**Sesión 2-08 Clase del 20 de noviembre**

Continuamos con el proyecto de la sesión 2-07.

1.- Vamos primero a aprender a trabajar con **RadioButtons**. Para ello vamos a incluir una encuesta dentro de la vista Home.

Texto

Descripción generada automáticamente con confianza media

1.1.- La encuesta responde a una pregunta eligiendo sólo una de entre varias respuestas posibles. Creamos en **model** una **sealed class** (donde definimos dos objetos encuestas):

sealed class Encuesta (pregunta:String, respuestas:List<String>)  
{  
 object encuestaHoy:Encuesta(  
 "¿cual es el módulo que menos te gusta?",  
 *listOf*("AD","DI","PSP","PMDM")  
 )  
 object encuestaAyer:Encuesta(  
 "¿que gardo de satisfacción tienes con el profe de PMDM?",  
 *listOf*("Totalmente satisfecho","Bastante satisfecho",  
 "Poco satisfecho","Nada satisfecho")  
 )  
}

1.2.- Añade una función composable **EncuestaSection** para implementar la encuesta dentro de **HomeView().**

1.3.- Incluye este código base de ejemplo de como implementar un texto, varios **RadioButtons**, y un botón enviar. Incluye el código en **EncuestaSection().** Analiza el código. Tienes varios comentarios que te deberían ayudar a comprenderlo. Si no es así, consulta tus dudas. Comprueba la ejecución.

fun EncuestaSection() {  
 Column(modifier = Modifier.*fillMaxWidth*()) **{** var respuestaSeleccionada by remember **{** *mutableStateOf*<String?>(null) **}** Text(text = "valora esto", fontSize = 20.*sp*)  
 val respuestas= *listOf*("bien","normal","mal")  
 respuestas.*forEach* **{** respuesta **->** Row(  
 modifier = Modifier.*fillMaxWidth*().*height*(30.*dp*),  
 verticalAlignment = Alignment.CenterVertically  
 ) **{** RadioButton(  
 selected = respuesta == respuestaSeleccionada,  
 onClick = **{** respuestaSeleccionada = respuesta  
 **}** )  
 Spacer(modifier = Modifier.*width*(8.*dp*))  
 Text(text = respuesta)  
 **}  
 }** Button(  
 onClick = **{ }** ) **{** Text(text = "Enviar")  
 **}  
 }** }

Comprueba lo que ocurre cuando la variable de estado la inicias con una de las respuestas.

1.4.- Añade la siguiente funcionalidad al botón **Enviar**:

Se escribe un Toast con el texto de la respuesta seleccionada y con el índice de la respuesta en la lista de respuestas. Si no hay respuesta seleccionada, se debe escribir un Toast indicando de esa situación.

Recuerda que, en el primer parámetro de un Toast, tienes que usar un objeto contexto y que ese objeto se crea con:

val context = *LocalContext*.current

1.5.- Añade la siguiente funcionalidad al **onClick** sobre los **RadioButtons:**

Se escribe un Toast indicando “*Has pulsado justo en el* ***RadioButton*** *de la respuesta ….*.”

1.6.- Sustituye la pregunta y las respuestas por las del objeto **PreguntadeHoy** definido en el apartado 1.1.

Para, por ejemplo, acceder a la pregunta de ese objeto, tienes que usar:

Text(text = Encuesta.EncuestaHoy.pregunta, fontSize = 20.*sp*)

2.- Ahora vamos a añadir funcionalidad al botón Enviar para que, cuando se pulse y haya pregunta seleccionada, se inhabilite ese botón.

* Primero hay que crear una variable de estado **encuestaActivada** para contener si va a estar o no habilitada la encuesta. El valor inicial será **true.**
* En el Button **Enviar,** cuando haya respuesta seleccionda, asignaremos a la propiedad **enabled** el valor de esa variable.
* En **onClick de Enviar** asignamos a la variable de estado el valor **false.**

*Usando esa variable de estado también podríamos desactivar los* ***RadioButtons.***

3.- Vamos a usar ahora la variable de estado anterior para que, cuando esté desactivada la encuesta, el contenido de la encuesta sea totalmente transparente.

Eso es fácil, tienes que usar el modificador **Alpha** en la **Column** raíz de **EncuestaSection** y asignarle un valor en función de la variable de estado **encuestaActivada.**

Column(  
 modifier = Modifier  
 .*fillMaxWidth*()  
 .*alpha*(if (encuestaActivada) 1.0f else 0.0f)

*Pero la encuesta sigue ocupando el mismo espacio que ocupaba en pantalla. Si añadiéramos un componente debajo de* ***EncuestaSection****(), veríamos que ese espacio queda vacío.*

4.- Si en lugar de lo anterior quisiéramos que la columna no se pintara (no ocupando espacio) cuando está desactivada la encuesta, deberíamos meter todo el código de **Column**  dentro de un:

if(encuestaActivada) {

5.- Ahora vamos a pasar a implementar la navegación de la **BottonBar.** Lo que hagamos, sería igualmente válido pata la **TopBar.**

Vamos a usar el código del proyecto **AppMusica\_Base2** que tiene implementadas todas las funcionalidades desarrolladas hasta ahora.

* En **MiMusicaView.kt** puedes observar que tenemos implementada la función para componer **BottomBar**  mediante componentes **NavigationBar y NavigationBarItem.**
* Se tiene un archivo **ArtistsView.kt** con la función composable para la pantalla de **Artists** (realmente la parte de pantalla que se incluirá en el **Scaffold** de **MiMusicaView**).
* En el archivo **Gradle de module** se carga la librería para realizar navegación.

6.- Crea los archivos y funciones para componer las pantallas **SearchView, MyAppView y PremiumView** copiando a partir de **ArtistsView.**

7.- Creamos un **enum** **class NavView** (o el nombre que queramos) para identificar con sus valores las rutas de navegación que vamos a crear.

enum class NavView {  
 *Home*,  
 *Artists*,  
 *Search*,  
 *MyApp*,  
 *Premium*}

8.- Pasamos a desarrollar el sistema de navegación. En un package **navigation** creamos un archivo **MiMusicaNavigation** con una función composable **MiMusicaNavigation** que debe recibir un parámetro de tipo **NavHostController.**

@Composable  
fun MiMusicaNavigation(navController: NavHostController){  
   
  
}

En la función incluimos un componente **NavHost** cuya ruta de destino inicial sea la correspondiente a **HomeView** (por ahora, lo del enumerado).

@Composable  
fun MiMusicaNavigation(navController: NavHostController){  
 NavHost(navController = navController,  
 startDestination = NavView.*Home*.name)  
 **{** //aqui asociaremos las rutas de navegación con su correspondiente View  
   
 **}**}

Dentro del **scope de NavHost** pasamos a definir las View que se corresponden con cada ruta de navegación. Por ejemplo, para la ruta **NavView.Home:**

*composable*(NavView.*Home*.name)**{** HomeView() **}**

Asocia las otras cuatro rutas.

9.- Ahora vamos a definir en **model** una **sealed class** para que contenga los datos de cada botón de la barra de navegación (icono, texto y ruta que abre el botón).

sealed class Items\_Bottom\_Nav (  
 val icono:ImageVector,  
 val texto:String,  
 val ruta:String  
){  
 object Item\_bottom\_nav\_home:Items\_Bottom\_Nav(  
 Icons.Filled.*Home*,"Home",NavView.*Home*.name)  
 object Item\_bottom\_nav\_artists:Items\_Bottom\_Nav(  
 Icons.Filled.*List*,"Artists",NavView.*Artists*.name)  
 object Item\_bottom\_nav\_search:Items\_Bottom\_Nav(  
 Icons.Filled.*Search*,"Search",NavView.*Search*.name)  
 object Item\_bottom\_nav\_myapp:Items\_Bottom\_Nav(  
 Icons.Filled.*Person*,"MyApp",NavView.*MyApp*.name)  
 object Item\_bottom\_nav\_premium:Items\_Bottom\_Nav(  
 Icons.Filled.*Star*,"Premium",NavView.*Premium*.name)  
}

10.- Ahora tenemos que hacer modificaciones en la función **MiMusicaBottonBar (archivo MiMusicaView.kt).**

* La función debe recibir un parámetro de tipo **NavHostController**.
* En la función declararemos una lista con los objetos de la **sealed class** creada anteriormente. Podríamos no usar la lista y cargar uno a uno los ítems correspondientes a esos objetos, pero lo vamos a hacer con la lista para que el código sea más reducido y eficiente.
* Dentro del **NavigationBar** que tenemos, eliminamos todos los componentes **NavigationBarItem.**

@Composable  
private fun MiMusicaBottomBar(navController: NavHostController) {  
 val bar\_items= *listOf*(Items\_Bottom\_Nav.Item\_bottom\_nav\_home,  
 Items\_Bottom\_Nav.Item\_bottom\_nav\_artists,  
 Items\_Bottom\_Nav.Item\_bottom\_nav\_search,  
 Items\_Bottom\_Nav.Item\_bottom\_nav\_myapp,  
 Items\_Bottom\_Nav.Item\_bottom\_nav\_premium)  
 NavigationBar(  
 containerColor = MaterialTheme.colorScheme.surfaceVariant,  
 ) **{  
  
 }**}

11.- Dentro del contenido de **NavigationBar** recorremos la lista de ítems y, con su contenido creamos componente **NavigationBarItem.**

bar\_items.*forEach* **{** it ->  
 NavigationBarItem(selected = false,  
 onClick = **{** /\* pendiente de realizar \*/ **}**,  
 icon = **{** Icon(imageVector = it.icono, contentDescription = null)**}**,  
 label = **{**Text(text=it.texto)**}** )  
**}**

12.- El **NavHostController** que se recibe en esta función hay que instanciarlo o crearlo en algún momento para que llegue a ésta y otras funciones.

Se debe hacer en la función **composable** de mayor nivel de la aplicación, es decir, en **MiMusicaView().**

val navController= rememberNavController()

Ese objeto se debe pasar a la función **MiMusicaBottomBar**:

fun MiMusicaView() {  
 val navController= rememberNavController()  
 Scaffold(  
 topBar = **{** MiMusicaTopBar() **}**,  
 bottomBar = **{** MiMusicaBottomBar(navController) **}**,  
 content = **{** HomeView()**}** )  
}

Si probamos la ejecución, veremos que se pintan los botones en la **BottomBar,** pero que no hay funcionalidad alguna en ellos.

13.- Para que haya la funcionalidad requerida en esos botones:

* En **content** del **Scaffold** de **MiMusicaView** hay que llamar a la función composable de navegación pasándole el **navController**.

Scaffold(  
 topBar = **{** MiMusicaTopBar() **}**,  
 bottomBar = **{** MiMusicaBottomBar(navController=navController) **}**,  
 content = **{** MiMusicaNavigation(navController = navController)**}**)

* En **onClick** de los **NavigationBarItems** hay que indicar que se navegue a la ruta correspondiente a cada botón.

NavigationBarItem(selected = false,  
 onClick = **{** navController.navigate(it.ruta) **}**,

Al ejecutar, todo debería funcionar bien. Sin embargo, debería estar marcado de alguna forma el botón que se ha seleccionado en la **BottomBar**

14.- La siguiente función nos devolverá el String correspondiente a la ruta seleccionada con el botón que se hay clicado.

@Composable  
fun currentRoute(navController: NavHostController):String?=  
 navController.currentBackStackEntryAsState()  
 .value?.destination?.route

Tenemos actualmente que todos los botones de **NavigationBar** se están pintando como no seleccionados (**selected**=false). Podemos llamar a la función anterior para que se pinte cada botón en el estado **selected** en que se encuentre realmente.

bar\_items.*forEach* **{** it ->  
 // en clicked queda true si se ha seleccionado el botón que se está pintando  
 // se está comparando la ruta del seleccionado con la ruta del botón actual  
 val clicked= currentRoute( navController)==it.ruta  
 //la propiedad selected se pone al valor de clicked  
 NavigationBarItem(selected = clicked,