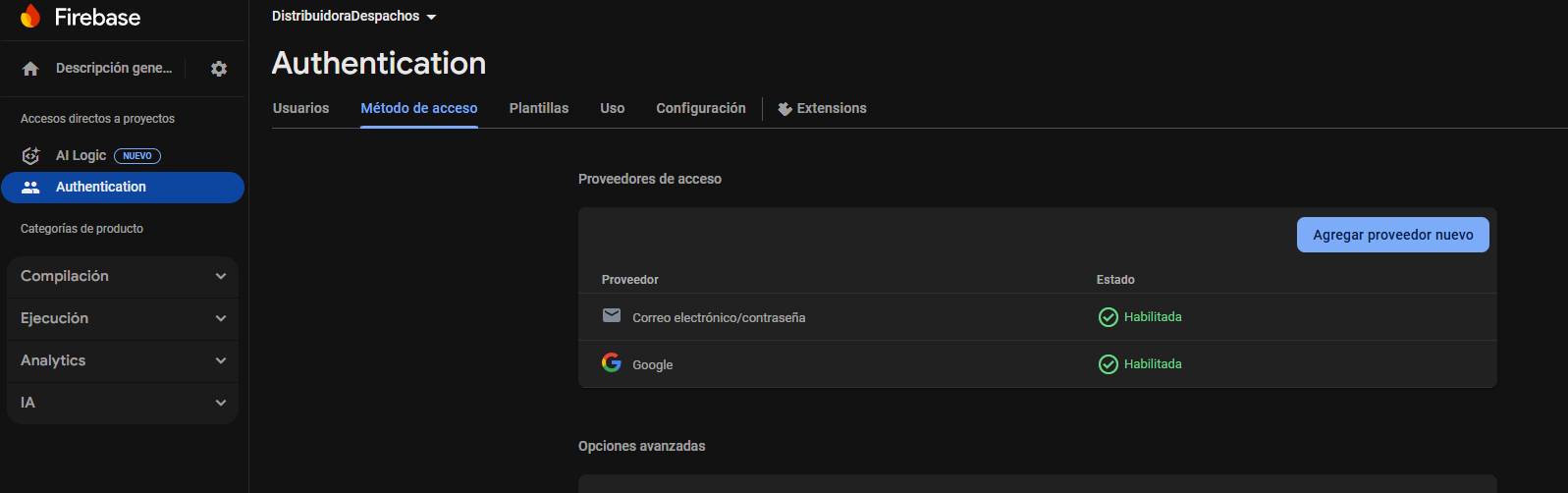
**Crear cuenta:** FirebaseAuth.getInstance().createUserWithEmailAndPassword(email, pass)  
si es exitoso, se deja logueado al usuario y se navega a **MenuActivity**.

**Entrar:** FirebaseAuth.getInstance().signInWithEmailAndPassword(email, pass)  
 si las credenciales son correctas, se abre **MenuActivity.**

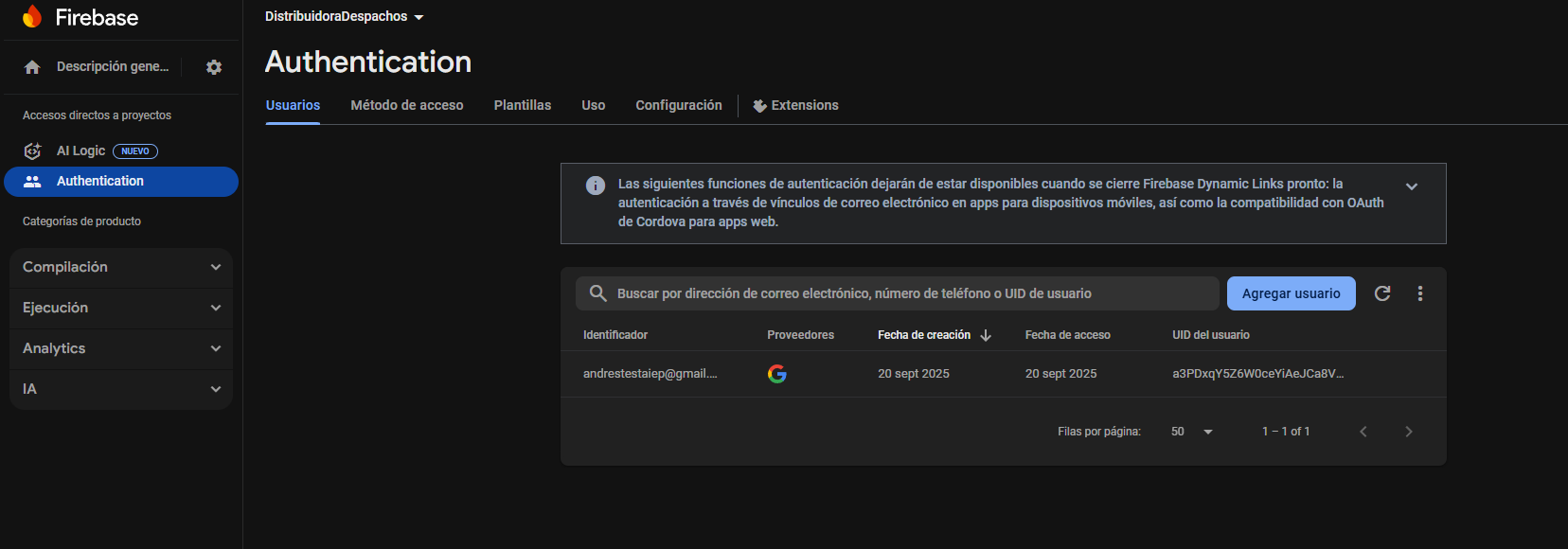
**Evidencias firebase:  
firebase google-service sha-1 sha-256:**



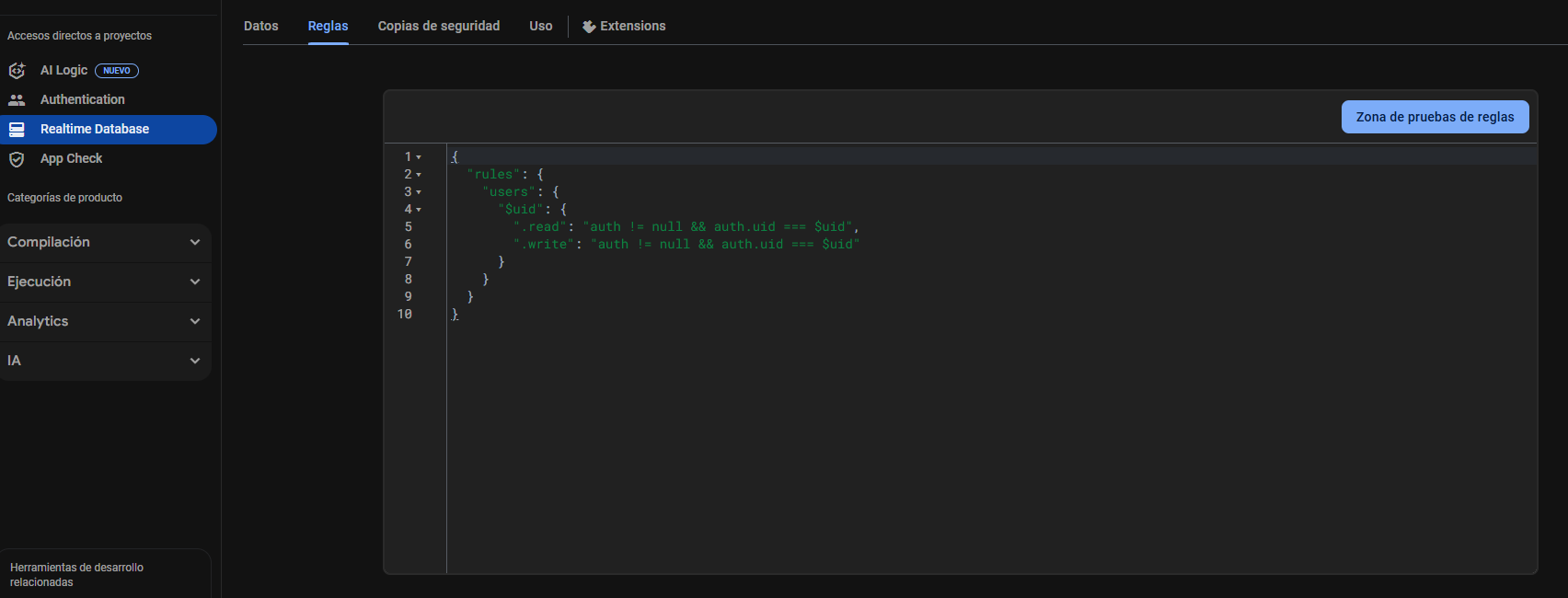
**Metodo de acceso habilitado:**



**Authentication:**



**Reglas realtime database:**



**b) Guardar posición GPS en Realtime Database si el login es válido**

En MenuActivity implemente el guardado de la última ubicación del usuario autenticado:

Al iniciar la actividad (onStart()), pedimos permisos ACCESS\_FINE\_LOCATION/ACCESS\_COARSE\_LOCATION.  
Si están concedidos → se llama a capturarYGuardarUbicacion().  
Se obtiene la ubicación con FusedLocationProviderClient:

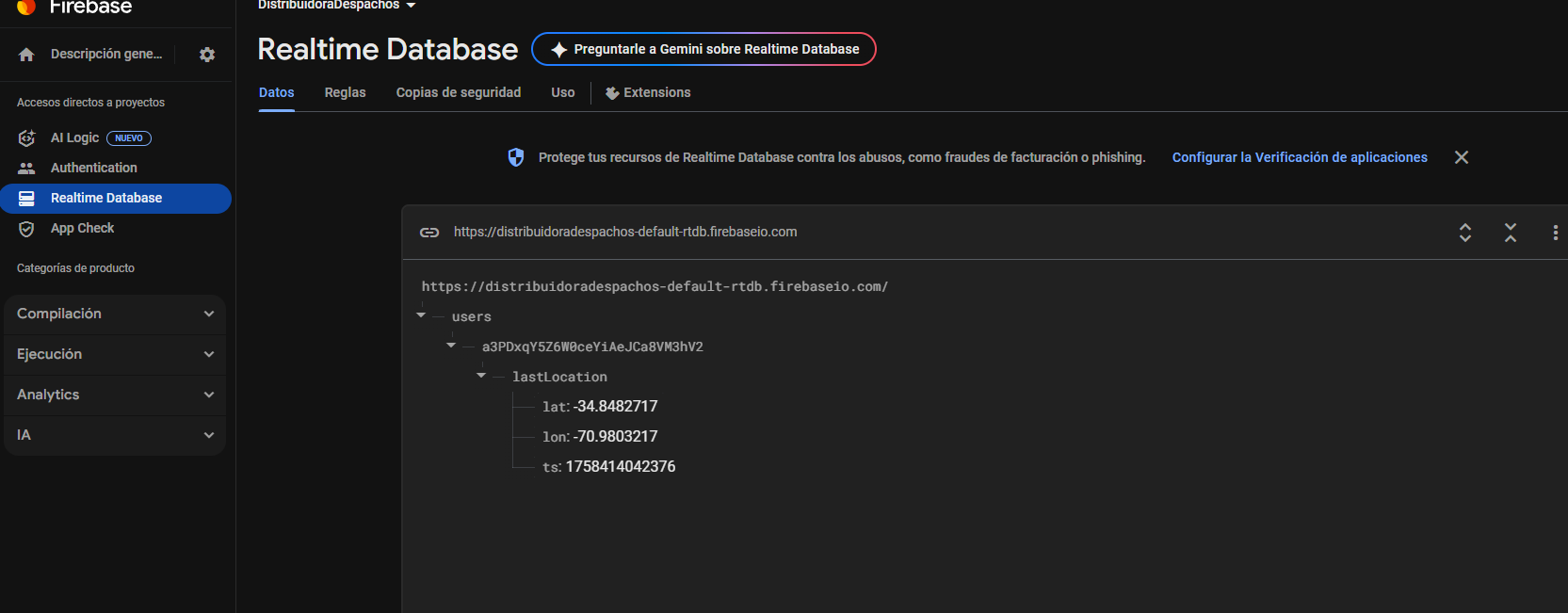
used.getCurrentLocation(Priority.*PRIORITY\_HIGH\_ACCURACY*, null)

Si hay ubicación y uid != null, se escribe en **Realtime Database**:

FirebaseDatabase.getInstance()  
 .getReference("users")  
 .child(uid)  
 .child("lastLocation")  
  
val data = *mapOf*(  
 "lat" *to* loc.*latitude*,  
 "lon" *to* loc.*longitude*,  
 "ts" *to* ServerValue.*TIMESTAMP*)

**Evidencia:**

**Captura de** **Realtime Database:**



**c) Instalación y prueba en dispositivo/emulador**

**Emulador recomendado:** AVD con **Google APIs** (ej. Pixel 6, API 30+ con Google Play).

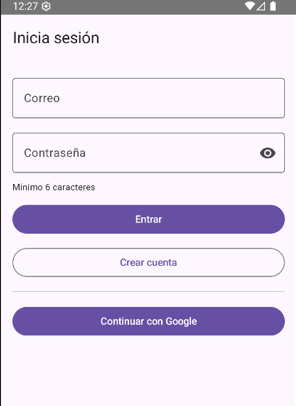
Ingresar a Play Store del emulador (confirma que tiene servicios de Google).  
Ajustar ubicación en **Extended controls → Location** para simular coordenadas.

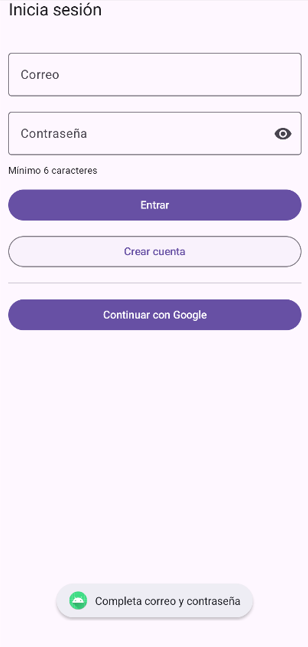
Compilaciónyejecución**:**

Sync Project with Gradle Files.  
Runseleccionando el dispositivo/emulador.

Probar:  
Login con correo/clave válidos.  
Ver que, al abrir MenuActivity, se escribe la ubicación en la DB.  
Repetir negando permisos para verificar el toast de rechazo.

**Evidencia:**

**Login app:**   
  


**Crear cuenta sin completar campos:**  
  


**Menú app:**



# **Conclusión**

El proyecto integró de forma práctica Android con Kotlin, Jetpack Compose y Firebase. Se implementó login con correo/contraseña, registro de la ubicación del usuario en Realtime Database y la lógica de negocio (radianes, Haversine y cálculo de despacho), manteniendo compatibilidad con minSdk 21. Además, el uso de GitHub con issues y ramas por feature mejoró la organización y el trabajo colaborativo. En conjunto, se reforzaron competencias técnicas y de gestión aplicables a proyectos móviles reales.

# **Anexo**

GitHub

<https://github.com/Gorilacl/Activididad6.git>

códigos

**build.gradle.kts (:app)**

plugins **{** id("com.android.application")  
 id("org.jetbrains.kotlin.android")  
 id("org.jetbrains.kotlin.plugin.compose")  
 id("com.google.gms.google-services")  
**}***android* **{** namespace = "cl.andres.semana4"  
 compileSdk = 34  
  
 defaultConfig **{** applicationId = "cl.andres.semana4"  
 minSdk = 21  
 targetSdk = 34  
 versionCode = 1  
 versionName = "1.0"  
 **}** buildTypes **{** *release* **{** isMinifyEnabled = false  
 proguardFiles(  
 getDefaultProguardFile("proguard-android-optimize.txt"),  
 "proguard-rules.pro"  
 )  
 **}  
 }** compileOptions **{** sourceCompatibility = JavaVersion.*VERSION\_17* targetCompatibility = JavaVersion.*VERSION\_17* **}** *kotlinOptions* **{** jvmTarget = "17" **}** // Compose habilitado  
 buildFeatures **{** compose = true **}  
  
}***kotlin* **{** jvmToolchain(17) **}***dependencies* **{** // Compose BOM  
 *implementation*(platform("androidx.compose:compose-bom:2024.10.00"))  
  
 // Núcleo Compose  
 *implementation*("androidx.activity:activity-compose:1.9.2")  
 *implementation*("androidx.compose.ui:ui")  
 *implementation*("androidx.compose.ui:ui-tooling-preview")  
 *implementation*("androidx.compose.material3:material3")  
  
 // Iconos (Visibility / VisibilityOff)  
 *implementation*("androidx.compose.material:material-icons-extended")  
  
 *implementation*("androidx.core:core-ktx:1.13.1")  
  
 // Ubicación / Google  
 *implementation*("com.google.android.gms:play-services-location:21.2.0")  
 *implementation*("com.google.android.gms:play-services-auth:20.7.0")  
  
 // Firebase  
 *implementation*("com.google.firebase:firebase-auth-ktx:22.3.1")  
 *implementation*("com.google.firebase:firebase-database-ktx:20.2.2")  
  
 // Debug  
 *debugImplementation*("androidx.compose.ui:ui-tooling")  
 *debugImplementation*("androidx.compose.ui:ui-test-manifest")  
**}**

**build.gradle.kts (PRO401Semana4)**

plugins **{** id("com.android.application") *version* "8.5.2" *apply* false  
 id("org.jetbrains.kotlin.android") *version* "2.0.0" *apply* false  
 id("org.jetbrains.kotlin.plugin.compose") *version* "2.0.0" *apply* false  
 id("com.google.gms.google-services") *version* "4.4.2" *apply* false  
**}**

**AndroidManifest.xml**

<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android">  
  
 <!-- Permisos -->  
 <uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />  
 <uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />  
 <uses-permission android:name="android.permission.INTERNET" />  
 <uses-permission android:name="android.permission.POST\_NOTIFICATIONS" />  
  
 <uses-feature android:name="android.hardware.location.gps" android:required="false" />  
 <uses-feature android:name="android.hardware.location.network" android:required="false" />  
  
 <application  
 android:allowBackup="true"  
 android:dataExtractionRules="@xml/data\_extraction\_rules"  
 android:fullBackupContent="@xml/backup\_rules"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/Theme.PRO401Semana4">  
  
 <!-- Pantalla de inicio -->  
 <activity  
 android:name=".LoginActivity"  
 android:exported="true">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
 </activity>  
  
 <!-- Menú post-login -->  
 <activity  
 android:name=".MenuActivity"  
 android:exported="true" />  
  
 <!-- Tu app principal con conversor/ despacho/ distancia -->  
 <activity  
 android:name=".MainActivity"  
 android:exported="true" />  
  
 </application>  
</manifest>

**LoginActivity.kt**

package cl.andres.semana4  
  
import android.content.Intent  
import android.os.Bundle  
import android.util.Log  
import android.widget.Toast  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
import androidx.activity.result.contract.ActivityResultContracts  
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.platform.*LocalContext*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  
import com.google.android.gms.auth.api.signin.GoogleSignIn  
import com.google.android.gms.auth.api.signin.GoogleSignInOptions  
import com.google.android.gms.common.ConnectionResult  
import com.google.android.gms.common.GoogleApiAvailability  
import com.google.firebase.auth.FirebaseAuth  
import com.google.firebase.auth.GoogleAuthProvider  
  
class LoginActivity : ComponentActivity() {  
  
 private val auth by *lazy* **{** FirebaseAuth.getInstance() **}** // Launcher para el intent de Google  
 private val googleLauncher = registerForActivityResult(  
 ActivityResultContracts.StartActivityForResult()  
 ) **{** result **->** val task = GoogleSignIn.getSignedInAccountFromIntent(result.data)  
 try {  
 val account = task.*result* val credential = GoogleAuthProvider.getCredential(account.*idToken*, null)  
 auth.signInWithCredential(credential).addOnCompleteListener **{** t **->** if (t.*isSuccessful*) {  
 Toast.makeText(this, "Login con Google OK", Toast.*LENGTH\_SHORT*).show()  
 goToMenu()  
 } else {  
 Toast.makeText(  
 this,  
 t.*exception*?.*localizedMessage* ?: "Error Google",  
 Toast.*LENGTH\_LONG* ).show()  
 Log.e("Auth", "Google sign-in error", t.*exception*)  
 }  
 **}** } catch (e: Exception) {  
 Toast.makeText(this, "Cancelado o error: ${e.*localizedMessage*}", Toast.*LENGTH\_LONG*)  
 .show()  
 Log.e("Auth", "Google intent error", e)  
 }  
 **}** override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *setContent* **{** LoginScreen(  
 onEmailLogin = **{** email, pass **->** signInEmail(email, pass) **}**,  
 onRegister = **{** email, pass **->** registerEmail(email, pass) **}**,  
 onGoogle = **{** Toast.makeText(this, "Abriendo Google…", Toast.*LENGTH\_SHORT*).show()  
 startGoogleSignIn()  
 **}** ) **}** }  
  
 private fun canUsePlayServices(): Boolean {  
 val api = GoogleApiAvailability.getInstance()  
 val code = api.isGooglePlayServicesAvailable(this)  
 if (code != ConnectionResult.*SUCCESS*) {  
 if (api.isUserResolvableError(code)) {  
 api.getErrorDialog(this, code, /\*requestCode\*/9000)?.show()  
 } else {  
 Toast.makeText(this, "Google Play Services no disponible", Toast.*LENGTH\_LONG*).show()  
 }  
 return false  
 }  
 return true  
 }  
  
 private fun startGoogleSignIn() {  
 if (!canUsePlayServices()) return  
 val gso = GoogleSignInOptions.Builder(GoogleSignInOptions.*DEFAULT\_SIGN\_IN*)  
 .requestIdToken(getString(R.string.*default\_web\_client\_id*)) // generado por google-services  
 .requestEmail()  
 .build()  
 val client = GoogleSignIn.getClient(this, gso)  
 googleLauncher.launch(client.*signInIntent*)  
 }  
  
 private fun goToMenu() {  
 startActivity(Intent(this, MenuActivity::class.*java*))  
 finish()  
 }  
  
 // ---------- Email/Password ----------  
 private fun signInEmail(email: String, pass: String) {  
 if (email.*isBlank*() || pass.*isBlank*()) {  
 Toast.makeText(this, "Completa correo y contraseña", Toast.*LENGTH\_SHORT*).show()  
 return  
 }  
 auth.signInWithEmailAndPassword(email, pass).addOnCompleteListener **{** t **->** if (t.*isSuccessful*) goToMenu()  
 else Toast.makeText(  
 this, t.*exception*?.*localizedMessage* ?: "Error al iniciar",  
 Toast.*LENGTH\_LONG* ).show()  
 **}** }  
  
 private fun registerEmail(email: String, pass: String) {  
 if (email.*isBlank*() || pass.*isBlank*()) {  
 Toast.makeText(this, "Completa correo y contraseña", Toast.*LENGTH\_SHORT*).show()  
 return  
 }  
 if (pass.length < 6) {  
 Toast.makeText(this, "La contraseña debe tener al menos 6 caracteres", Toast.*LENGTH\_SHORT*).show()  
 return  
 }  
 auth.createUserWithEmailAndPassword(email, pass).addOnCompleteListener **{** t **->** if (t.*isSuccessful*) {  
 Toast.makeText(this, "Cuenta creada, iniciando sesión…", Toast.*LENGTH\_SHORT*).show()  
 goToMenu()  
 } else {  
 Toast.makeText(  
 this, t.*exception*?.*localizedMessage* ?: "No se pudo crear la cuenta",  
 Toast.*LENGTH\_LONG* ).show()  
 }  
 **}** }  
}  
  
@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
fun LoginScreen(  
 onEmailLogin: (String, String) -> Unit,  
 onRegister: (String, String) -> Unit,  
 onGoogle: () -> Unit  
) {  
 var email by remember **{** *mutableStateOf*("") **}** var pass by remember **{** *mutableStateOf*("") **}** var passVisible by remember **{** *mutableStateOf*(false) **}** val context = *LocalContext*.current  
  
 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 }**,  
 label = **{** Text("Correo") **}**,  
 singleLine = true,  
 keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Email),  
 modifier = Modifier.*fillMaxWidth*()  
 )  
  
 OutlinedTextField(  
 value = pass,  
 onValueChange = **{** pass = **it }**,  
 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*()  
 )  
 Text("Mínimo 6 caracteres", style = MaterialTheme.typography.bodySmall)  
  
 Button(  
 onClick = **{** onEmailLogin(email.*trim*(), pass) **}**,  
 modifier = Modifier.*fillMaxWidth*(),  
 shape = MaterialTheme.shapes.extraLarge  
 ) **{** Text("Entrar") **}** OutlinedButton(  
 onClick = **{** onRegister(email.*trim*(), pass) **}**,  
 modifier = Modifier.*fillMaxWidth*(),  
 shape = MaterialTheme.shapes.extraLarge  
 ) **{** Text("Crear cuenta") **}** Divider(Modifier.*padding*(vertical = 4.*dp*))  
  
 Button(  
 onClick = onGoogle,  
 modifier = Modifier.*fillMaxWidth*(),  
 shape = MaterialTheme.shapes.extraLarge  
 ) **{** Text("Continuar con Google") **}  
 }  
 }**}

**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.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.auth.FirebaseAuth  
import com.google.firebase.database.FirebaseDatabase  
import com.google.firebase.database.ServerValue  
  
class MenuActivity : ComponentActivity() {  
  
 private lateinit var fused: FusedLocationProviderClient  
  
 // Launcher para solicitar permisos de ubicación  
 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 = **{** FirebaseAuth.getInstance().signOut()  
 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()  
 // guarda la ubicación apenas se abre 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* )  
 )  
 }  
 }  
  
 // chequeo de permisos manualmente  
 @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  
 }  
  
 val uid = FirebaseAuth.getInstance().*currentUser*?.*uid* ?: return  
  
 try {  
 fused.getCurrentLocation(Priority.*PRIORITY\_HIGH\_ACCURACY*, null)  
 .addOnSuccessListener **{** loc **->** if (loc != null) {  
 val ref = FirebaseDatabase.getInstance()  
 .getReference("users")  
 .child(uid)  
 .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") **}  
 }  
 }**}

**Main.Activity.kt**

package cl.andres.semana4  
  
import android.Manifest  
import android.content.pm.PackageManager  
import android.os.Bundle  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
import androidx.activity.compose.rememberLauncherForActivityResult  
import androidx.activity.result.contract.ActivityResultContracts  
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 kotlin.math.\*  
import java.text.NumberFormat  
import java.util.Locale  
  
// Plaza de Armas  
private const val *BODEGA\_LAT* = -35.016  
private const val *BODEGA\_LON* = -71.333  
  
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)  
  
private const val *RADIO\_GRATIS\_KM* = 20.0  
  
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)  
}  
  
class MainActivity : ComponentActivity() {  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *setContent* **{** AppScreen() **}** val grados = 90.0  
 val rad = *gradosARadianes*(grados)  
 *println*("$grados grados = $rad radianes")  
 }  
}  
  
@OptIn(ExperimentalMaterial3Api::class)  
@Composable  
fun AppScreen() {  
 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 **}** )  
 **}  
 }**}  
  
@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  
 *println*("$valor grados = $rad radianes")  
 }  
 **}**,  
 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: ${*formatoCLP*(r.costoDespacho)}"  
 }  
 *println*("DEBUG despacho → total=$total, km=$km, costo=${r.costoDespacho}, gratis=${r.aplicaGratis}")  
 }  
 **}**,  
 modifier = Modifier.*fillMaxWidth*(),  
 shape = MaterialTheme.shapes.extraLarge  
 ) **{** Text("Calcular despacho") **}** if (mensaje.*isNotBlank*()) Text(mensaje)  
}

# **Bibliografía**

Firebase. (s. f.). **Autenticación con correo y contraseña en Android**. Recuperado el 20 de septiembre de 2025, de https://firebase.google.com/docs/auth/android/password-auth  
Firebase. (s. f.). **Configurar Realtime Database para Android**. Recuperado el 20 de septiembre de 2025, de https://firebase.google.com/docs/database/android/start  
Firebase. (s. f.). **Reglas de seguridad de Realtime Database**. Recuperado el 20 de septiembre de 2025, de https://firebase.google.com/docs/rules