**Sesión 2-07 Clase del 14 de noviembre**

En esta sesión comenzamos a trabajar con un proyecto **AppMusica** en el que iremos haciendo mejoras en esta y otras sesiones.

En el proyecto se va a trabajar con imágenes vectoriales e imágenes de trama o de mapa de bits. Es recomendable que leas en este enlace de **Android Developers:**

<https://developer.android.com/develop/ui/compose/graphics?hl=es-419>

Verás que la función **BottomNavigationButton** está sobrecargada. En una primera versión, recibe en el parámetro **icon** un **ImageVector** (imagen vectorial-XML), En la segunda versión recibe un objeto **Painter** (clase para manejar cualquier tipo de imagen no vectorial).

1.- Como el icono que representa a los artistas no nos parece muy adecuado vamos a cambiarlo por otro también vectorial. Podemos buscarlo en webs de descarga gratuita de iconos, pero en este caso vamos a generarlo desde las fuentes de assets de Android Studio.

En el siguiente enlace de Android Studio Developer:

<https://developer.android.com/studio/write/vector-asset-studio?hl=es-419#running>

Tienes información sobre como agregar iconos de **Material** a los recursos del proyecto. Agrega a la carpeta **res/drawable** un archivo de imagen vectorial para un icono de tamaño 24x24. Por ejemplo, busca el de nombre **queue\_music** (puedes elegir otro a tu gusto).

Ahora vamos a asignar ese icono al botón de navegación **Artists.** Tenemos que sustituir:

icon = Icons.Filled.*List*

por

icon = ImageVector.vectorResource(R.drawable.*archivo\_res*)

Repite el proceso para cambiar el icono del botón **premium.**

2.- Queremos usar para el botón **premium** más adecuado y nos encontramos en el supuesto de que no encontramos un XML adecuado. Podríamos diseñarlo en Android Studio, pero no es objetivo del curso.

Vamos a buscar uno, aunque no sea vectorial. Buscamos en una web de descarga gratuita de iconos e imágenes, por ejemplo, **FlatIcon**.

Busca uno apropiado, lo descargas como png en tamaño 24x24 y lo agregas a **res/drawable** del proyecto.

Icono

Descripción generada automáticamente

Una vez agregado, se tendrá que pasar un **Painter** que cargue ese iconoa la función **BottonNavigationButton.** Se estará en este caso llamando a la función sobrecargada, la que recibe un **Painter**. La forma de pasar el **painter** de ese archivo de recurso será:

icon = painterResource(id = R.drawable.*archivo\_res*),

3.- **Configuración de Home**.

3.1.- Diseñamos el contenido para **Home.** Creamos un archivo **ContentHomeView** en el package **views.** En ese archivo añadimos una función con el mismo nombre que usa un **Box** que ocupa todo el espacio disponible e incluye un **Text** que pinta en letra grande el texto **“HOME”** centrado en la pantalla.

3.2.- Queremos que en Home se escriba una **Reflexión del día.**

Para ello vamos a tener una lista de objetos Reflexión(frase, autor fecha). Se escribirá en un recuadro el texto y autor de la frase correspondiente la reflexión que tenga la fecha actual.

Crea una **data class Reflexión** dentro del package **model.**

Crea la lista **reflexiones** dentro de un archivo **Reflexiones.kt** en un **package data.** Este es el código fuente de una lista con 10 reflexiones de filósofos.

val *reflexiones* = *listOf*(  
 Reflexion(  
 frase = "La vida debe ser comprendida hacia atrás. Pero debe ser vivida hacia adelante.",  
 autor = "Søren Kierkegaard",  
 fecha = LocalDate.of(2024, 11, 10)  
 ),  
 Reflexion(  
 frase = "La felicidad no es un ideal de la razón, sino de la imaginación.",  
 autor = "Immanuel Kant",  
 fecha = LocalDate.of(2024, 11, 11)  
 ),  
 Reflexion(  
 frase = "El que tiene un porqué para vivir puede soportar casi cualquier cómo.",  
 autor = "Friedrich Nietzsche",  
 fecha = LocalDate.of(2024, 11, 12)  
 ),  
 Reflexion(  
 frase = "No se puede pisar dos veces el mismo río.",  
 autor = "Heráclito",  
 fecha = LocalDate.of(2024, 11, 13)  
 ),  
 Reflexion(  
 frase = "Solo sé que no sé nada.",  
 autor = "Sócrates",  
 fecha = LocalDate.of(2024, 11, 14)  
 ),  
 Reflexion(  
 frase = "La verdadera sabiduría está en reconocer la propia ignorancia.",  
 autor = "Platón",  
 fecha = LocalDate.of(2024, 11, 15)  
 ),  
 Reflexion(  
 frase = "Pienso, luego existo.",  
 autor = "René Descartes",  
 fecha = LocalDate.of(2024, 11, 16)  
 ),  
 Reflexion(  
 frase = "El hombre está condenado a ser libre.",  
 autor = "Jean-Paul Sartre",  
 fecha = LocalDate.of(2024, 11, 17)  
 ),  
 Reflexion(  
 frase = "No hay hechos, solo interpretaciones.",  
 autor = "Friedrich Nietzsche",  
 fecha = LocalDate.of(2024, 11, 18)  
 ),  
 Reflexion(  
 frase = "La vida es realmente simple, pero insistimos en hacerla complicada.",  
 autor = "Confucio",  
 fecha = LocalDate.of(2024, 11, 19)  
 )  
)

3.3.- Establece el diseño de la pantalla **Home** para que se muestre un recuadro con un color de la paleta debajo del texto **Frase del día.** En el recuadro se mostrará en cursiva el texto de la reflexión correspondiente a la fecha actual y su autor. El nombre del autor irá en negrita y alineado a la derecha.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamentePara hacer esto puedes componer:

Una columna que centra verticalmente e incluye:

El texto Frase del día (usa fontWeight para negrita)

Un spacer

Un box con un color background a elegir y bordes redondeados (parámetro shape en backgrund).

El box incluye una columna que contiene:

* El texto de la frase. Usa FontStyle.Italic y overflow = TextOverflow.Ellipsis para el comportamiento del texto cuando va a exceder el ancho.
* Un spacer
* El autor (con alineación derecha

3.4.- Para obtener la reflexión del día crea una función que devuelva la reflexión que tiene la fecha actual dentro de las reflexiones de data. Si no existe, se devuelve una reflexión con el texto “NO HAY FRASE DEL DÍA”.

Para buscar en reflexiones la que tiene la fecha actual, se usa la lambda:

*reflexiones*.*find* **{ it**.fecha == fechaActual **}**

4.- Ahora vamos a tratar de añadir a Home una imagen que represente a un audio más recientemente cargado en una lista.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

4.1.- Creamos un data class para representar los datos de un audio:

data class AudioMusical (  
 val titulo:String,  
 val artist:String,  
 val imagen : Int,  
 val audio:Int,  
 val fecha:LocalDate,  
 val estrellas:Double  
)

4.2- Creamos un archivo que contiene una lista de audios:

val *audiosMusicales* = *listOf*(  
 AudioMusical(  
 titulo = "El Encuentro",  
 artist = "Vetusta Morla",  
 imagen = R.drawable.*vetusta\_morla\_el\_encuentro*,  
 audio = 1,  
 fecha = LocalDate.of(2024, 11, 4),  
 estrellas = 4.6  
 ),  
 AudioMusical(  
 titulo = "Baile Existencialista",  
 artist = "León Benavente",  
 imagen = R.drawable.*leon\_benavente\_baile\_existencialista*,  
 audio = 2,  
 fecha = LocalDate.of(2024, 11, 5),  
 estrellas = 4.7  
 ),  
 AudioMusical(  
 titulo = "Devenir Paisaje",  
 artist = "Fin del Mundo",  
 imagen = R.drawable.*fin\_del\_mundo\_devenir\_paisaje*,  
 audio = 3,  
 fecha = LocalDate.of(2024, 11, 12),  
 estrellas = 4.6  
 ),  
 AudioMusical(  
 titulo = "Brujería!",  
 artist = "Judeline",  
 imagen = R.drawable.*judeline\_brujeria*,  
 audio = 4,  
 fecha = LocalDate.of(2024, 11, 8),  
 estrellas = 4.4  
 ),  
 AudioMusical(  
 titulo = "Nos Volveremos a Ver",  
 artist = "La La Love You",  
 imagen = R.drawable.*la\_la\_love\_you\_nos\_volveremos\_a\_ver*,  
 audio = 5,  
 fecha = LocalDate.of(2024, 11, 6),  
 estrellas = 4.6  
 ),  
 AudioMusical(  
 titulo = "Tenemos la respuesta",  
 artist = "ELYELLA feat. Lori Meyers",  
 imagen = R.drawable.*elyella\_tenemos\_la\_respuesta*,  
 audio = 6,  
 fecha = LocalDate.of(2024, 11, 10),  
 estrellas = 4.5  
 )  
)

4.3- Añadimos a la columna principal de **HomeView** un componente **AudioMasReciente** que muestre lo correspondiente a:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Esto se hace con:

Una columna que contiene:

* Un Texto con letra en negrilla y tamaño grande.
* Un Box que ocupa todo el ancho de pantalla y tiene altura 200.dp que contiene:
  + Una imagen que ocupa todo el espacio disponible.
  + Un **IconButton** con alineación centrada, tamaño 50.dp, fondo con color primario y **shape=CircleShape**, e incluyendo el icono **Icons.Filled.playarround** con color **tint** blanco y tamaño 32.dp.
  + Un Text que ocupa todo el ancho disponible, esta alineado verticalmente abajo y tiene un color de fondo amarillo semitransparente (**Color.Yellow.copy(alpha = 0.7f)).**

Para obtener de una lista de audio el más reciente, puedes usar:

**val audioMasReciente = *audiosMusicales*.*maxBy* { it.fecha }**

La imagen se carga con el componente **Image**:

Image(  
 painter = painterResource(id = audioMasReciente.imagen),  
 contentDescription = "Imagen del audio más reciente",  
 contentScale = ContentScale.Crop,  
 modifier = Modifier.fillMaxSize()  
)

Prueba lo que ocurre con otros valores de **ContentScale** para esa imagen o para otras que decidas. En este tema de la escala hay varios casos y se elije el escalado más adecuado en función sobre todo de cuanto mayores o menores sean las dimensiones de la imagen (archivo) respecto de las del contenedor (Image).

5.-Lo más normal en este tipo de aplicaciones, es que las imágenes se obtengan usando una **url** de recurso que nos permitirá acceder al recurso y descargarlo. Hay fundamentalmente dos librerías que facilitan la descarga por Internet y representación de imágenes**: coil y glide**. Usaremos **glide**.

5.1.- Usaremos un data class similar al anterior para que en los datos de la imagen y del audio se tengan urls en lugar de Int identificadores de recursos.

data class AudioMusicalExt (  
 val titulo:String,  
 val artist:String,  
 val urlImagen : String,  
 val urlAudio:String,  
 val fecha:LocalDate,  
 val estrellas:Double  
)

También crearemos una lista **audiosMusicalesExt**. Puedes copiar la de **audiosMusicales** y sustituir los datos de imagen y audio por datos de **urlImagen** y **urlAudio**. En **urlAudio** dejamos por el momento texto vacío. En **urlImagen** puedes poner la url de cualquier imagen, aunque no tengan relación con los artistas o con los audios musicales.

5.2.- Para poder usar **glide** en el proyecto hay que agregar la dependencia de **glide** al archivo Gradle de módulo.

*implementation*("com.github.bumptech.glide:compose:1.0.0-beta01")

5.3.- Ahora, en vez del componente **Image** que teníamos para representar las imágenes incluidas en **drawable,** usaremos la función **GlideImage de coil**. Esta función composable se encarga de gestionar la descarga y pintado de una imagen. La descarga la realiza dentro de una **corrutina** que se ejecuta en un hilo independiente del de la UI.

La sintaxis básica de **GlideImage** es:

GlideImage(  
 model = url\_descarga imagen,  
 contentDescription = "Imagen del audio más reciente",  
 modifier = Modifier.*fillMaxSize*(), //ocupa *todo el espacio disponible* contentScale = ContentScale.Crop, //escalado  
 // agregado de placeholder de carga y de error (opcional)  
 loading = *placeholder(R.drawable.archivo\_load)*,  
 failure = *placeholder*(R.drawable.archivo\_*error*)  
)

En **model** se especifica la url de la imagen que se quiere descargar y mostrar. En nuestro caso, tendremos que usar **urlImage** del objeto **audioMasReciente**.

Otros parámetros interesantes de **GlideImage** son:

* **contentScale**
* **failure** (recurso de imagen de **drawable** que se carga cuando no se puede descargar la imagen indicada en **model**)
* **loading** (recurso de la imagen de **drawable** que se mostrará mientras se está realizando la descarga de la imagen)

Puedes obtener urls de descarga de imágenes en **Flickr.com.** Visualizas una imagen y en el botón para descargarla, **copiar dirección de enlace** de uno de los tamaños de descarga.

Si pruebas la ejecución verás que no funciona. Es debido a que, dado que la aplicación va a usar Internet, requiere que se registre en el archivo de manifiesto (**AndroidManifest.xml)** el permiso de nivel normal INTERNET.

Texto

Descripción generada automáticamente

Los permisos de **nivel normal** no afectan a la privacidad y seguridad de las personas y sólo basta con registrarlos en la app. Los permisos que afectan a esas condiciones son de **nivel peligroso** y requerirán que sean aceptados por los usuarios al iniciar las apps, además de registrarse también en el archivo de manifiesto.

Un usuario de una app puede ver los permisos que se tienen registrados y, por tanto, se usarán en la app.