**Capítulo 5 – ML Kit**

Código **AndroidManifest.xml**:

*<?*xml version="1.0" encoding="utf-8"*?>*<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools">  
  
 *<!-- Acceso a la red -->* <uses-permission android:name="android.permission.INTERNET" />  
  
 *<!-- Permisos -->* <uses-permission android:name="android.permission.CAMERA" />  
 <uses-feature android:name="android.hardware.camera" android:required="false"/>  
 <uses-permission android:name="android.permission.RECORD\_AUDIO" />  
  
  
 <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.Capitulo5\_practica1"  
 tools:targetApi="31">  
 <activity  
 android:name=".MainActivity"  
 android:exported="true"  
 android:label="@string/app\_name"  
 android:theme="@style/Theme.Capitulo5\_practica1">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
 </activity>  
 </application>  
  
</manifest>

Código en **build.gradle.kts**(Module: App):

plugins **{** alias(*libs*.*plugins*.*android*.*application*)  
 alias(*libs*.*plugins*.*kotlin*.*android*)  
 alias(*libs*.*plugins*.*kotlin*.*compose*)  
**}***android* **{** namespace = "com.example.capitulo5\_practica1"  
 compileSdk = 35  
  
 defaultConfig **{** applicationId = "com.example.capitulo5\_practica1"  
 minSdk = 35  
 targetSdk = 35  
 versionCode = 1  
 versionName = "1.0"  
  
 testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"  
 **}** buildTypes **{** *release* **{** isMinifyEnabled = false  
 proguardFiles(  
 getDefaultProguardFile("proguard-android-optimize.txt"),  
 "proguard-rules.pro"  
 )  
 **}  
 }** compileOptions **{** sourceCompatibility = JavaVersion.*VERSION\_11* targetCompatibility = JavaVersion.*VERSION\_11* **}** *kotlinOptions* **{** jvmTarget = "11"  
 **}** buildFeatures **{** compose = true  
 **}  
}***dependencies* **{** *implementation*(*libs*.*androidx*.*core*.*ktx*)  
 *implementation*(*libs*.*androidx*.*lifecycle*.*runtime*.*ktx*)  
 *implementation*(*libs*.*androidx*.*activity*.*compose*)  
 *implementation*(platform(*libs*.*androidx*.*compose*.*bom*))  
 *implementation*(*libs*.*androidx*.*ui*)  
 *implementation*(*libs*.*androidx*.*ui*.*graphics*)  
 *implementation*(*libs*.*androidx*.*ui*.*tooling*.*preview*)  
 *implementation*(*libs*.*androidx*.*material3*)  
 *testImplementation*(*libs*.*junit*)  
 *androidTestImplementation*(*libs*.*androidx*.*junit*)  
 *androidTestImplementation*(*libs*.*androidx*.*espresso*.*core*)  
 *androidTestImplementation*(platform(*libs*.*androidx*.*compose*.*bom*))  
 *androidTestImplementation*(*libs*.*androidx*.*ui*.*test*.*junit4*)  
 *debugImplementation*(*libs*.*androidx*.*ui*.*tooling*)  
 *debugImplementation*(*libs*.*androidx*.*ui*.*test*.*manifest*)  
  
 *implementation* ("com.google.accompanist:accompanist-permissions:0.34.0")  
  
 *implementation* ("com.google.mlkit:text-recognition:16.0.1")  
  
 *implementation* ("org.jetbrains.kotlinx:kotlinx-coroutines-play-services:1.7.3")  
  
 *// CameraX core library using the camera2 implementation  
 // The following line is optional, as the core library is included indirectly by camera-camera2  
 implementation* ("androidx.camera:camera-core:1.4.2")  
 *implementation* ("androidx.camera:camera-camera2:1.4.2")  
 *// If you want to additionally use the CameraX Lifecycle library  
 implementation* ("androidx.camera:camera-lifecycle:1.4.2")  
 *// If you want to additionally use the CameraX VideoCapture library  
 implementation* ("androidx.camera:camera-video:1.4.2")  
 *// If you want to additionally use the CameraX View class  
 implementation* ("androidx.camera:camera-view:1.4.2")  
 *// If you want to additionally add CameraX ML Kit Vision Integration  
 implementation* ("androidx.camera:camera-mlkit-vision:1.4.2")  
 *// If you want to additionally use the CameraX Extensions library  
 implementation* ("androidx.camera:camera-extensions:1.4.2")  
  
**}**

**Configuración inicial y gestión de permisos**

@OptIn(ExperimentalPermissionsApi::class)

class MainActivity : ComponentActivity() {

override fun onCreate(savedInstanceState: Bundle?) {

*// ...*

val multiplePermissionsState = rememberMultiplePermissionsState(

listOf(

android.Manifest.permission.RECORD\_AUDIO,

android.Manifest.permission.CAMERA,

)

)

*// ...*

}

}

**Controlador de cámara y estado de la aplicación**

val context = LocalContext.current

val cameraController = remember {

LifecycleCameraController(context)

}

var capturedImage by remember { mutableStateOf<ImageBitmap?>(null) }

var recognizedText by remember { mutableStateOf<String?>(null) }

val coroutineScope = rememberCoroutineScope()

**Estructura de la interfaz**

Column(

modifier = Modifier

.fillMaxSize()

.padding(16.dp)

) {

*// Tarjetas para: permisos, cámara, imagen capturada*

}

**Captura de imágenes y reconocimiento de texto**

CameraPreviewComposable2(

cameraController = cameraController,

lifecycleOwner = this@MainActivity

) { imageBitmap ->

capturedImage = imageBitmap

coroutineScope.launch(Dispatchers.Default) {

recognizedText = recognizeText(imageBitmap, context)

}

}

**Vista previa de la cámara**

@Composable

fun CameraPreviewComposable2(

cameraController: LifecycleCameraController,

lifecycleOwner: LifecycleOwner,

onImageCaptured: (ImageBitmap) -> Unit

) {

*// ...*

}

**Toma de fotografías**

Button(

onClick = {

val mainExecutor = ContextCompat.getMainExecutor(context)

cameraController.takePicture(executor, object : ImageCapture.OnImageCapturedCallback() {

override fun onCaptureSuccess(image: ImageProxy) {

val bitmap = image.toBitmap()

val imageBitmap = bitmap.asImageBitmap()

mainExecutor.execute {

onImageCaptured(imageBitmap)

}

image.close()

}

*// ...*

})

},

modifier = Modifier.align(Alignment.BottomCenter)

) {

Text("Capturar")

}

**Procesamiento de imágenes y reconocimiento de texto**

fun ImageProxy.toBitmap(): Bitmap {

val buffer = planes[0].buffer

buffer.rewind()

val bytes = ByteArray(buffer.capacity())

buffer.get(bytes)

return BitmapFactory.decodeByteArray(bytes, 0, bytes.size)

}

private suspend fun recognizeText(imageBitmap: ImageBitmap, context: Context): String {

return try {

val recognizer = TextRecognition.getClient(TextRecognizerOptions.DEFAULT\_OPTIONS)

val image = InputImage.fromBitmap(imageBitmap.asAndroidBitmap(), 0)

val result = recognizer.process(image).await()

result.text

} catch (e: Exception) {

"Error en el reconocimiento de texto: ${e.message}"

}

}

Código **MainActivity.kt**:

package com.example.capitulo5\_practica1  
  
import android.content.Context  
import android.graphics.Bitmap  
import android.graphics.BitmapFactory  
import android.os.Bundle  
import android.util.Log  
import android.view.ViewGroup  
import android.widget.Toast  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
import androidx.activity.enableEdgeToEdge  
import androidx.camera.core.ImageCapture  
import androidx.camera.core.ImageCaptureException  
import androidx.camera.core.ImageProxy  
import androidx.camera.view.LifecycleCameraController  
import androidx.camera.view.PreviewView  
import androidx.compose.foundation.Image  
import androidx.compose.foundation.layout.Box  
import androidx.compose.foundation.layout.Column  
import androidx.compose.foundation.layout.Spacer  
import androidx.compose.foundation.layout.aspectRatio  
import androidx.compose.foundation.layout.fillMaxSize  
import androidx.compose.foundation.layout.fillMaxWidth  
import androidx.compose.foundation.layout.height  
import androidx.compose.foundation.layout.padding  
import androidx.compose.material3.Button  
import androidx.compose.material3.Card  
import androidx.compose.material3.Text  
import androidx.compose.runtime.Composable  
import androidx.compose.runtime.getValue  
import androidx.compose.runtime.mutableStateOf  
import androidx.compose.runtime.remember  
import androidx.compose.runtime.rememberCoroutineScope  
import androidx.compose.runtime.setValue  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.graphics.ImageBitmap  
import androidx.compose.ui.graphics.asAndroidBitmap  
import androidx.compose.ui.graphics.asImageBitmap  
import androidx.compose.ui.platform.*LocalContext*import androidx.compose.ui.unit.dp  
import androidx.compose.ui.viewinterop.AndroidView  
import androidx.core.content.ContextCompat  
import androidx.lifecycle.LifecycleOwner  
import com.google.accompanist.permissions.ExperimentalPermissionsApi  
import com.google.accompanist.permissions.PermissionState  
import com.google.accompanist.permissions.rememberMultiplePermissionsState  
import com.google.mlkit.vision.common.InputImage  
import com.google.mlkit.vision.text.TextRecognition  
import com.google.mlkit.vision.text.latin.TextRecognizerOptions  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.launch  
import kotlinx.coroutines.tasks.await  
import java.util.concurrent.Executors  
import com.example.capitulo5\_practica1.ui.theme.Capitulo5\_practica1Theme  
  
*// La anotación @OptIn indica que estamos usando una API experimental - en este caso la API de Permisos*@OptIn(ExperimentalPermissionsApi::class)  
*// Clase principal de la actividad - el punto de entrada a nuestra aplicación*class MainActivity : ComponentActivity() {  
 *// onCreate se llama cuando la actividad se crea por primera vez - como un constructor para la actividad* override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 *// Habilita la visualización de borde a borde (usa la pantalla completa incluyendo áreas alrededor de notches)  
 enableEdgeToEdge*()  
 *// Establece el contenido de nuestra actividad usando Jetpack Compose  
 setContent* **{** *// Aplica el tema definido para nuestra aplicación* Capitulo5\_practica1Theme **{** *// Crea un objeto de estado para gestionar múltiples permisos (cámara y audio)  
 // Esto usa la biblioteca de permisos Accompanist para simplificar el manejo de permisos* val multiplePermissionsState = rememberMultiplePermissionsState(  
 *listOf*(  
 android.Manifest.permission.*RECORD\_AUDIO*,  
 android.Manifest.permission.*CAMERA*,  
 )  
 )  
  
 *// Obtiene el contexto actual (necesario para interacciones con el sistema Android)* val context = *LocalContext*.current  
 *// Crea y recuerda un controlador de cámara - recordarlo evita la recreación durante la recomposición* val cameraController = remember **{** LifecycleCameraController(context)  
 **}** *// Columna contenedora principal que sostiene todos nuestros elementos de UI verticalmente* Column(  
 modifier = Modifier  
 .*fillMaxSize*() *// Usa todo el espacio disponible* .*padding*(16.*dp*) *// Añade relleno alrededor de todos los bordes* ) **{** *// Primera Card - Tarjeta de solicitud de permisos* Card **{** *// Muestra texto explicando qué permisos se necesitan* Text(  
 *getTextToShowGivenPermissions*(  
 multiplePermissionsState.revokedPermissions,  
 multiplePermissionsState.shouldShowRationale  
 )  
 )  
 *// Añade espacio vertical* Spacer(modifier = Modifier.*height*(8.*dp*))  
 *// Botón para solicitar permisos* Button(onClick = **{** multiplePermissionsState.launchMultiplePermissionRequest() **}**) **{** Text("Request permissions")  
 **}  
 }** *// Variables de estado para almacenar la imagen capturada y el texto reconocido  
 // mutableStateOf crea objetos de estado observables que desencadenan recomposición cuando cambian* var capturedImage by remember **{** *mutableStateOf*<ImageBitmap?>(null) **}** var recognizedText by remember **{** *mutableStateOf*<String?>(null) **}** *// Crea un ámbito de corrutinas vinculado al ciclo de vida de este composable* val coroutineScope = rememberCoroutineScope()  
  
 *// Segunda Card - Vista previa de la cámara y botón de captura* Card() **{** *// Composable personalizado que muestra la vista previa de la cámara y maneja la captura de imágenes* CameraPreviewComposable2(  
 cameraController = cameraController,  
 lifecycleOwner = this@MainActivity *// Pasa la actividad como propietario del ciclo de vida* ) **{** imageBitmap **->** *// Esta lambda se llama cuando se captura una imagen* capturedImage = imageBitmap  
 *// Inicia el proceso de reconocimiento de texto en un hilo de fondo* coroutineScope.*launch*(Dispatchers.Default) **{** recognizedText = recognizeText(imageBitmap, context)  
 **}  
 }  
 }** *// Tercera Card - Muestra la imagen capturada* Card **{** *// Usa el operador de llamada segura (?.) para renderizar la imagen solo si capturedImage no es null* capturedImage?.*let* **{** bitmap **->** Image(  
 bitmap = bitmap,  
 contentDescription = "Captured image",  
 modifier = Modifier.*fillMaxSize*()  
 )  
 **}  
 }** *// Muestra el texto reconocido en una notificación Toast  
 // Nuevamente usando el operador de llamada segura para mostrar solo si recognizedText no es null* recognizedText?.*let* **{** text **->** *//Text(text = "Texto reconocido: $text") // Comentado - se mostraría en pantalla* Toast.makeText(context, "Texto reconocido: $text", Toast.*LENGTH\_SHORT*).show()  
 **}  
  
 }***//Column()* **}  
 }** }  
}  
  
  
*// Función auxiliar para generar texto explicando el estado de los permisos  
// Anotación @OptIn nuevamente para la API experimental de Permisos*@OptIn(ExperimentalPermissionsApi::class)  
private fun getTextToShowGivenPermissions(  
 permissions: List<PermissionState>, *// Lista de permisos que están revocados* shouldShowRationale: Boolean *// Si debemos mostrar explicación de por qué necesitamos permisos*): String {  
 *// Si no hay permisos revocados, devuelve cadena vacía* val revokedPermissionsSize = permissions.size  
 if (revokedPermissionsSize == 0) return ""  
  
 *// Usa StringBuilder para concatenación eficiente de cadenas* val textToShow = StringBuilder().*apply* **{** append("The ")  
 **}** *// Recorre todos los permisos revocados y los formatea como una lista separada por comas* for (i in permissions.*indices*) {  
 textToShow.append(permissions[i].permission)  
 when {  
 *// Caso especial para el penúltimo elemento (añade ", and ")* revokedPermissionsSize > 1 && i == revokedPermissionsSize - 2 -> {  
 textToShow.append(", and ")  
 }  
 *// Para el último elemento, solo añade un espacio* i == revokedPermissionsSize - 1 -> {  
 textToShow.append(" ")  
 }  
 *// Para todos los demás elementos, añade una coma y un espacio* else -> {  
 textToShow.append(", ")  
 }  
 }  
 }  
  
 *// Añade forma singular o plural basada en el número de permisos* textToShow.append(if (revokedPermissionsSize == 1) "permission is" else "permissions are")  
  
 *// Añade diferente texto final basado en si debemos mostrar la justificación* textToShow.append(  
 if (shouldShowRationale) {  
 *// Si debemos explicar por qué necesitamos permisos* " important. Please grant all of them for the app to function properly."  
 } else {  
 *// Si los permisos fueron explícitamente denegados* " denied. The app cannot function without them."  
 }  
 )  
 return textToShow.toString()  
}  
  
*// Función composable personalizada para la vista previa de la cámara y captura de imágenes  
// La anotación @Composable marca esto como una función UI de Compose*@Composable  
fun CameraPreviewComposable2(  
 cameraController: LifecycleCameraController, *// Controlador para operaciones de cámara* lifecycleOwner: LifecycleOwner, *// Propietario para vincular el ciclo de vida de la cámara* onImageCaptured: (ImageBitmap) -> Unit *// Función de callback cuando se captura una imagen*) {  
 *// Obtiene el contexto actual para llamadas al sistema Android* val context = *LocalContext*.current  
 *// Crea y recuerda un ejecutor de un solo hilo para operaciones de cámara* val executor = remember **{** Executors.newSingleThreadExecutor() **}** *// El layout Box permite posicionar elementos uno encima de otro (como vista previa de cámara con superposición de botón)* Box **{** *// AndroidView es un puente de Compose a las Vistas tradicionales de Android* AndroidView(  
 modifier = Modifier  
 .*fillMaxWidth*() *// Usa todo el ancho disponible* .*aspectRatio*(16f / 9f), *// Mantiene relación de aspecto 16:9 (estándar para cámaras)  
 // La lambda factory crea la Vista de Android (PreviewView para la cámara)* factory = **{** ctx **->** PreviewView(ctx).*apply* **{** *// Establece parámetros de layout para la vista previa  
 layoutParams* = ViewGroup.LayoutParams(  
 ViewGroup.LayoutParams.*WRAP\_CONTENT*,  
 ViewGroup.LayoutParams.*WRAP\_CONTENT* )  
 *// Establece el controlador de cámara para esta vista previa  
 controller* = cameraController  
 **}  
 }** )  
  
 *// Botón de captura posicionado en la parte inferior central del box* Button(  
 onClick = **{** *// Obtiene el ejecutor del hilo principal para actualizaciones de UI* val mainExecutor = ContextCompat.getMainExecutor(context)  
 *// Llama a takePicture en el controlador de cámara* cameraController.takePicture(executor, object : ImageCapture.OnImageCapturedCallback() {  
 *// Esto se llama cuando la captura de imagen es exitosa* override fun onCaptureSuccess(image: ImageProxy) {  
 *// Convierte la imagen capturada a bitmap* val bitmap = image.toBitmap()  
 *// Convierte a ImageBitmap (formato de imagen de Compose)* val imageBitmap = bitmap.*asImageBitmap*()  
 *// Ejecuta en el hilo principal para actualizar UI* mainExecutor.execute **{** *// Llama al callback con la imagen capturada* onImageCaptured(imageBitmap)  
 **}** *// Importante: cierra la imagen para liberar recursos* image.close()  
 }  
  
 *// Maneja errores durante la captura de imagen* override fun onError(exception: ImageCaptureException) {  
 Log.e("CameraPreview", "Error capturing image", exception)  
 }  
 })  
 **}**,  
 modifier = Modifier.*align*(Alignment.BottomCenter) *// Posiciona el botón en la parte inferior central* ) **{** Text("Capturar")  
 **}  
 }** *// Vincula el controlador de cámara al propietario del ciclo de vida para gestionar automáticamente el ciclo de vida de la cámara* cameraController.bindToLifecycle(lifecycleOwner)  
}  
  
*// Función de extensión para convertir ImageProxy (de CameraX) a Bitmap*fun ImageProxy.toBitmap(): Bitmap {  
 *// Obtiene el buffer de imagen del primer plano* val buffer = *planes*[0].*buffer  
 // Posiciona el buffer al principio* buffer.rewind()  
 *// Crea un array de bytes con la capacidad del buffer* val bytes = ByteArray(buffer.capacity())  
 *// Transfiere datos del buffer al array de bytes* buffer.get(bytes)  
 *// Decodifica el array de bytes en un Bitmap* return BitmapFactory.decodeByteArray(bytes, 0, bytes.size)  
}  
  
*// Función para realizar reconocimiento de texto en una imagen  
// La palabra clave suspend indica que esta es una función de corrutina*private suspend fun recognizeText(imageBitmap: ImageBitmap, context: Context): String {  
 return try {  
 *// Crea un reconocedor de texto de ML Kit con opciones predeterminadas* val recognizer = TextRecognition.getClient(TextRecognizerOptions.*DEFAULT\_OPTIONS*)  
 *// Convierte el ImageBitmap de Compose al formato InputImage de ML Kit* val image = InputImage.fromBitmap(imageBitmap.*asAndroidBitmap*(), 0)  
 *// Procesa la imagen y espera el resultado (función suspendida)* val result = recognizer.process(image).await()  
 *// Devuelve el texto reconocido* result.*text* } catch (e: Exception) {  
 *// Si ocurre un error, devuelve un mensaje de error* "Error en el reconocimiento de texto: ${e.message}"  
 }  
}

Código **Androidmanifest.xml**:

*<?*xml version="1.0" encoding="utf-8"*?>*<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools">  
  
 *<!-- Permisos para la cámara -->* <uses-feature android:name="android.hardware.camera" />  
 <uses-permission android:name="android.permission.CAMERA" />  
  
 <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.Capitulo5\_practica2"  
 tools:targetApi="31">  
 <activity  
 android:name=".MainActivity"  
 android:exported="true"  
 android:label="@string/app\_name"  
 android:theme="@style/Theme.Capitulo5\_practica2">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
 </activity>  
 </application>  
  
</manifest>

**Paso 2:** Añadir dependencias en **build.gradle.kts**(Module: App):

Código en **build.gradle.kts**(Module: App):

plugins **{** alias(*libs*.*plugins*.*android*.*application*)  
 alias(*libs*.*plugins*.*kotlin*.*android*)  
 alias(*libs*.*plugins*.*kotlin*.*compose*)  
**}***android* **{** namespace = "com.example.capitulo5\_practica2"  
 compileSdk = 35  
  
 defaultConfig **{** applicationId = "com.example.capitulo5\_practica2"  
 minSdk = 35  
 targetSdk = 35  
 versionCode = 1  
 versionName = "1.0"  
  
 testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"  
 **}** buildTypes **{** *release* **{** isMinifyEnabled = false  
 proguardFiles(  
 getDefaultProguardFile("proguard-android-optimize.txt"),  
 "proguard-rules.pro"  
 )  
 **}  
 }** compileOptions **{** sourceCompatibility = JavaVersion.*VERSION\_11* targetCompatibility = JavaVersion.*VERSION\_11* **}** *kotlinOptions* **{** jvmTarget = "11"  
 **}** buildFeatures **{** compose = true  
 **}  
}***dependencies* **{** *implementation*(*libs*.*androidx*.*core*.*ktx*)  
 *implementation*(*libs*.*androidx*.*lifecycle*.*runtime*.*ktx*)  
 *implementation*(*libs*.*androidx*.*activity*.*compose*)  
 *implementation*(platform(*libs*.*androidx*.*compose*.*bom*))  
 *implementation*(*libs*.*androidx*.*ui*)  
 *implementation*(*libs*.*androidx*.*ui*.*graphics*)  
 *implementation*(*libs*.*androidx*.*ui*.*tooling*.*preview*)  
 *implementation*(*libs*.*androidx*.*material3*)  
 *testImplementation*(*libs*.*junit*)  
 *androidTestImplementation*(*libs*.*androidx*.*junit*)  
 *androidTestImplementation*(*libs*.*androidx*.*espresso*.*core*)  
 *androidTestImplementation*(platform(*libs*.*androidx*.*compose*.*bom*))  
 *androidTestImplementation*(*libs*.*androidx*.*ui*.*test*.*junit4*)  
 *debugImplementation*(*libs*.*androidx*.*ui*.*tooling*)  
 *debugImplementation*(*libs*.*androidx*.*ui*.*test*.*manifest*)  
  
 *// Coil para cargar imágenes  
 implementation*("io.coil-kt:coil-compose:2.4.0")  
  
 *// CameraX  
 implementation*("androidx.camera:camera-camera2:1.4.2")  
 *implementation*("androidx.camera:camera-lifecycle:1.4.2")  
 *implementation*("androidx.camera:camera-view:1.4.2")  
  
 *// ML Kit para detección de objetos  
 implementation*("com.google.mlkit:object-detection:17.0.2")  
  
 *// Accompanist para gestión de permisos  
 implementation*("com.google.accompanist:accompanist-permissions:0.34.0")  
  
 *implementation*("androidx.lifecycle:lifecycle-viewmodel-compose:2.9.0")  
 *implementation*("androidx.lifecycle:lifecycle-runtime-ktx:2.9.0")  
 *implementation*("androidx.lifecycle:lifecycle-viewmodel-ktx:2.9.0")  
  
 *implementation*("androidx.lifecycle:lifecycle-runtime-compose:2.9.0")  
  
  
**}**

Código **Theme.kt**:

package com.example.capitulo5\_practica2.ui.theme  
  
import android.app.Activity  
import androidx.compose.foundation.isSystemInDarkTheme  
import androidx.compose.material3.MaterialTheme  
import androidx.compose.material3.darkColorScheme  
import androidx.compose.material3.dynamicDarkColorScheme  
import androidx.compose.material3.dynamicLightColorScheme  
import androidx.compose.material3.lightColorScheme  
import androidx.compose.runtime.Composable  
import androidx.compose.runtime.SideEffect  
import androidx.compose.ui.graphics.Color  
import androidx.compose.ui.graphics.toArgb  
import androidx.compose.ui.platform.*LocalContext*import androidx.compose.ui.platform.*LocalView*import androidx.core.view.WindowCompat  
  
*// Colores para el tema claro*private val *LightColorScheme* = *lightColorScheme*(  
 primary = *Color*(0xFF006E1C),  
 onPrimary = Color.White,  
 primaryContainer = *Color*(0xFF8BF990),  
 onPrimaryContainer = *Color*(0xFF002204),  
 secondary = *Color*(0xFF516350),  
 onSecondary = Color.White,  
 secondaryContainer = *Color*(0xFFD3E8D0),  
 onSecondaryContainer = *Color*(0xFF0F1F11),  
 tertiary = *Color*(0xFF39656B),  
 onTertiary = Color.White,  
 tertiaryContainer = *Color*(0xFFBDEAF1),  
 onTertiaryContainer = *Color*(0xFF001F23),  
 error = *Color*(0xFFBA1A1A),  
 background = *Color*(0xFFFCFDF6),  
 onBackground = *Color*(0xFF1A1C19),  
 surface = *Color*(0xFFFCFDF6),  
 onSurface = *Color*(0xFF1A1C19)  
)  
  
*// Colores para el tema oscuro*private val *DarkColorScheme* = *darkColorScheme*(  
 primary = *Color*(0xFF70DC76),  
 onPrimary = *Color*(0xFF00390A),  
 primaryContainer = *Color*(0xFF005313),  
 onPrimaryContainer = *Color*(0xFF8BF990),  
 secondary = *Color*(0xFFB7CCB5),  
 onSecondary = *Color*(0xFF233424),  
 secondaryContainer = *Color*(0xFF394B3A),  
 onSecondaryContainer = *Color*(0xFFD3E8D0),  
 tertiary = *Color*(0xFFA1CED5),  
 onTertiary = *Color*(0xFF00363C),  
 tertiaryContainer = *Color*(0xFF1F4D53),  
 onTertiaryContainer = *Color*(0xFFBDEAF1),  
 error = *Color*(0xFFFFB4AB),  
 background = *Color*(0xFF1A1C19),  
 onBackground = *Color*(0xFFE2E3DD),  
 surface = *Color*(0xFF1A1C19),  
 onSurface = *Color*(0xFFE2E3DD)  
)  
  
*/\*\*  
 \* Tema principal de la aplicación que configura Material3 con nuestros esquemas de colores.  
 \*/*@Composable  
fun Capitulo5\_practica2Theme(  
 darkTheme: Boolean = isSystemInDarkTheme(),  
 *// Habilitar el tema dinámico en Android 12+* dynamicColor: Boolean = true,  
 content: @Composable () -> Unit  
) {  
 val context = *LocalContext*.current  
  
 *// Determinar el esquema de colores basado en el modo oscuro/claro y si se soporta el color dinámico* val colorScheme = when {  
 dynamicColor && android.os.Build.VERSION.*SDK\_INT* >= android.os.Build.VERSION\_CODES.*S* -> {  
 if (darkTheme) *dynamicDarkColorScheme*(context) else *dynamicLightColorScheme*(context)  
 }  
 darkTheme -> *DarkColorScheme* else -> *LightColorScheme* }  
  
 *// Configurar la barra de estado* val view = *LocalView*.current  
 if (!view.*isInEditMode*) {  
 SideEffect **{** val window = (view.*context* as Activity).*window* val insetsController = WindowCompat.getInsetsController(window, view)  
  
 *// Configurar la apariencia de la barra de estado según el tema* insetsController.*isAppearanceLightStatusBars* = !darkTheme  
 **}** }  
  
 *// Aplicar el tema Material3 - usamos la tipografía definida en Type.kt* MaterialTheme(  
 colorScheme = colorScheme,  
 content = content  
 )  
}

Código **DetectedObject.kt**:

package com.example.capitulo5\_practica2

import android.graphics.Rect

/\*\*

\* Modelo de datos para representar un objeto detectado.

\* @param id Identificador único del objeto

\* @param label Etiqueta del objeto detectado (por ejemplo, "persona", "coche", etc.)

\* @param confidence Nivel de confianza de la detección (0.0 a 1.0)

\* @param boundingBox Rectángulo que delimita la posición del objeto en la imagen

\*/

data class DetectedObject(

val id: Int,

val label: String,

val confidence: Float,

val boundingBox: Rect

)

**Estructura Fundamental y Gestión de Estado**

class ObjectDetectionViewModel : ViewModel() {

*// Estado para almacenar la lista de objetos detectados*

private val \_detectedObjects = mutableStateOf<List<DetectedObject>>(emptyList())

val detectedObjects: State<List<DetectedObject>> = \_detectedObjects

*// Estado para indicar si la detección está en progreso*

private val \_isProcessing = mutableStateOf(false)

val isProcessing: State<Boolean> = \_isProcessing

*// ...*

}

**Configuración del Detector de ML Kit**

*// Configuración del detector de objetos*

private val options = ObjectDetectorOptions.Builder()

.setDetectorMode(ObjectDetectorOptions.SINGLE\_IMAGE\_MODE)

.enableMultipleObjects() *// Permite detectar múltiples objetos en una imagen*

.enableClassification() *// Habilita la clasificación de objetos*

.build()

*// Inicialización del detector de objetos*

private val objectDetector: ObjectDetector = ObjectDetection.getClient(options)

Esta

**Función Principal de Análisis**

fun analyzeImage(bitmap: Bitmap) {

if (\_isProcessing.value) return

\_isProcessing.value = true

\_detectedObjects.value = emptyList()

*// Crear una imagen de entrada para ML Kit a partir del bitmap*

val inputImage = InputImage.fromBitmap(bitmap, 0)

*// Lanzar la detección en un hilo en segundo plano*

CoroutineScope(Dispatchers.IO).launch {

*// ...procesamiento...*

}

}

**Procesamiento y Transformación de Resultados**

*// Convertir los resultados al formato de nuestro modelo de datos*

val detectedObjects = results.mapIndexed { index, detectedObject ->

*// Encontrar la etiqueta con mayor confianza*

val topLabel = if (detectedObject.labels.isNotEmpty()) {

detectedObject.labels.maxByOrNull { it.confidence }

} else null

val categoryIndex = topLabel?.index ?: 0

val confidence = topLabel?.confidence ?: 0f

*// Obtener etiqueta legible basada en la categoría*

val label = labelMap[categoryIndex] ?: topLabel?.text ?: "Objeto ${index + 1}"

*// Añadir el porcentaje de confianza a la etiqueta*

val labelWithConfidence = "$label (${(confidence \* 100).toInt()}%)"

DetectedObject(

id = index,

label = labelWithConfidence,

confidence = confidence,

boundingBox = detectedObject.boundingBox

)

}

**Ajuste de Coordenadas Visuales**

fun adjustBoundingBox(

detectedObject: DetectedObject,

previewWidth: Int,

previewHeight: Int,

imageWidth: Int,

imageHeight: Int

): RectF {

*// Calcular los factores de escala para ajustar el cuadro delimitador*

val scaleX = previewWidth.toFloat() / imageWidth.toFloat()

val scaleY = previewHeight.toFloat() / imageHeight.toFloat()

*// Obtener el cuadro delimitador original*

val boundingBox = detectedObject.boundingBox

*// Crear un nuevo cuadro delimitador ajustado a las dimensiones de la vista previa*

return RectF(

boundingBox.left \* scaleX,

boundingBox.top \* scaleY,

boundingBox.right \* scaleX,

boundingBox.bottom \* scaleY

)

}

**Gestión de Recursos y Ciclo de Vida**

override fun onCleared() {

super.onCleared()

*// Liberar recursos del detector cuando el ViewModel se destruya*

objectDetector.close()

}

**Extensión para Operaciones Asíncronas**

*// Extensión para convertir la tarea de ML Kit a una función suspendida para usar con coroutines*

suspend fun <T> com.google.android.gms.tasks.Task<T>.await(): T {

return suspendCoroutine { continuation ->

addOnSuccessListener { result ->

continuation.resume(result)

}

addOnFailureListener { exception ->

continuation.resumeWithException(exception)

}

}

}

Código **ObjectDetectionViewModel.kt**:

package com.example.capitulo5\_practica2  
  
import android.graphics.Bitmap  
import android.graphics.RectF  
import androidx.compose.runtime.State  
import androidx.compose.runtime.mutableStateOf  
import androidx.lifecycle.ViewModel  
import com.google.mlkit.vision.common.InputImage  
import com.google.mlkit.vision.objects.ObjectDetection  
import com.google.mlkit.vision.objects.ObjectDetector  
import com.google.mlkit.vision.objects.defaults.ObjectDetectorOptions  
import kotlinx.coroutines.CoroutineScope  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.launch  
import kotlinx.coroutines.withContext  
import kotlin.coroutines.resume  
import kotlin.coroutines.resumeWithException  
import kotlin.coroutines.suspendCoroutine  
  
*/\*\*  
 \* ViewModel que maneja la lógica de detección de objetos usando ML Kit.  
 \*/*class ObjectDetectionViewModel : ViewModel() {  
  
 *// Estado para almacenar la lista de objetos detectados* private val \_detectedObjects = *mutableStateOf*<List<DetectedObject>>(*emptyList*())  
 val detectedObjects: State<List<DetectedObject>> = \_detectedObjects  
  
 *// Estado para indicar si la detección está en progreso* private val \_isProcessing = *mutableStateOf*(false)  
 val isProcessing: State<Boolean> = \_isProcessing  
  
 *// Configuración del detector de objetos* private val options = ObjectDetectorOptions.Builder()  
 .setDetectorMode(ObjectDetectorOptions.*SINGLE\_IMAGE\_MODE*)  
 .enableMultipleObjects() *// Permite detectar múltiples objetos en una imagen* .enableClassification() *// Habilita la clasificación de objetos* .build()  
  
 *// Inicialización del detector de objetos* private val objectDetector: ObjectDetector = ObjectDetection.getClient(options)  
  
 *// Mapeo de categorías a etiquetas legibles* private val labelMap = *mapOf*(  
 0 *to* "Desconocido",  
 1 *to* "Hogar y Mobiliario",  
 2 *to* "Moda y Accesorios",  
 3 *to* "Comida y Bebida",  
 4 *to* "Plantas",  
 5 *to* "Lugar",  
 6 *to* "Persona",  
 7 *to* "Animal",  
 8 *to* "Vehículo",  
 9 *to* "Dispositivo",  
 10 *to* "Deporte",  
 11 *to* "Cocina",  
 12 *to* "Artículo"  
 )  
  
 */\*\*  
 \* Método para analizar una imagen y detectar objetos.  
 \* @param bitmap La imagen a analizar en formato Bitmap  
 \*/* fun analyzeImage(bitmap: Bitmap) {  
 if (\_isProcessing.value) return  
  
 \_isProcessing.value = true  
 \_detectedObjects.value = *emptyList*()  
  
 *// Crear una imagen de entrada para ML Kit a partir del bitmap* val inputImage = InputImage.fromBitmap(bitmap, 0)  
  
 *// Lanzar la detección en un hilo en segundo plano  
 CoroutineScope*(Dispatchers.IO).*launch* **{** try {  
 *// Procesar la imagen con el detector de objetos* val results = objectDetector.process(inputImage).await()  
  
 *// Convertir los resultados al formato de nuestro modelo de datos* val detectedObjects = results.*mapIndexed* **{** index, detectedObject **->** *// Encontrar la etiqueta con mayor confianza* val topLabel = if (detectedObject.*labels*.*isNotEmpty*()) {  
 detectedObject.*labels*.*maxByOrNull* **{ it**.*confidence* **}** } else null  
  
 val categoryIndex = topLabel?.*index* ?: 0  
 val confidence = topLabel?.*confidence* ?: 0f  
  
 *// Obtener etiqueta legible basada en la categoría* val label = labelMap[categoryIndex] ?: topLabel?.*text* ?: "Objeto ${index + 1}"  
  
 *// Añadir el porcentaje de confianza a la etiqueta* val labelWithConfidence = "$label (${(confidence \* 100).toInt()}%)"  
  
 DetectedObject(  
 id = index,  
 label = labelWithConfidence,  
 confidence = confidence,  
 boundingBox = detectedObject.*boundingBox* )  
 **}** *// Actualizar el estado con los objetos detectados* withContext(Dispatchers.Main) **{** \_detectedObjects.value = detectedObjects  
 \_isProcessing.value = false  
 **}** } catch (e: Exception) {  
 *// Manejar errores y actualizar el estado* withContext(Dispatchers.Main) **{** \_detectedObjects.value = *emptyList*()  
 \_isProcessing.value = false  
 **}** }  
 **}** }  
  
 */\*\*  
 \* Método para ajustar los cuadros delimitadores según las dimensiones de la vista.  
 \* @param detectedObject El objeto detectado con su cuadro delimitador original  
 \* @param previewWidth Ancho de la vista previa  
 \* @param previewHeight Alto de la vista previa  
 \* @param imageWidth Ancho de la imagen original  
 \* @param imageHeight Alto de la imagen original  
 \*/* fun adjustBoundingBox(  
 detectedObject: DetectedObject,  
 previewWidth: Int,  
 previewHeight: Int,  
 imageWidth: Int,  
 imageHeight: Int  
 ): RectF {  
 *// Calcular los factores de escala para ajustar el cuadro delimitador* val scaleX = previewWidth.toFloat() / imageWidth.toFloat()  
 val scaleY = previewHeight.toFloat() / imageHeight.toFloat()  
  
 *// Obtener el cuadro delimitador original* val boundingBox = detectedObject.boundingBox  
  
 *// Crear un nuevo cuadro delimitador ajustado a las dimensiones de la vista previa* return RectF(  
 boundingBox.left \* scaleX,  
 boundingBox.top \* scaleY,  
 boundingBox.right \* scaleX,  
 boundingBox.bottom \* scaleY  
 )  
 }  
  
 */\*\*  
 \* Método para limpiar los resultados de detección.  
 \*/* fun clearDetections() {  
 \_detectedObjects.value = *emptyList*()  
 }  
  
 override fun onCleared() {  
 super.onCleared()  
 *// Liberar recursos del detector cuando el ViewModel se destruya* objectDetector.close()  
 }  
}  
  
*// Extensión para convertir la tarea de ML Kit a una función suspendida para usar con coroutines*suspend fun <T> com.google.android.gms.tasks.Task<T>.await(): T {  
 return suspendCoroutine **{** continuation **->** addOnSuccessListener **{** result **->** continuation.*resume*(result)  
 **}** addOnFailureListener **{** exception **->** continuation.*resumeWithException*(exception)  
 **}  
 }**}

**Patrón de Composición Reactiva**

@OptIn(ExperimentalPermissionsApi::class)

@Composable

fun ObjectDetectionScreen(viewModel: ObjectDetectionViewModel) {

*// Captura de contextos del sistema*

val context = LocalContext.current

val lifecycleOwner = LocalLifecycleOwner.current

*// Estados locales del composable*

val cameraPermissionState = rememberPermissionState(android.Manifest.permission.CAMERA)

val previewView = remember { PreviewView(context) }

var capturedBitmap by remember { mutableStateOf<Bitmap?>(null) }

*// Estados derivados del ViewModel*

val detectedObjects = viewModel.detectedObjects.value

val isProcessing = viewModel.isProcessing.value

}

**Gestión de Permisos como Estado Reactivo**

val cameraPermissionState = rememberPermissionState(android.Manifest.permission.CAMERA)

LaunchedEffect(Unit) {

cameraPermissionState.launchPermissionRequest()

}

**Arquitectura Condicional de la Interfaz**

if (cameraPermissionState.status.isGranted) {

*// Interfaz principal de detección*

CameraView(*/\* parámetros \*/*)

*// Overlay condicional para resultados*

if (detectedObjects.isNotEmpty() && capturedBitmap != null && */\* validaciones \*/*) {

DetectionOverlay(*/\* parámetros \*/*)

}

*// Controles de interacción*

Button(*/\* configuración \*/*)

*// Estado de carga*

if (isProcessing) {

Column(*/\* indicador de progreso \*/*)

}

} else {

*// Interfaz de solicitud de permisos*

Column(*/\* UI de permisos \*/*)

}

**Puente Entre Compose y el Sistema de Cámara Nativo**

@Composable

fun CameraView(

context: Context,

lifecycleOwner: androidx.lifecycle.LifecycleOwner,

previewView: PreviewView,

onBitmapCaptured: (Bitmap) -> Unit,

onPreviewSizeChanged: (Int, Int) -> Unit

) {

var cameraInitialized by remember { mutableStateOf(false) }

LaunchedEffect(Unit) {

if (!cameraInitialized) {

*// Configuración compleja de CameraX*

}

}

AndroidView(

factory = { previewView.apply { */\* configuración \*/* } },

update = { view -> */\* gestión de cambios \*/* }

)

}

La configuración de CameraX dentro del LaunchedEffect coordina múltiples **use cases**:

*// Tres casos de uso especializados*

val preview = Preview.Builder().build() *// Visualización en tiempo real*

val imageAnalysis = ImageAnalysis.Builder().build() *// Análisis continuo (futuro)*

val imageCapture = ImageCapture.Builder().build() *// Captura bajo demanda*

*// Coordinación del ciclo de vida*

cameraProvider.bindToLifecycle(lifecycleOwner, cameraSelector, preview, imageAnalysis, imageCapture)

**Sistema de Visualización por Capas**

@Composable

fun DetectionOverlay(

detectedObjects: List<DetectedObject>,

viewModel: ObjectDetectionViewModel,

previewWidth: Int, previewHeight: Int,

imageWidth: Int, imageHeight: Int

) {

Canvas(modifier = Modifier.fillMaxSize().alpha(0.9f)) {

detectedObjects.forEach { detectedObject ->

val adjustedBox = viewModel.adjustBoundingBox(*/\* parámetros \*/*)

*// Dibujo de primitivas gráficas*

drawRect(*/\* contorno del objeto \*/*)

drawRect(*/\* fondo de etiqueta \*/*)

drawContext.canvas.nativeCanvas.drawText(*/\* texto descriptivo \*/*)

}

}

}

La función adjustBoundingBox del ViewModel realiza esta transformación:

*// Cálculo de factores de escala*

val scaleX = previewWidth.toFloat() / imageWidth.toFloat()

val scaleY = previewHeight.toFloat() / imageHeight.toFloat()

*// Aplicación de la transformación*

return RectF(

boundingBox.left \* scaleX,

boundingBox.top \* scaleY,

boundingBox.right \* scaleX,

boundingBox.bottom \* scaleY

)

**Pipeline de Procesamiento de Imágenes**

Button(

onClick = {

captureImage(

context = context,

previewView = previewView,

onBitmapCaptured = { bitmap ->

capturedBitmap = bitmap *// 1. Almacenamiento local*

viewModel.analyzeImage(bitmap) *// 2. Envío para análisis*

}

)

}

)

**Funciones de Utilidad y Extensiones**

*// Extensión para integración asíncrona*

private suspend fun Context.getCameraProvider(): ProcessCameraProvider =

suspendCoroutine { continuation ->

ProcessCameraProvider.getInstance(this).also { future ->

future.addListener(

{ continuation.resume(future.get()) },

ContextCompat.getMainExecutor(this)

)

}

}

*// Procesamiento de imágenes*

fun captureImage(context: Context, previewView: PreviewView, onBitmapCaptured: (Bitmap) -> Unit) {

val bitmap = previewView.bitmap

if (bitmap != null) {

val mutableBitmap = bitmap.copy(Bitmap.Config.ARGB\_8888, true)

val rotatedBitmap = rotateBitmap(mutableBitmap, 0f)

onBitmapCaptured(rotatedBitmap)

}

}

Código **ObjectDetectionScreen.kt**:

package com.example.capitulo5\_practica2  
  
import android.content.Context  
import android.graphics.Bitmap  
import android.graphics.Matrix  
import android.util.Log  
import android.view.ViewGroup  
import androidx.camera.core.CameraSelector  
import androidx.camera.core.ImageAnalysis  
import androidx.camera.core.ImageCapture  
import androidx.camera.core.ImageProxy  
import androidx.camera.core.Preview  
import androidx.camera.lifecycle.ProcessCameraProvider  
import androidx.camera.view.PreviewView  
import androidx.compose.foundation.Canvas  
import androidx.compose.foundation.background  
import androidx.compose.foundation.layout.Box  
import androidx.compose.foundation.layout.Column  
import androidx.compose.foundation.layout.fillMaxSize  
import androidx.compose.foundation.layout.fillMaxWidth  
import androidx.compose.foundation.layout.padding  
import androidx.compose.material3.Button  
import androidx.compose.material3.CircularProgressIndicator  
import androidx.compose.material3.Scaffold  
import androidx.compose.material3.Text  
import androidx.compose.runtime.Composable  
import androidx.compose.runtime.LaunchedEffect  
import androidx.compose.runtime.getValue  
import androidx.compose.runtime.mutableStateOf  
import androidx.compose.runtime.remember  
import androidx.compose.runtime.setValue  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.draw.alpha  
import androidx.compose.ui.geometry.Offset  
import androidx.compose.ui.graphics.Color  
import androidx.compose.ui.graphics.drawscope.Stroke  
import androidx.compose.ui.graphics.*nativeCanvas*import androidx.compose.ui.platform.*LocalContext*import androidx.compose.ui.platform.*LocalDensity*import androidx.compose.ui.text.TextStyle  
import androidx.compose.ui.text.font.FontWeight  
import androidx.compose.ui.text.style.TextAlign  
import androidx.compose.ui.unit.dp  
import androidx.compose.ui.unit.sp  
import androidx.compose.ui.viewinterop.AndroidView  
import androidx.core.content.ContextCompat  
import androidx.lifecycle.compose.*LocalLifecycleOwner*import com.google.accompanist.permissions.ExperimentalPermissionsApi  
import com.google.accompanist.permissions.*isGranted*import com.google.accompanist.permissions.rememberPermissionState  
import java.nio.ByteBuffer  
import kotlin.coroutines.resume  
import kotlin.coroutines.suspendCoroutine  
  
*/\*\*  
 \* Pantalla principal para la detección de objetos que integra CameraX y ML Kit.  
 \*/*@OptIn(ExperimentalPermissionsApi::class)  
@Composable  
fun ObjectDetectionScreen(viewModel: ObjectDetectionViewModel) {  
 *// El contexto actual y el propietario del ciclo de vida* val context = *LocalContext*.current  
 val lifecycleOwner = *LocalLifecycleOwner*.current  
  
 *// Estado para el permiso de cámara* val cameraPermissionState = rememberPermissionState(  
 android.Manifest.permission.*CAMERA* )  
  
 *// Estado para la vista previa de la cámara* val previewView = remember **{** PreviewView(context) **}** *// Estado para el bitmap capturado* var capturedBitmap by remember **{** *mutableStateOf*<Bitmap?>(null) **}** *// Estado para las dimensiones de la vista previa* var previewWidth by remember **{** *mutableStateOf*(0) **}** var previewHeight by remember **{** *mutableStateOf*(0) **}** *// Obtenemos los objetos detectados y el estado de procesamiento del ViewModel* val detectedObjects = viewModel.detectedObjects.value  
 val isProcessing = viewModel.isProcessing.value  
  
 *// Efecto para solicitar el permiso de cámara cuando se inicia la pantalla* LaunchedEffect(Unit) **{** cameraPermissionState.launchPermissionRequest()  
 **}** Scaffold **{** paddingValues **->** Box(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*padding*(paddingValues)  
 ) **{** if (cameraPermissionState.status.*isGranted*) {  
 *// Si se concede el permiso, mostramos la cámara* CameraView(  
 context = context,  
 lifecycleOwner = lifecycleOwner,  
 previewView = previewView,  
 onBitmapCaptured = **{** bitmap **->** capturedBitmap = bitmap  
 *// Analizamos la imagen capturada con ML Kit* viewModel.analyzeImage(bitmap)  
 **}**,  
 onPreviewSizeChanged = **{** width, height **->** previewWidth = width  
 previewHeight = height  
 **}** )  
  
 *// Dibujamos los cuadros delimitadores para los objetos detectados* if (detectedObjects.*isNotEmpty*() && capturedBitmap != null && previewWidth > 0 && previewHeight > 0) {  
 DetectionOverlay(  
 detectedObjects = detectedObjects,  
 viewModel = viewModel,  
 previewWidth = previewWidth,  
 previewHeight = previewHeight,  
 imageWidth = capturedBitmap?.*width* ?: 0,  
 imageHeight = capturedBitmap?.*height* ?: 0  
 )  
 }  
  
 *// Botón para capturar y analizar una imagen* Button(  
 onClick = **{** *captureImage*(  
 context = context,  
 previewView = previewView,  
 onBitmapCaptured = **{** bitmap **->** capturedBitmap = bitmap  
 viewModel.analyzeImage(bitmap)  
 **}** )  
 **}**,  
 modifier = Modifier  
 .*align*(Alignment.BottomCenter)  
 .*padding*(bottom = 16.*dp*)  
 ) **{** Text("Detectar Objetos")  
 **}** *// Mostrar indicador de carga durante el procesamiento* if (isProcessing) {  
 Column(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*align*(Alignment.Center)  
 .*background*(Color.Black.copy(alpha = 0.6f))  
 .*padding*(16.*dp*),  
 horizontalAlignment = Alignment.CenterHorizontally  
 ) **{** CircularProgressIndicator()  
 Text(  
 text = "Procesando imagen...",  
 color = Color.White,  
 modifier = Modifier.*padding*(top = 8.*dp*)  
 )  
 **}** }  
 } else {  
 *// Si no se concede el permiso, mostramos un mensaje* Column(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*padding*(16.*dp*),  
 horizontalAlignment = Alignment.CenterHorizontally  
 ) **{** Text(  
 text = "Se requiere permiso de cámara para esta aplicación",  
 style = TextStyle(  
 fontSize = 18.*sp*,  
 fontWeight = FontWeight.Bold,  
 textAlign = TextAlign.Center  
 ),  
 modifier = Modifier.*padding*(bottom = 16.*dp*)  
 )  
 Button(onClick = **{** cameraPermissionState.launchPermissionRequest() **}**) **{** Text("Solicitar Permiso")  
 **}  
 }** }  
 **}  
 }**}  
  
*/\*\*  
 \* Componente que muestra la vista previa de la cámara.  
 \*/*@Composable  
fun CameraView(  
 context: Context,  
 lifecycleOwner: androidx.lifecycle.LifecycleOwner,  
 previewView: PreviewView,  
 onBitmapCaptured: (Bitmap) -> Unit,  
 onPreviewSizeChanged: (Int, Int) -> Unit  
) {  
 *// Estado para controlar la inicialización de la cámara* var cameraInitialized by remember **{** *mutableStateOf*(false) **}** *// Configurar la cámara cuando el composable entre en la composición* LaunchedEffect(Unit) **{** if (!cameraInitialized) {  
 try {  
 val cameraProvider = context.getCameraProvider()  
  
 *// Configurar la vista previa* val preview = Preview.Builder().build().*also* **{  
 it**.setSurfaceProvider(previewView.*surfaceProvider*)  
 **}** *// Configurar el selector de cámara (usamos cámara trasera por defecto)* val cameraSelector = CameraSelector.*DEFAULT\_BACK\_CAMERA  
  
 // Configurar el analizador de imágenes* val imageAnalysis = ImageAnalysis.Builder()  
 .setBackpressureStrategy(ImageAnalysis.*STRATEGY\_KEEP\_ONLY\_LATEST*)  
 .build()  
  
 *// Configurar la captura de imágenes* val imageCapture = ImageCapture.Builder()  
 .setCaptureMode(ImageCapture.*CAPTURE\_MODE\_MINIMIZE\_LATENCY*)  
 .build()  
  
 *// Unbind todos los casos de uso antes de rebind* cameraProvider.unbindAll()  
  
 *// Bind los casos de uso a la cámara* cameraProvider.bindToLifecycle(  
 lifecycleOwner,  
 cameraSelector,  
 preview,  
 imageAnalysis,  
 imageCapture  
 )  
  
 cameraInitialized = true  
 } catch (e: Exception) {  
 Log.e("CameraView", "Error al vincular los casos de uso de la cámara", e)  
 }  
 }  
 **}** *// Mostrar la vista previa de la cámara* AndroidView(  
 factory = **{** previewView.*apply* **{** this.*layoutParams* = ViewGroup.LayoutParams(  
 ViewGroup.LayoutParams.*MATCH\_PARENT*,  
 ViewGroup.LayoutParams.*MATCH\_PARENT* )  
  
 *implementationMode* = PreviewView.ImplementationMode.*COMPATIBLE  
 scaleType* = PreviewView.ScaleType.*FILL\_CENTER* **}  
 }**,  
 modifier = Modifier.*fillMaxSize*(),  
 update = **{** view **->** *// Actualizar las dimensiones de la vista previa cuando estén disponibles* view.addOnLayoutChangeListener **{** \_, left, top, right, bottom, \_, \_, \_, \_ **->** val width = right - left  
 val height = bottom - top  
 if (width > 0 && height > 0) {  
 onPreviewSizeChanged(width, height)  
 }  
 **}  
 }** )  
}  
  
*/\*\*  
 \* Componente que muestra los objetos detectados sobre la vista previa de la cámara.  
 \*/*@Composable  
fun DetectionOverlay(  
 detectedObjects: List<DetectedObject>,  
 viewModel: ObjectDetectionViewModel,  
 previewWidth: Int,  
 previewHeight: Int,  
 imageWidth: Int,  
 imageHeight: Int  
) {  
 val density = *LocalDensity*.current.density  
  
 Canvas(  
 modifier = Modifier  
 .*fillMaxSize*()  
 .*alpha*(0.9f)  
 ) **{** detectedObjects.*forEach* **{** detectedObject **->** *// Ajustar el cuadro delimitador a las dimensiones de la vista previa* val adjustedBox = viewModel.adjustBoundingBox(  
 detectedObject,  
 previewWidth,  
 previewHeight,  
 imageWidth,  
 imageHeight  
 )  
  
 *// Dibujar el cuadro delimitador* drawRect(  
 color = Color.Green,  
 topLeft = *Offset*(adjustedBox.left, adjustedBox.top),  
 size = androidx.compose.ui.geometry.*Size*(  
 adjustedBox.width(),  
 adjustedBox.height()  
 ),  
 style = Stroke(width = 3f)  
 )  
  
 *// Dibujar la etiqueta y el nivel de confianza* drawRect(  
 color = Color.Black.copy(alpha = 0.7f),  
 topLeft = *Offset*(adjustedBox.left, adjustedBox.top - 30 / density),  
 size = androidx.compose.ui.geometry.*Size*(  
 adjustedBox.width(),  
 30 / density  
 )  
 )  
  
 drawContext.canvas.*nativeCanvas*.drawText(  
 "${detectedObject.label} (${(detectedObject.confidence \* 100).toInt()}%)",  
 adjustedBox.left + 5,  
 adjustedBox.top - 10 / density,  
 android.graphics.Paint().*apply* **{** *color* = android.graphics.Color.*WHITE  
 textSize* = 14 \* density  
 *isFakeBoldText* = true  
 **}** )  
 **}  
 }**}  
  
*/\*\*  
 \* Función para capturar una imagen desde la vista previa de la cámara.  
 \*/*fun captureImage(  
 context: Context,  
 previewView: PreviewView,  
 onBitmapCaptured: (Bitmap) -> Unit  
) {  
 *// Obtenemos el bitmap de la vista previa* val bitmap = previewView.*bitmap* if (bitmap != null) {  
 *// Crear una copia mutable del bitmap* val mutableBitmap = bitmap.copy(Bitmap.Config.*ARGB\_8888*, true)  
  
 *// Rotar el bitmap si es necesario según la orientación del dispositivo* val rotatedBitmap = *rotateBitmap*(mutableBitmap, 0f)  
  
 *// Notificar que se ha capturado un bitmap* onBitmapCaptured(rotatedBitmap)  
 } else {  
 Log.e("CaptureImage", "No se pudo capturar la imagen desde la vista previa")  
 }  
}  
  
*/\*\*  
 \* Función para rotar un bitmap.  
 \*/*private fun rotateBitmap(bitmap: Bitmap, rotation: Float): Bitmap {  
 if (rotation == 0f) return bitmap  
  
 val matrix = Matrix()  
 matrix.postRotate(rotation)  
  
 return Bitmap.createBitmap(  
 bitmap,  
 0,  
 0,  
 bitmap.*width*,  
 bitmap.*height*,  
 matrix,  
 true  
 )  
}  
  
*/\*\*  
 \* Extensión para obtener un proveedor de cámara de forma suspendida.  
 \*/*private suspend fun Context.getCameraProvider(): ProcessCameraProvider = suspendCoroutine **{** continuation **->** ProcessCameraProvider.getInstance(this).*also* **{** future **->** future.addListener(  
 **{** continuation.*resume*(future.get())  
 **}**,  
 ContextCompat.getMainExecutor(this)  
 )  
 **}  
}***/\*\*  
 \* Función para convertir un ImageProxy en un Bitmap.  
 \*/*fun imageProxyToBitmap(imageProxy: ImageProxy): Bitmap {  
 val buffer: ByteBuffer = imageProxy.*planes*[0].*buffer* val bytes = ByteArray(buffer.remaining())  
 buffer.get(bytes)  
  
 return android.graphics.BitmapFactory.decodeByteArray(bytes, 0, bytes.size)  
}

**Transición a la UI Declarativa**

setContent {

Capitulo5\_practica2Theme {

*// Configuración de la interfaz de usuario*

}

}

La función setContent marca el momento crucial donde **Android tradicional entrega el control a Jetpack Compose**. Esta función:

* Reemplaza el sistema tradicional de setContentView(R.layout.activity\_main).
* Establece el root del árbol de composición de Compose.
* Configura el contexto de composición donde se ejecutarán todos los composables.

Dentro del bloque de setContent, todo el código sigue el paradigma declarativo de Compose, donde describimos "qué queremos" en lugar de "cómo conseguirlo".

**Aplicación del Sistema de Temas**

Capitulo5\_practica2Theme {

*// Contenido de la aplicación*

}

Este composable envuelve toda la aplicación con el **sistema de temas personalizado**. Capitulo5\_practica2Theme es el composable que se definió anteriormente que:

* Aplica el esquema de colores para modo claro/oscuro.
* Configura la tipografía de Material 3.
* Establece las formas y elevaciones del tema.
* Habilita colores dinámicos en dispositivos Android 12+.
* Configura la apariencia de la barra de estado.

Este patrón de "theme wrapper" asegura que todos los componentes descendientes en el árbol de composición hereden automáticamente el styling correcto.

**Contenedor Principal: Surface**

Surface(

modifier = Modifier.fillMaxSize(),

color = MaterialTheme.colorScheme.background

) {

*// Contenido principal*

}

**Inicialización del ViewModel**

*// Inicializamos el ViewModel*

val viewModel: ObjectDetectionViewModel = viewModel()

**Composición de la Pantalla Principal**

*// Pantalla principal de detección de objetos*

ObjectDetectionScreen(viewModel = viewModel)