Presentación de una colección

Descargar estos apuntes

Ejercicio

Vamos a definir la **pantalla de presentación** (*splash* o pantalla de bienvenida) de una futura aplicación que gestionará **una colección sobre un tema que te resulte interesante** (cómics, videojuegos, discos de vinilo, plantas, coches clásicos, cartas, recetas, series, etc.).

La pantalla de la imagen se propone como **ejemplo de partida**: puedes tomarla como idea e inspiración para generar tu propia presentación, adaptando el diseño, los textos, los colores y los componentes de Material Design 3 (M3) al tema que hayas elegido.

La presentación mostrará, como mínimo:

- Una **imagen** de un elemento de tu colección que cambiará de forma **aleatoria cada vez que se inicie la aplicación**.
- El **nombre** de la aplicación y un **eslogan** o breve descripción.
- Los **botones** de acceso a la aplicación (de momento no harán nada).
- Debes crear un proyecto nuevo a partir del **proyecto base** eligiendo un nombre para tu aplicación, renombrando el proyecto base y **añadiendo un icono**, como ya hemos hecho en otras ocasiones.

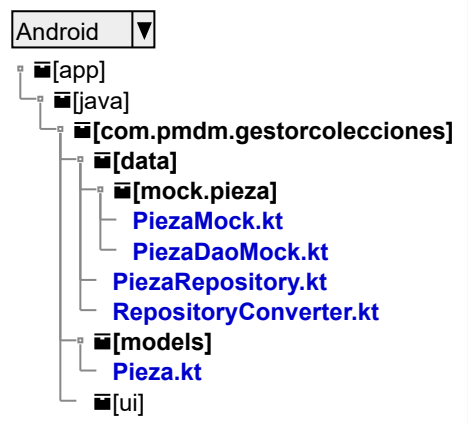


Paso 1: Definiendo los datos *mock* en el paquete `data`

A partir del proyecto base renombrado a `GestorColecciones` y Siguiendo la arquitectura propuesta en el curso, vamos a comenzar a construir una App que se irá completando a lo largo de los temas. Para ello en el paquete `data` de tu aplicación crearás los datos de prueba (*mock*) de tu colección. Los datos *mock* son datos ficticios que se definen directamente en el código para poder desarrollar la UI sin necesidad de una base de datos ni de un servidor de datos.

Llamaremos `Pieza` a cada elemento de la colección. La estructura de paquetes será la siguiente:

- En `com.pmdm.gestorcolecciones` crearemos el paquete `models` con el fichero `Pieza.kt`, que es el modelo que usará la capa de UI.
- Dentro de `data` crearemos el paquete `mock` y dentro de este el paquete `pieza` con los ficheros `PiezaMock.kt` (la entidad mock, que sustituye a lo que sería la entidad de la base de datos) y `PiezaDaoMock.kt` (el DAO mock que contiene los datos de prueba).
- Fuera de `mock`, en el paquete `data`, crearemos `PiezaRepository.kt` y `RepositoryConverter.kt`, como ya hemos hecho en otros ejercicios.



Define en `models/Pieza.kt` una `data class` que represente un elemento de tu colección, adaptando los campos al tema elegido. Por ejemplo, para una colección de videojuegos:

```
data class Pieza(
    val id: Int,
    val nombre: String,
    val descripcion: String,
    val imagenUrl: String   // URL de la imagen alojada en el servidor nginx.
)
```

En `data/mock/pieza/PiezaMock.kt` define la entidad mock con los mismos campos:

```
data class PiezaMock(
    val id: Int,
    val nombre: String,
    val descripcion: String,
    val imagenUrl: String
)
```

En `PiezaDaoMock.kt` crea la lista con **al menos 5 piezas** de tu colección y **cada una de ellas con su imagen**:

```
class PiezaDaoMock {
    private val piezas: List<PiezaMock> = listOf(
        PiezaMock(
            id = 1,
            nombre = "...",
            descripcion = "...",
            imagenUrl = "http://<IP_DEL_SERVIDOR>/imagenes/pieza1.jpg"
        ),
        // Al menos 5 piezas
    )

    fun getAll(): List<PiezaMock> = piezas
}
```

En `RepositoryConverter.kt` define la conversión de la entidad mock al modelo, y en `PiezaRepository.kt` el repositorio que usará la UI:

```
fun PiezaMock.toPieza() = Pieza(id, nombre, descripcion, imagenUrl)
```

```
class PiezaRepository(private val piezaDao: PiezaDaoMock = PiezaDaoMock()) {
    fun getAll(): List<Pieza> = piezaDao.getAll().map { it.toPieza() }
}
```

Importante: las imágenes se alojan en el servidor nginx

Las imágenes **no** deben incluirse en los recursos (`res/drawable`) del proyecto. Cada `imagenUrl` debe apuntar a una imagen **alojada en el servidor nginx** del curso (el mismo que se utiliza en el resto de ejercicios). Debes, por tanto:

- Subir las imágenes de tu colección al directorio publicado por nginx.
- Comprobar desde el navegador que cada URL es accesible antes de usarla en la aplicación.
- Declarar el permiso de Internet en el `AndroidManifest.xml` :

```
<uses-permission android:name="android.permission.INTERNET" />
```

- Si el servidor usa `http` en lugar de `https` , recuerda permitir el tráfico en texto claro (`android:usesCleartextTraffic="true"` en la etiqueta `<application>`) o configurar un *network security config*.

> **Warning**

> Recuerda, la dirección de acceso al servidor local desde el emulador de android debe ser `10.0.2.2`, por ejemplo para Nginx activado con una carpeta coleccion en la parte pública de este, donde tuviésemos nuestras imágenes, la url base sería: **`http://10.0.2.2/coleccion/`**

Paso 2: Cargando la imagen con `AsyncImage`

Para mostrar las imágenes cuya fuente es una URL **debes utilizar el componente** `AsyncImage` de la librería **Coil**, que se encarga de descargar y mostrar la imagen de forma asíncrona.

Añade la dependencia en el fichero `libs.versions.toml` y en el `build.gradle.kts` del módulo `app`:

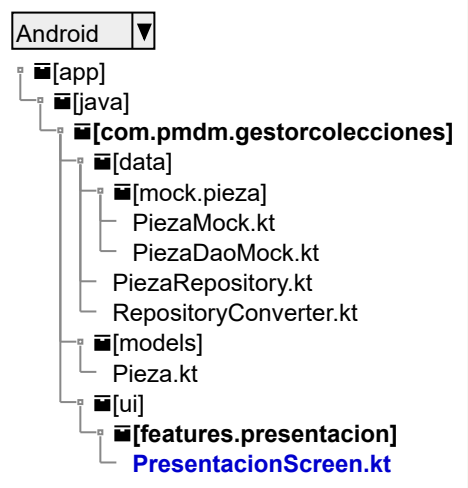
```
implementation("io.coil-kt.coil3:coil-compose:3.0.4")
implementation("io.coil-kt.coil3:coil-network-okhttp:3.0.4")
```

Ejemplo de uso:

```
AsyncImage(
    model = pieza.imagenUrl,
    contentDescription = pieza.nombre,
    contentScale = ContentScale.Crop,
    modifier = Modifier.fillMaxWidth()
)
```

Paso 3: Definiendo la pantalla de presentación

En el paquete **`ui`** crea el paquete **`features.presentacion`** y dentro el fichero **`PresentacionScreen.kt`** con la función composable que define la pantalla.



La función recibirá el elemento a mostrar y la acción del botón, de modo que sea fácil de probar con un `@Preview`:

```
@Composable
fun PresentacionScreen(
    pieza: Pieza,
    onEntrarClick: () -> Unit
)
```

Requisitos:

- La imagen mostrada debe elegirse de forma **aleatoria** entre las piezas obtenidas de `PiezaRepository` **cada vez que se inicia la aplicación** (por ejemplo con `PiezaRepository().getAll().random()`).
- Utiliza componentes de **M3** (`Surface`, `Text`, `Button`, etc.) y los estilos de `MaterialTheme.typography` y `MaterialTheme.colorScheme`.
- El resto del diseño (disposición, colores, tipografías, fondos) queda a tu criterio, de forma que se adapten al tema que hayas elegido.

Paso 4: Definiendo el `@Preview` y la `MainActivity`

Define un `@Preview` que muestre la pantalla con una pieza de tu colección:

```
@Preview(showBackground = true)
@Composable
fun PresentacionPreview() {
    MiColeccionTheme {
        PresentacionScreen(
            pieza = PiezaRepository().getAll().first(),
            onEntrarClick = {}
        )
    }
}
```

Por último, en la `MainActivity` llama a `PresentacionScreen` pasándole una pieza aleatoria de la lista del repositorio, y ejecuta la aplicación varias veces para comprobar que la imagen cambia en cada inicio.