

Tema 2.1 - Usando un proyecto plantilla

Descargar estos apuntes [pdf](#) o [html](#)

Índice

- [Introducción](#)
- ▼ [Proyecto base](#)
 - [Características del proyecto base](#)
 - [Renombrando un proyecto base](#)
 - [Ampliando la plantilla del proyecto base](#)

Introducción

Android Studio es un IDE que se está actualizando constantemente desde Google empezó a adoptar Jetpack Compose como nueva forma de declarar Interfaces de Usuario. Es más, las propias librerías de Compose y Componentes de Material también se actualizan muy a menudo.

Cada versión de Android Studio tiene sus propias plantillas para crear proyectos nuevos. Estas plantillas se actualizan con cada nueva versión de Android Studio. Por lo tanto, esto supone que posiblemente los proyectos básico que se crean en cada versión utilicen versiones diferentes de las librerías de Compose, Material, etc.

Además, las plantillas que traen actualmente compose son muy básicas y no incluyen muchas de las librerías que se suelen utilizar en un proyecto real y el '*scaffolding*' o andamiaje básico de un proyecto a veces es un poco complejo y laborioso de crear.

Existen muchos repositorios en GitHub que contienen proyectos base para empezar a trabajar con Compose siguiendo las buenas prácticas de arquitectura de Android y diferentes interpretaciones de patrones.

De hecho, existe un repositorio oficial de Android en GitHub

<https://github.com/android/architecture-templates>. Donde se está trabajando en diferentes plantillas para crear proyectos con diferentes arquitecturas y patrones de diseño que aún está en versión beta y cuya idea es integrarlo en el futuro con Android Studio. Este tipo de proyectos incluyen scripts para renombrar de forma sencilla paquetes, nombres de clases, etc.

Proyecto base

Nosotros hemos incluido un proyecto base que incluye las librerías principales a usar en el curso a modo de plantilla como si fuera una foto de las dependencias y versiones a la fecha **11/09/2024**.

Pudes descargar el proyecto base desde el siguiente enlace:

- [Proyecto base 11/09/2024](#)

👉 **Importante:** Lo ideal es que partas de esta proyecto base, copiando la carpeta del '*workspace*' y la renombres con el nombre de tu aplicación. A continuación, puedes seguir los pasos que se indican en el siguiente apartado para renombrar el proyecto base.

Características del proyecto base

✦ **Nota:** Este apartado, se incluye a modo divulgativo para ser consultado más adelante en el curso. No es necesario que lo leas ahora pues mucho de lo que pone aún carece de sentido para nosotros

1. Usa **Kotlin** como lenguaje para el DSL de Gradle en lugar de Groovy.
2. `build.gradle.kts` del proyecto..
3. `build.gradle.kts` del módulo de la aplicación: Usa la versión `1.5.1` del compilador de compose `kotlinCompilerExtensionVersion`.

✦ **Nota:** Esta definición desaparece si usamos el compilador K2 de Kotlin y versiones 2.0 del mismo o superiores. A 11/9/2024 la plantilla de AStudio **Kotlin 1.9.0**

4. Incluye como dependencias las siguientes librerías que a lo largo del curso irán **tomando sentido** para nosotros, muchas de ellas llevan el prefijo `androidx` indicándonos que son extensiones oficiales de la librería de soporte de Android y puede que en un futuro se incluyan en el propio SDK de Android:

- **Librerías de Kotlin para Android:**

- `group = "androidx.core", name = "core-ktx"` : Esta librería es necesaria para usar las extensiones de Kotlin para Android.
- `group = "androidx.lifecycle", name = "lifecycle-runtime-ktx"` : Esta librería nos permite usar el ViewModelScope para lanzar corutinas desde un ViewModel.

- **Librerías de compose para Android:**

- **Bill Of Materials (BOM):** **BOM** nos permite usar las librerías de Compose y Material de forma coherente y que todas las librerías tengan la misma versión.

```
// En el libs.versions.toml
[versions]
composeBom = "2024.09.00"

[libraries]
androidx-compose-bom = {
    group = "androidx.compose",
    name = "compose-bom",
    version.ref = "composeBom"
}

// En el build.gradle.kts del módulo de la aplicación
implementation(platform(libs.androidx.compose.bom))
```

Algunas de estas **librerías** son:

```
androidx.compose.foundation:foundation
androidx.compose.ui:ui
androidx.compose.ui:ui-graphics
androidx.compose.ui:ui-tooling-preview
androidx.compose.material:material-icons-core
androidx.compose.material:material-icons-extended
```

Nosotros podremos especificar actualizaciones personalizadas de estas librerías en el archivo `libs.versions.toml`. Pero hay que tener cuidado por que el BOM ya incluye las versiones de las librerías de Compose y Material que se deben usar juntas por compatibilidad.

- o **Relacionadas con el ciclo de vida de las Activities** como:


```
androidx-lifecycle-runtime-ktx = {
    group = "androidx.lifecycle",
    name = "lifecycle-runtime-ktx",
    version.ref = "lifecycleRuntimeKtx"
}
```

Básicamente, esta librería, como ya veremos más adelante, nos permite lanzar corutinas desde un ViewModel y que se cancelen automáticamente cuando la Activity se destruye.

Renombrando un proyecto base

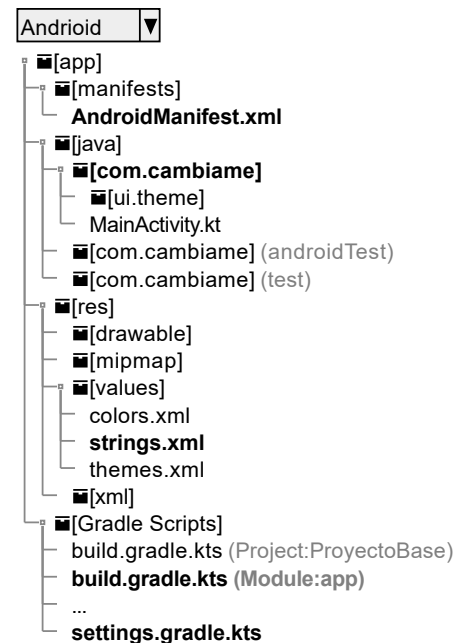
Una vez hemos descargado el proyecto base y lo hemos descomprimido.

Los pasos a realizar para renombrar correctamente el proyecto son los siguientes:

1. Copiamos el proyecto base en el lugar que deseemos.
2. Cambiamos el **nombre a la carpeta** o *'workspace'* del proyecto base, si lo consideramos necesario. Por ejemplo, `MiPrimerPrograma` .
3. Abrimos Android Studio y seleccionamos la carpeta que contiene nuestro proyecto base.

4. Cuando Gradle finalice todo seleccionaremos la vista **Android**. En esta más que carpeta lo que contendrá una serie de unidades organizativas jerárquicas de los elementos del proyecto similar al esquema de la derecha. renombraremos el paquete de la aplicación, para ello haremos **click derecho** sobre el paquete `com.cambiame` y seleccionaremos **Refactor -> Rename** o **Ctrl + R, R** si usamos el Keymap de Visual Studio.

- Seleccionaremos **All Directories**
- Cambiaremos `cambiame` por el nombre de nuestra app en minúsculas, sin espacios y sin caracteres especiales. Por ejemplo, `miprimerprograma` y pulsaremos **Refactor**. Si nos ofrece un preview de los cambios pulsaremos **Do Refactor**.



5. Aunque los paquetes se hayan renombrado, todavía quedaría por realizar algún paso más. Si abrimos el fichero `app → manifests → AndroidManifest.xml` veremos que se produce un error al no encontrar la actividad de entrada, además de que todavía aparece el nombre `ProyectoBase` en la etiqueta y en el `Theme`. Para resolver estos problemas tendremos que hacer lo siguiente:

```
<application ...>
    ...
    <activity
4        // Añadiremos la ruta con el nuevo nombre del paquete
5        // para encontrar la clase MainActivity
        android:name="com.holamundo.MainActivity"

        android:exported="true"
        android:label="@string/app_name"


11        // Renombraremos el ProyectoBase a HolaMundo con
12        // la refactorización del menú contextual o Ctrl + R, R
        android:theme="@style/Theme.HolaMundo">
        ...
    </activity>
```

6. Cambiaremos el nombre de la APP en `app → res → values → strings.xml`. Por ejemplo, por `HolaMundo`.

```
<resources>
2    <string name="app_name">HolaMundo</string>
</resources>
```

7. Abriremos al archivo de configuración de Gradle `app → Gradle Scripts → settings.gradle.kts` y cambiaremos el nombre del proyecto `ProyectoBase` por el nuevo nombre que le queramos dar. Por ejemplo, `HolaMundo`.

```
...
2    rootProject.name = "HolaMundo"
    include(":app")
```

 **Nota:** Este será el nombre del proyecto que aparecerá al abrir el proyecto con Android Studio independientemente del nombre que le hayamos puesto a la carpeta del *'Workspace'*.

8. Abriremos el archivo gradle del módulo de la aplicación que se encuentra en

app → Gradle Scripts → build.gradle.kts (Module:app)

Cambiaremos el nombre del **paquete** de nuestro programa principal **com.cambiame** por el que hayamos puesto en el paso 4. En nuestro ejemplo **com.holamundo**. Fíjate en el siguiente script de ejemplo.

```
plugins {  
    alias(libs.plugins.android.application)  
    alias(libs.plugins.kotlin.android)  
}  
  
android {  
    7 namespace = "com.pmdm.proyectobase2425"  
    compileSdk = 34  
  
    defaultConfig {  
    11 applicationId = "com.pmdm.proyectobase2425"  
        minSdk = 28  
        targetSdk = 34  
        versionCode = 1  
        versionName = "1.0"  
  
        ...  
    }  
}
```

9. Además del nombre del **tema de la MainActivity** que hemos cambiado en el '*manifest*', deberemos cambiar el nombre del **tema de compose**. Para ello iremos

app → java → com.holamundo.ui.theme → Theme.kt y cambiaremos el nombre del tema **ProyectoBaseTheme** por **HolaMundoTheme** para seguir la nomenclatura de nuestro ejemplo.

🔴 **Nota:** Recuerda que es importante hacer una refactorización del nombre del tema con **Ctrl + R, R** para que se cambie en todos los lugares donde se usa.

```
@Composable  
2 fun HolaMundoTheme(  
    darkTheme: Boolean = isSystemInDarkTheme(),  
    dynamicColor: Boolean = true,  
    content: @Composable() () -> Unit  
) {  
    ...  
}
```

10. Por último, sincronizaremos el proyecto con los cambios en los scripts de Gradle con la opción de menú **File → Sync Project with Gradle Files** o **Ctrl + Mayús + 0**. También podemos

pulsar el botón de sincronización de Gradle que aparece en la barra de herramientas arriba a la derecha con el icono del elefante de Gradle.

Volveremos a abrir el archivo del `AndroidManifest.xml` y veremos que el error ha desaparecido al volver a sincronizar y ya podremos ejecutar la aplicación para ver si todo ha ido bien

Ampliando la plantilla del proyecto base

Durante el curso iremos aplicando la arquitectura de una aplicación de Android propuesta por Google y que veremos en el siguiente tema. Para implementarla iremos añadiendo una organización de paquetes, clases, nombres y diferentes librerías según vayamos avanzando en el curso.

Por tanto, a medida que vayamos avanzando en el curso podemos ir actualizando la plantilla o esqueleto de proyecto base, con las nuevas librerías y organización de paquetes y clases que vayamos viendo y así tener un '*Template*' personalizado para iniciar un proyecto de forma rápida.