**Sesión 3-06 Clase del 18 de diciembre**

En esta sesión seguimos trabajan do con Room para desarrollar una app que incluye todas las funciones CRUD en sus funcionalidades.

Puedes continuar trabajando con el proyecto de la sesión anterior o usar un proyecto copia del anterior ya que seguimos trabajando con el mismo modelo de datos.

En el archivo **Archivos\_Sesion3\_06.rar** de la carpeta RECURSOS de Moodle tienes varios archivos fuente que puedes utilizar. Tienes los archivos de modelo de las tablas de la base de datos, los DAO y un archivo para el ViewModel que vamos a usar.

1.- En primer lugar, dado que vamos a usar MVVM, crea en tu proyecto un archivo **HomeViewModel.kt** para el ViewModel de las vistas a usar**.** Copia en ese archivo el contenido del mismo nombre que tienes en **Archivos\_Sesion3\_06.rar.** El archivo contiene:

* Un **LiveData** para proveer una lista de animales y otro para proveer un animal.
* Una función para obtener un animal de la tabla de animales a partir de su id.
* Una función para insertar en la tabla de familias el contenido de una lista de familias.
* Una función que devuelve en una lista todos los animales de la tabla de animales y la asigna al **LiveData**.
* Una función **iniciar** que añade una lisa de animales a la tabla de animales y después obtiene en una lista todos los animales de la tabla.

2.- Crear una pantalla **HomeView.kt** para mostrar una lista de animales resultado de los que ha suministrado el ViewModel. En primer lugar, como siempre que usamos un ViewModel, en **HomeView.kt** creamos una función composable que recibe el ViewModel.

@Composable  
fun HomeView(homeViewModel: HomeViewModel) { }

2.1.- Agregamos un dato para consumir la lista de animales que provee el ViewModel:

// quiero que animales se actualice con el contenido de animales de HomeViewModel  
val animales = homeViewModel.animales.observeAsState(initial = *emptyList*())

2.2.- Usaremos la siguiente función composable para representar cada elemento de la lista de animales:

@Composable  
fun AnimalBox(  
 animal: Animal,  
 onEditClick: (Animal) -> Unit, // Callback para editar  
 onDeleteClick: (Animal) -> Unit // Callback para eliminar  
) {  
 Row (  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*height*(60.*dp*)  
 .*background*(Color.Black)  
 .*padding*(horizontal = 8.*dp*),  
 verticalAlignment = Alignment.CenterVertically  
 ) **{** // Texto del nombre del animal  
 Text(  
 text = animal.nombre,  
 color = Color.White,  
 fontSize = 30.*sp*,  
 textAlign = TextAlign.Start,  
 fontWeight = FontWeight.Bold,  
 modifier = Modifier.*weight*(0.8f)  
 )  
  
 // IconButton para editar (superpuesto en la esquina superior derecha)  
 IconButton(  
 onClick = **{** onEditClick(animal) **}**,  
 modifier = Modifier.*weight*(0.1f)  
 ) **{** Icon(  
 imageVector = Icons.Default.*Edit*,  
 contentDescription = "Editar",  
 tint = Color.White  
 )  
 **}** // IconButton para eliminar  
 IconButton(  
 onClick = **{** onDeleteClick(animal) **}**,  
 modifier = Modifier.*weight*(0.1f)  
 ) **{** Icon(  
 imageVector = Icons.Default.*Delete*,  
 contentDescription = "Eliminar",  
 tint = Color.White  
 )  
 **}  
 }**}

Como puedes ver la función recibirá tres parámetros: El animal a representar, una función para editar y otra función para eliminar el animal.

2.3.- Ahora en **HomeView** debes crear un **LazyColumn** para representar la lista de animales usando para cada item el composable **AnimalBox**. En la llamada a **AnimalBox,** en los parámetros lambda puedes pasar esto para probar:

onEditClick = **{** animalToEdit **->** // Acción de ejemplo para editar   
 Log.i("ROOM\_PRUEBA","Editar: ${animalToEdit.nombre}")  
**}**,  
onDeleteClick = **{** animalToDelete **->** // Acción para eliminar (puedes mostrar una confirmación)  
 Log.i("ROOM\_PRUEBA","Eliminar: ${animalToDelete.nombre}")  
**}**

2.4.- Ahora dentro de MainActivity tenemos que instanciar el ViewModel y componer HomeView:

val database = DatabaseProvider.getDatabase(this)  
val homeViewModel= HomeViewModel(database)  
*setContent* **{** Sesion3\_06 **{** HomeView(homeViewModel)  
 **}  
}**

2.5.- Si ejecutarás ahora, no obtendrías nada ya que en ningún momento se han cargado datos en la base de datos ni tampoco se han consultado.

Para que se pueda hacer un lanzamiento de la carga de los datos en la base de datos y de una consulta inicial, debemos llamar a la función **iniciar** del ViewModel dentro de una función **LaunchedEffect** que se ejecuta de forma automática dentro de una corrutina. Esta función debes incluirla dentro del composable **HomeView.**

// Llamar a la función para cargar los animales y familias inicialmente  
// Esto debería realizarse dentro de una clave segura como `LaunchedEffect`   
// para evitar múltiples llamadas  
LaunchedEffect(Unit) **{** homeViewModel.iniciar()  
**}**

De momento, esto hace que esta parte se ejecute sólo la primera vez que se compone la pantalla y no las siguientes, cuando por ejemplo se añade un nuevo animal.

3.- Ejecutando inicialmente el funcionamiento debería ser el esperado, se muestra una lista con los 5 animales. Pero se ejecutas otra vez verás que la lista se duplica. Esto es debido a que la app ya está instalada y, por tanto, se mantienen los datos al cerrarla. Cuando se vuelve a inicia, se compone la pantalla por primera vez y se ejecuta el código de **LaunchedEffect**, con lo cual se añaden otra vez los mismos animales a la base de datos.

Esto por ahora no lo vamos a solucionar, simplemente desinstala la aplicación en el dispositivo, vuelve a ejecutar y tras esa primera ejecución, elimina o comenta el código que inserta los animales de la lista y vuelve a ejecutar.

En la siguiente sesión veremos una forma correcta de cargar inicialmente datos en la base de datos de una aplicación de forma que esos datos solo se carguen en la primera ejecución.

4.- Vamos a programar la funcionalidad **Eliminar** mediante un cuadro de diálogo que pregunte si acatamos eliminar el animal o lo cancelamos.

4.1.- En el ViewModel creamos una función para eliminar que llama a delete del DAO y vuelve a recargar la lista de animales (esta recarga es necesaria porque si no se hace, el ViewModel no notifica su modificación y por ello no se vería la lista de animales actualizada:

// Función para eliminar un animal  
fun eliminarAnimal(animal: Animal) {  
 *viewModelScope*.*launch*(Dispatchers.IO) **{** appDatabase.animalDao().deleteAnimal(animal)  
 obtenerAnimales() // Recargar la lista de animales

**}**}

4.2.- En **HomeView** necesitamos declarar dos variables de estado para controlar el animal que se querrá eliminar y si el cuadro de diálogo está visible o no:

// Estado para controlar la visibilidad del diálogo de eliminación  
// y estado para controlar el animal a eliminar  
var animalAEliminar by remember **{** *mutableStateOf*<Animal?>(null) **}**var showDialog by remember **{** *mutableStateOf*(false) **}**

4.3.- En **HomeView** añadimos el composable para representar el cuadro de diálogo. No explicamos el código pues ya lo vimos en otra sesión anterior. La función recibe el animal a eliminar, la función que se realiza al confirmar y la que se realizaría al cancelar:

@Composable  
fun ConfirmDeleteDialog(  
 animal: Animal,  
 onConfirm: (Animal) -> Unit,  
 onDismiss: () -> Unit  
) {  
 AlertDialog(  
 onDismissRequest = **{** onDismiss() **}**,  
 confirmButton = **{** TextButton(onClick = **{** onConfirm(animal) **}**) **{** Text(text = "Aceptar")  
 **}  
 }**,  
 dismissButton = **{** TextButton(onClick = **{** onDismiss() **}**) **{** Text(text = "Cancelar")  
 **}  
 }**,  
 title = **{** Text(text = "Confirmar eliminación")  
 **}**,  
 text = **{** Text(text = "¿Estás seguro de que deseas eliminar a ${animal.nombre}?")  
 **}** )  
}

4.4.- En la función composable **HomeView** debemos añadir después del **LazyColumn** el código para mostrar el cuadro de diálogo cuando la variable de estado esté a true:

// Mostrar cuadro de diálogo si `showDialog` es verdadero  
if (showDialog ) {  
 ConfirmDeleteDialog(  
 //Confía en mi, esta variable no es nula  
 animal = animalAEliminar!!,  
 onConfirm = **{** homeViewModel.eliminarAnimal(**it**)  
 showDialog = false  
 **}**,  
 onDismiss = **{** showDialog = false  
 **}** )  
}

4.5.- Ya por último tenemos que cambiar el código del parámetro **onDeleteClick** en la llamada a para que active el estado de mostrar el cuadro de diálogo y el estado del animal a eliminar:

onDeleteClick = **{** animalToDelete **->** animalAEliminar = animalToDelete  
 showDialog = true  
**}**