
**Departamento de Ciencias Básicas y Tecnologías**

**Carrera: Ingeniería en Sistemas**

**Fecha:15/11/2025**

**Integrante: Jostin Antonio Obregon López**

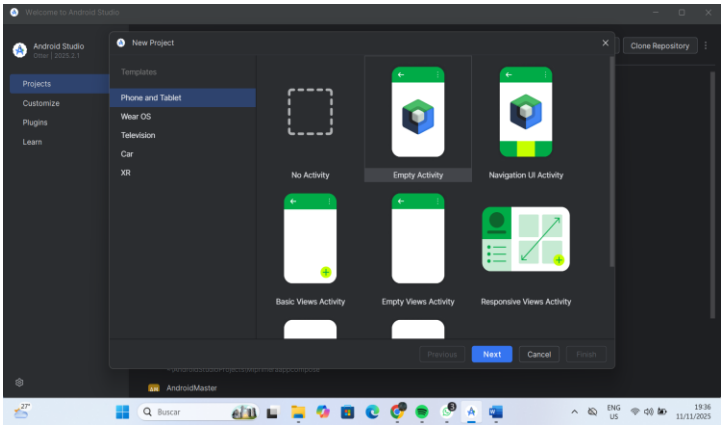
**Docente: ING. Danilo Rodríguez Guerrero**

**Materia: Desarrollo de Aplicaciones V**

**Managua Nicaragua**

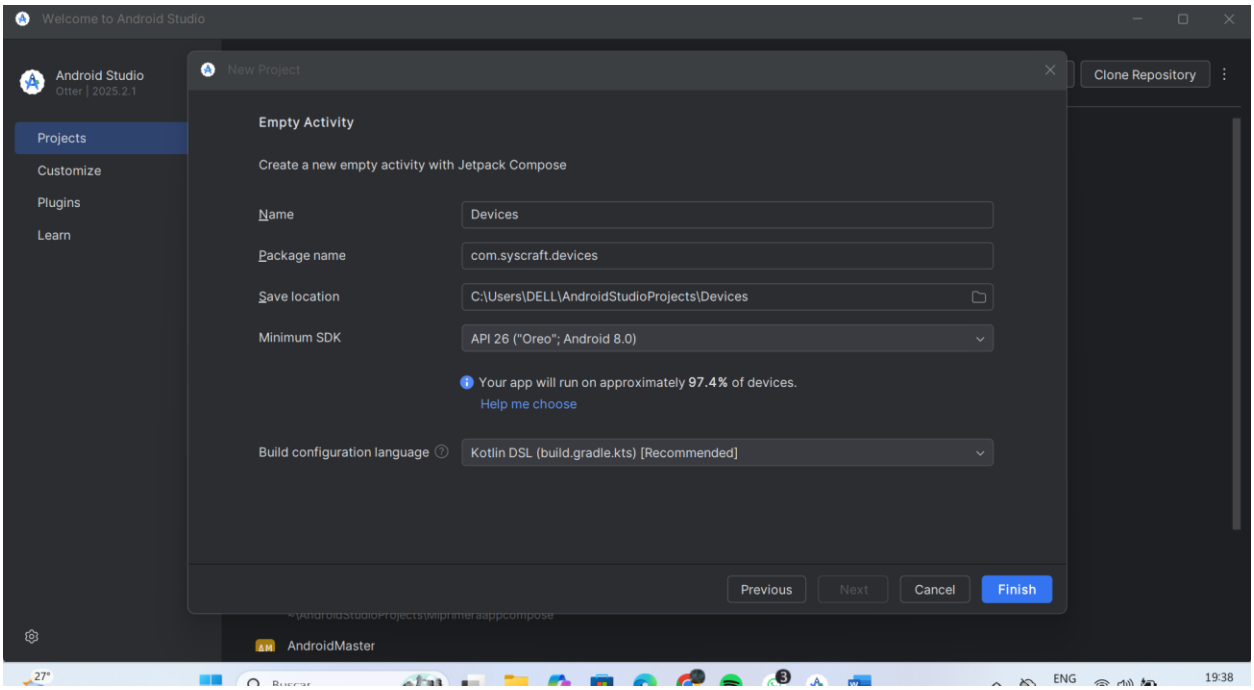
## Documentación del proyecto

### Paso 1 crear el Proyecto se elige empty activity



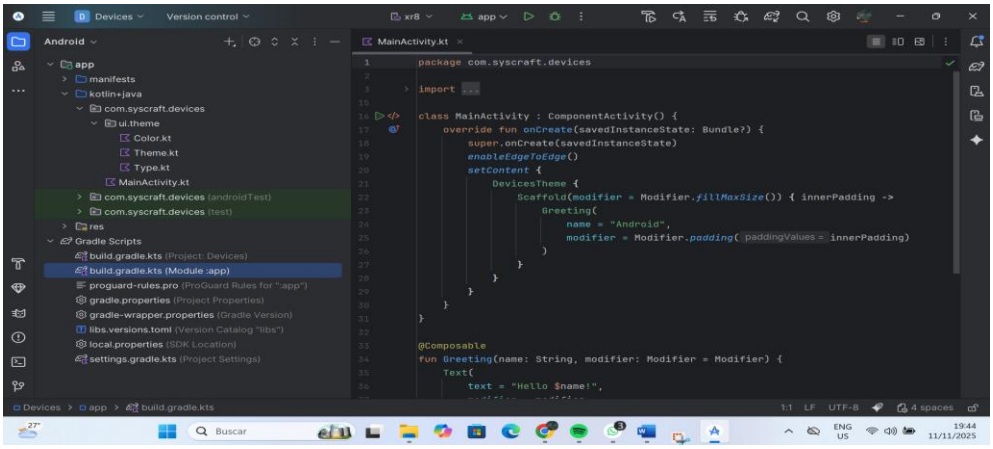

### Paso 2 especificaciones del proyecto

Estoy creando un nuevo proyecto en Android Studio llamado “Devices” con una actividad vacía usando Jetpack Compose. Le puse el paquete com.syscraft.devices y lo guardaré en mi carpeta de proyectos. Seleccioné la API 26 (Android 8.0 Oreo) como mínima y usaré Kotlin DSL para la configuración, que es la opción recomendada.

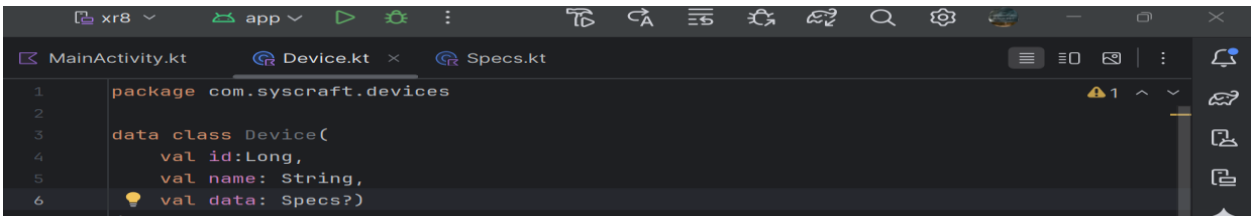


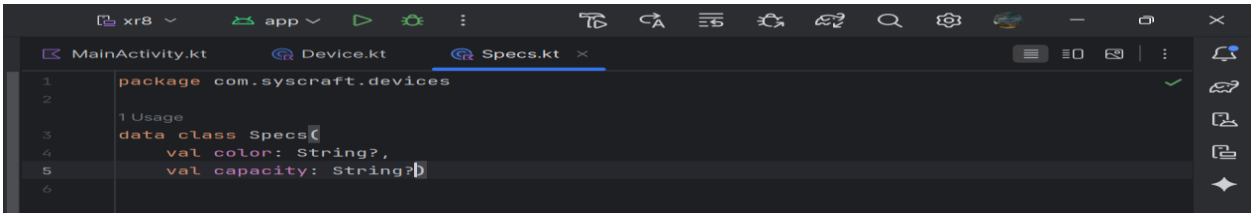
**Paso 3**

Ahora mismo estoy revisando la estructura de mi proyecto en Android Studio: en el directorio kotlin+java, tengo mi MainActivity.ktuso Jetpack Compose para definir la interfaz, como la función Greeting. Justo ahí, la carpeta ui.themees clave, pues en Color.kt, Type.kty Theme.ktdefine la paleta de colores, la tipografía y el estilo visual completo de mi aplicación. Finalmente, tengo seleccionado el archivo build.gradle.kts (Module: app), que es el encargado fundamental de gestionar y declarar todas las dependencias y configuraciones necesarias para que mi aplicación se compile y funcione correctamente y la carpeta res se enfoca en los recursos de Android




**Paso 4 creacion de clases**

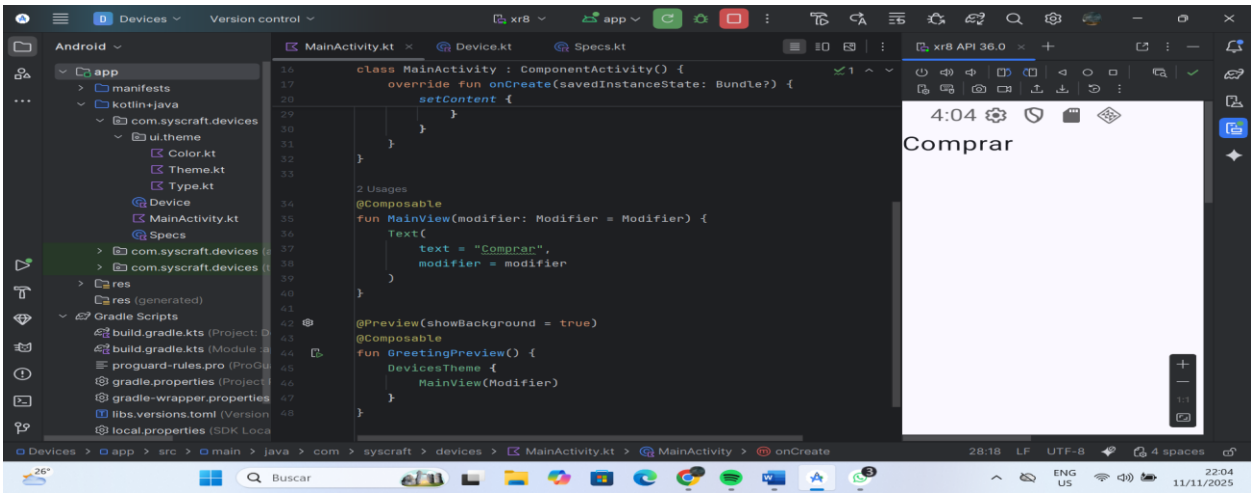



Acabo de crear dos archivos data classen Kotlin para estructurar los datos de mi aplicación de forma clara. En Device.kt, define la clase Devicecon propiedades clave como un id(Long) y un name(String). Lo más importante es que esta clase incluye la propiedad opcional data: Specs?, creando una relación. Luego, en Specs.kt, definí la clase Specspara almacenar los detalles del dispositivo, como el colory la capacity, ambos también como *String* opcionales. Esto me permite manejar datos complejos y anidados de una manera segura y eficiente.

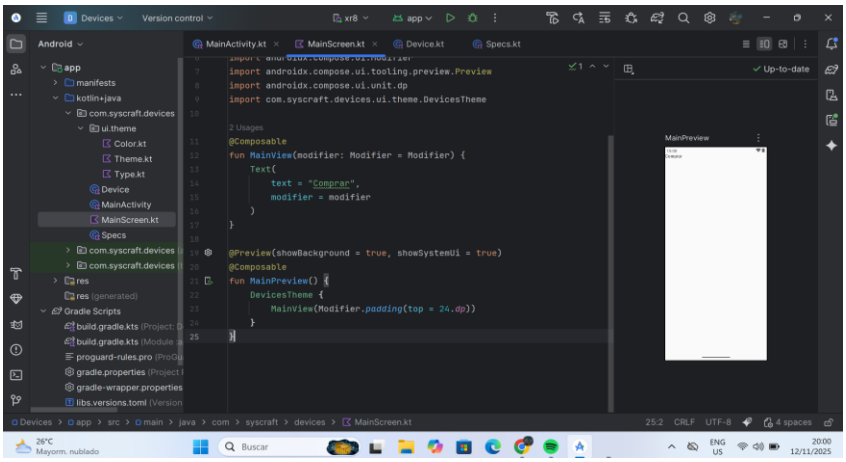
**Paso 5:**

En el directorio principal com.syscraft.devicesahora se ve claramente la organización con mis clases de datos Device.kty Specs.ktya definidas, junto con mi archivo de lógica principal MainActivity.kt. En el código, simplificó la función onCreate(líneas 17-20) para centrarme en llamar al contenido principal ( setContent). Justo debajo, mi función MainView (líneas 34-39) sigue siendo simple, mostrando el texto "Comprar" usando Jetpack Compose, tal como se ve en el emulador. Todo esto convive con mi configuración visual en ui.themey mis dependencias en los archivos build.gradle.kts.



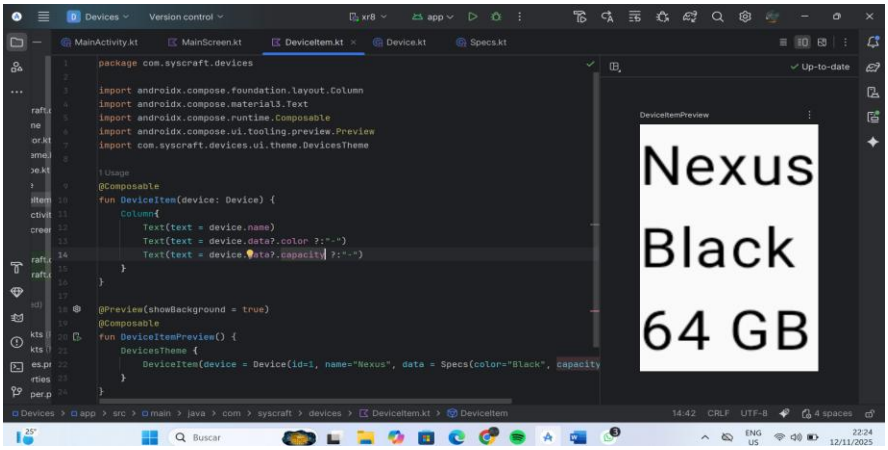

**Paso 6:**

Cree el archivo MainScreen.kt para separar y modular mi interfaz de usuario. En este archivo, define la función MainView, que es el componente principal que aún muestra el texto "Comprar". Además, añadí la vista previa MainPreview con la bandera showSystemUi = truey un *padding* superior de 24.d ppara probar la visualización de mi componente bajo la barra de estado.



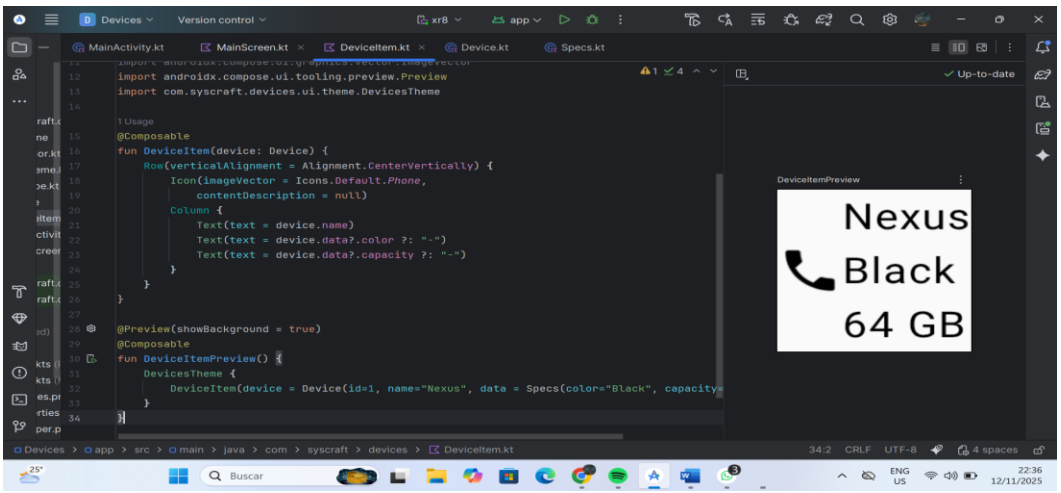
**Paso 7: trabajaremos en nuestro listado**

He mejorado el componente DeviceItemusando un Columnpara organizar la información verticalmente. Ahora, la función muestra el nombre, el color ( device.data?.color) y la capacidad ( device.data?.capacity). Para manejar los datos opcionales, utilicé el operador de llamada segura ( ?.) y el operador Elvis ( ?:) para que muestre un guion ("-") si el valor es nulo. En el DeviceItemPreview, se confirma que la información "Nexus", "Black" y "64 GB" se muestra correctamente.



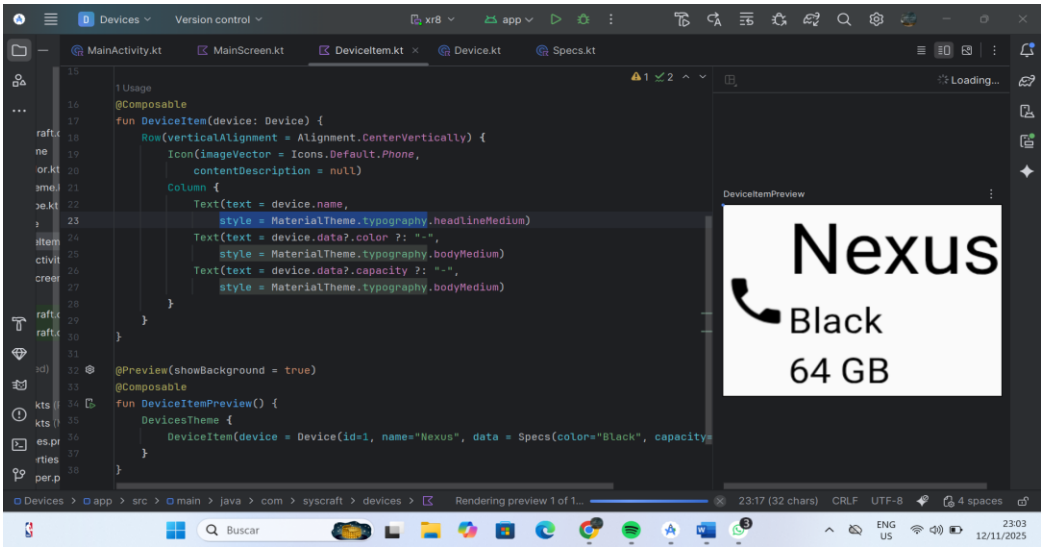

**Paso 8**

He finalizado la implementación de mi componente DeviceItemen DeviceItem.kt. La función ahora utiliza un Rowpara alinear horizontalmente un Icon de teléfono y un Column que organiza los detalles del dispositivo (nombre, color y capacidad) verticalmente. Esto me permite verificar, mediante el DeviceItemPreview, que la información del dispositivo "Nexus" se muestra correctamente junto a su ícono.



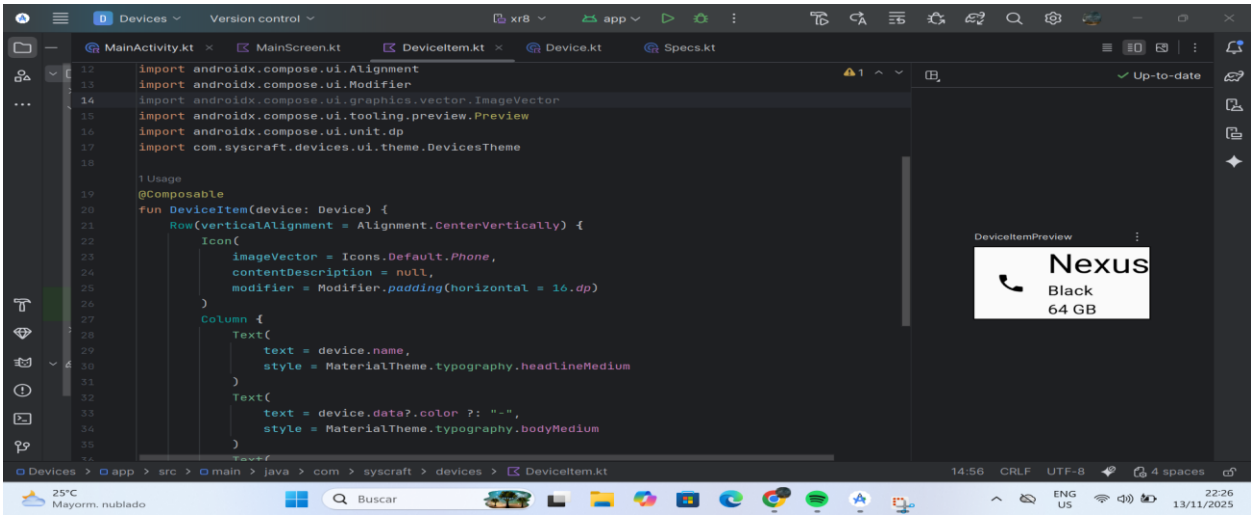
**Paso 9: estilo de texto**

El proyecto está bien estructurado con MainActivity.kty archivos de datos/componentes ( Device.kt, Specs.kt, etc.). La vista DeviceItemfinalizada utiliza un Rowy un Columnpara organizar el ícono y los detalles del dispositivo, aplicando estilos de MaterialTheme.typography clasificación visual. La vista previa confirma el correcto despliegue de la información del dispositivo "Nexus".



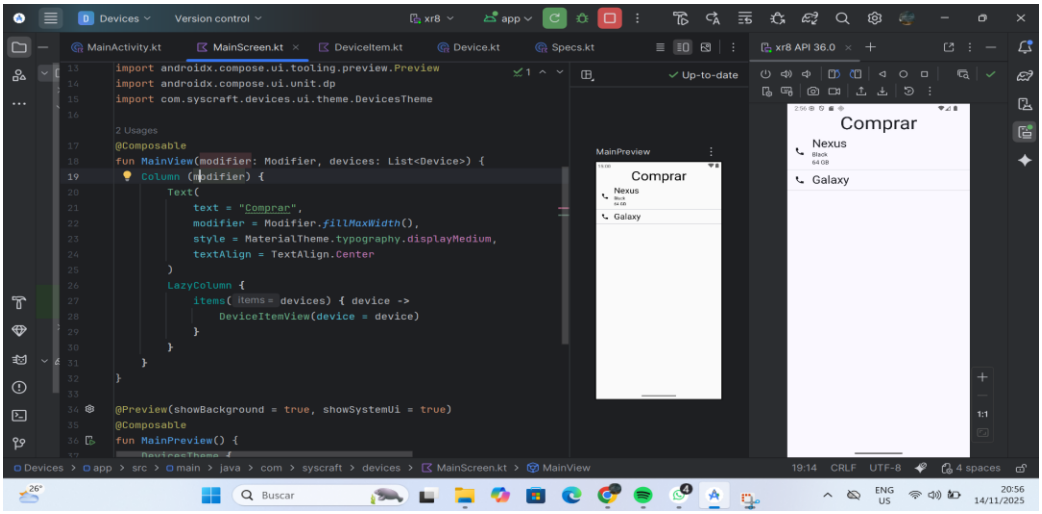

**Paso 10: modificadores de compose**

Acabo de modificar el componente Icon dentro de mi función DeviceItem para darle espacio y mejorar su presentación. Ahora, el Iconteléfono incluye una Modifier.padding(horizontal = 16. dp) (línea 25). Este modificador aplica un *padding* (espaciado) de 16.dpa los lados izquierdo y derecho del ícono. Esto separa el ícono del texto del dispositivo ("Nexus", "Black", etc.), haciendo que la lista sea más clara y estéticamente más agradable en el DeviceItemPreview.



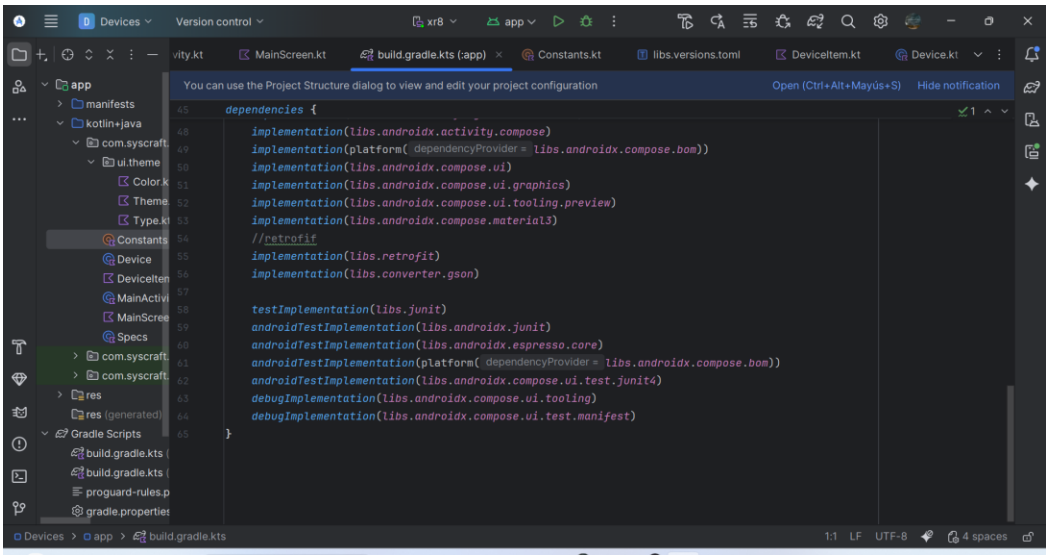
**Paso 11. Listado con columna**

He completado mi vista principal MainViewen MainScreen.kt. La función ahora acepta una lista de dispositivos ( List<Device>) y utiliza un Columnpara apilar el título "Comprar" y una LazyColumnpara iterar eficientemente sobre la lista. La LazyColumnllama al componente DeviceItemViewpor cada dispositivo, lo que permite visualizar correctamente el título y los elementos "Nexus" y "Galaxy" en el emulador.



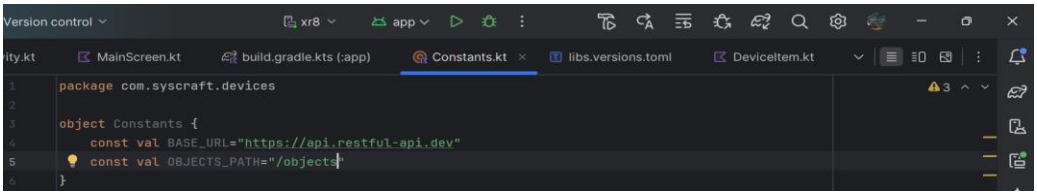

**Paso 12: Librería retrofif**

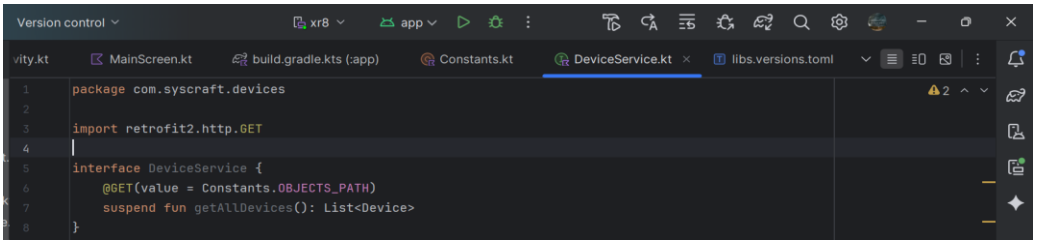
Se configuró Gradle para manejar solicitudes de red al agregar las dependencias de Retrofit y GSON ( libs.retrofity libs.converter.gson) en el archivo build.gradle.kts (app). Esto permite que mi aplicación se conecte a servicios web y convierta automáticamente las respuestas JSON en mis clases de datos de Kotlin.



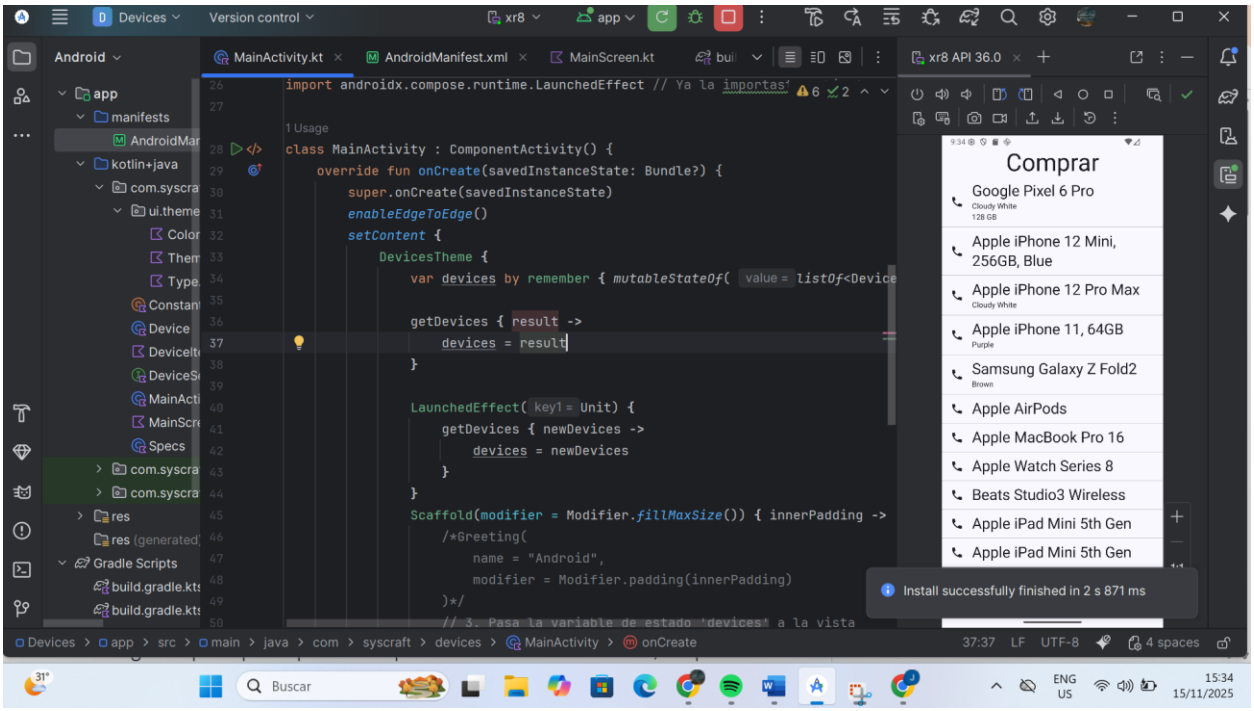
**Paso 13: Servicio de retrofif en Android**

He creado el archivo Constants.ktcon un Object llamado Constants. Su objetivo es centralizar los valores inmutables de la capa de red: definí BASE_URL( https://api.restful-api.dev) y OBJECTS_PATH( /objects). Esto asegura que mi código de Retrofit sea más limpio y fácil de mantener.




**14: EJECUTAR Y CONFIGURAR EL CONSUMO DE API EN ANDROID**




Conexión de Red y Carga de Datos

Acabo de completar la implementación para cargar la lista de dispositivos desde una API e hice los ajustes finales para que todo funcione. Los cambios más importantes fueron:

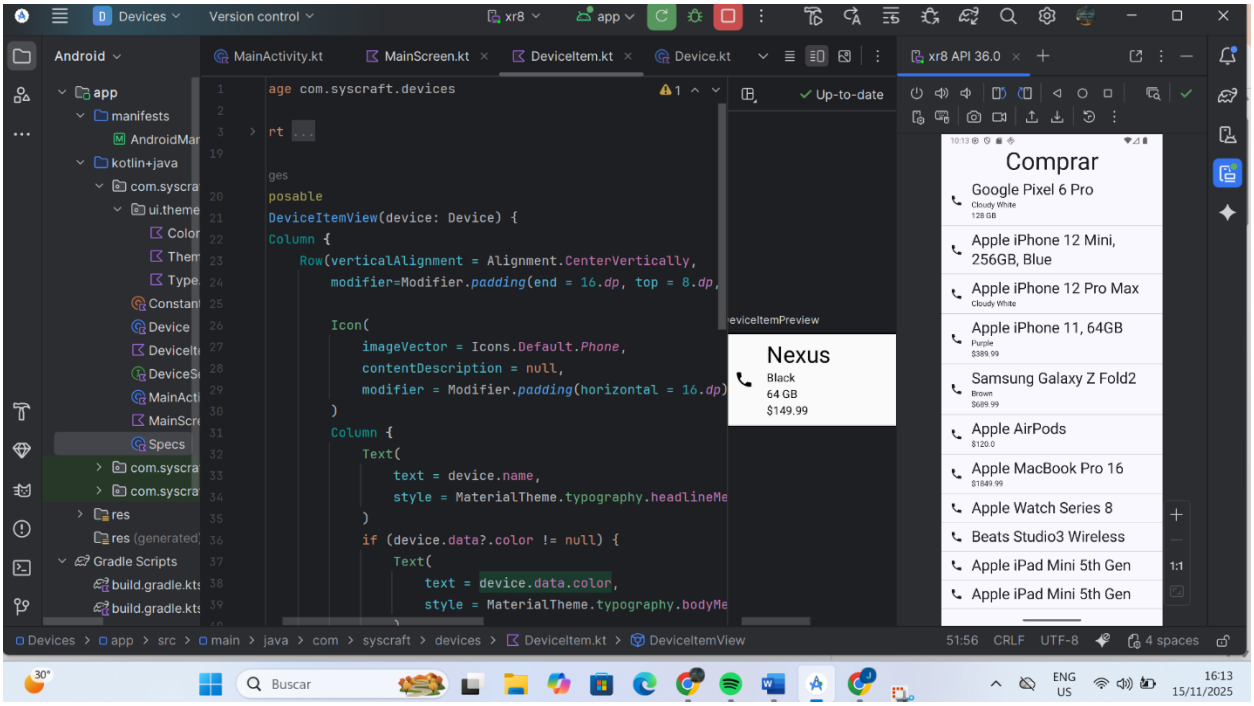
1. Activación de Internet: Añadí la línea <uses-permission android:name="android.permission.INTERNET"/>al archivo AndroidManifest.xmlpara darle permiso a la aplicación de acceder a la red.
2. Lógica de Carga (MainActivity.kt): Implementé el bloque LaunchedEffect(Unit) dentro de setContent. Este bloque asegura que la función getDevices(que configura Retrofit y hace la llamada con corrutinas) se ejecutará solo una vez al inicio.
3. Actualización de estado: Dentro de la función getDevices, al recibir los resultados (resulto newDevices), estos se usan para actualizar la variable de estado devices(devices = result), lo que refresca automáticamente la MainViewy muestra los dispositivos cargados en la interfaz.

Ahora, la aplicación carga datos dinámicamente al iniciar.

**Paso 15:  ejercicio 1 implementar valor al objeto**

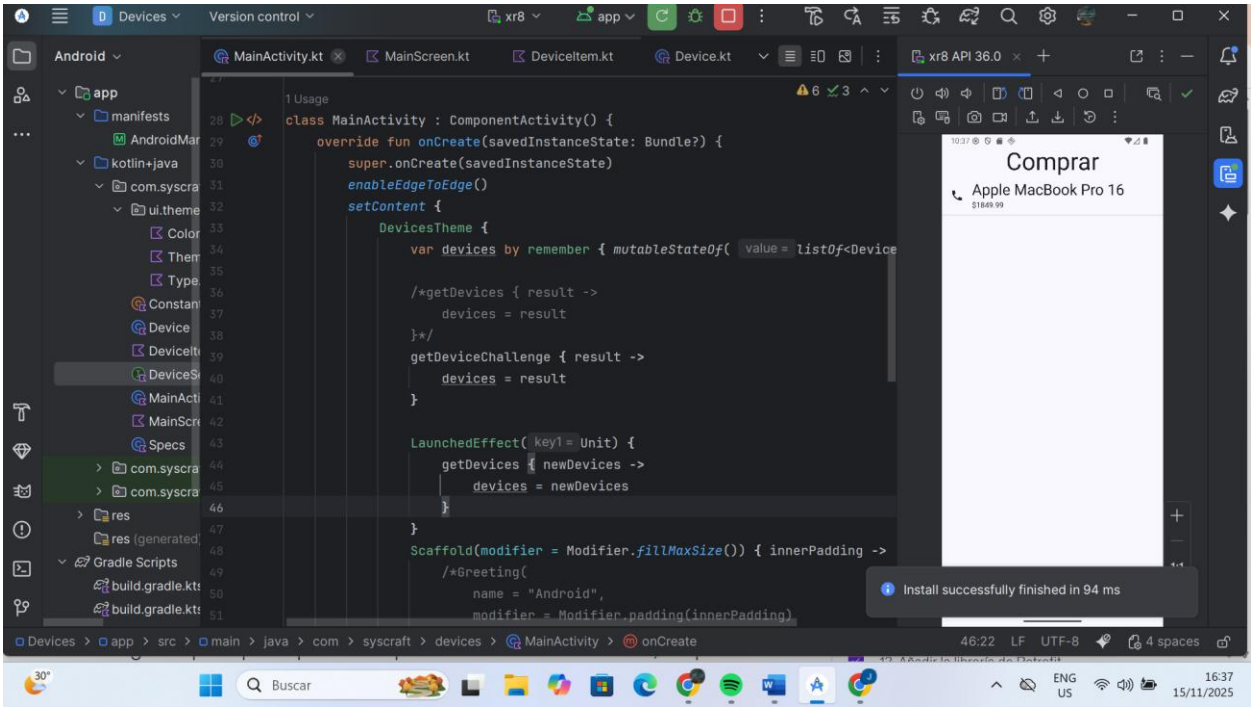
Modelo de Datos: Incluí el campo priceen la clase Specspara poder recibir el valor de la API (¿usando un tipo flexible como Any? para manejar inconsistencias en los datos).

Vista (DeviceItemView): Implementé un bloque condicional (if (device.data? price!= null)) para mostrar el precio y corregir un error de sintaxis en el estilo de texto, asegurando que use MaterialTheme.typography.bodyMedium para acceder a la tipografía del tema.



**Paso 16: ejercicio 2: único objeto**

Esta función utiliza la anotación @GETcon un *punto final* construido concatenando Constants.OBJECTS_PATH y Constants.OBJECT_PATH. Su objetivo es realizar una petición asíncrona (suspend fun) para obtener los detalles de un único dispositivo ( Device) de la API, apuntando a una URL específica (ej: /objects/7).




**Paso 17:**

Experiencia de Usuario: Se agregó un CircularProgressIndicatorcentralizado en la MainActivitypara mostrar el estado de carga mientras se obtienen los dispositivos de la red.