PRÁCTICA DE VIEWMODEL Y LIVEDATA

OpenWebinars

PRÁCTICA DE VIEWMODEL Y LIVEDATA

PASO 1:

- Partimos de una aplicación que contiene un componente **TextField** y un componente **Text** que refleja los cambios que se producen en **TextField** cuando el usuario introduce texto en él.

PRÁCTICA DE VIEWMODEL Y LIVEDATA

State en ViewModel:

- MainScreen

```
@Composable
fun MainScreen() {
    val nameState = remember { mutableStateOf( value: "") }
    Surface(
        color = Color.LightGray,
        modifier = Modifier.fillMaxSize()
    ) {
        MainLayout(
            nameState.value
        ) { newName -> nameState.value = newName }
    }
}
```

PRÁCTICA DE VIEWMODEL Y LIVEDATA

State en ViewModel:

- MainLayout

```
@Composable
fun MainLayout(
    name: String,
    onTextFieldChange: (String) -> Unit
) {
    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally
    ) { this: ColumnScope
        TextField(
            value = name,
            onValueChange = onTextFieldChange
        )
        Text(text = name)
    }
}
```

PRÁCTICA DE VIEWMODEL Y LIVEDATA

PASO 1:

- El siguiente paso será mover **nameState** a un componente **ViewModel**. Para ello, creamos una nueva clase **MainViewModel** que herede de **ViewModel** como se muestra a continuación:

PRÁCTICA DE VIEWMODEL Y LIVEDATA

State en ViewModel:

- MainViewModel

```
class MainViewModel: ViewModel() {

    val textFieldState = MutableLiveData( value: "")

    fun onTextChange(newText: String) {
        textFieldState.value = newText
    }
}
```

PRÁCTICA DE VIEWMODEL Y LIVEDATA

State en ViewModel:

- **textFieldState: MutableLiveData** refleja ahora el estado del dato al cual nuestra UI tendrá que suscribirse para recibir actualizaciones.

- A través del método público **onTextChange**, la UI mandará el evento de cambio de texto que genere el componente **TextField**

- Para leer los datos de nuestro nuevo **MainViewModel** desde la vista **MainScreen** tendremos que modificar el componente de la siguiente forma.

PRÁCTICA DE VIEWMODEL Y LIVEDATA

State en ViewModel:

- MainScreen

```
@Composable
fun MainScreen(viewModel: MainViewModel = MainViewModel()) {
    val nameState = viewModel.textFieldState.observeAsState( initial: "")
    Surface(
        color = Color.LightGray,
        modifier = Modifier.fillMaxSize()
    ) {
        MainLayout(
            nameState.value
        ) { newName -> viewModel.onTextChange(newName) }
    }
}
```

PRÁCTICA DE VIEWMODEL Y LIVEDATA

State en ViewModel:

- El valor de **nameState** proviene ahora del componente **LiveData** definido en **MainViewModel**.

- Necesitamos que **nameState** sea un **State** y no un **LiveData**. Hay que añadir una nueva dependencia al fichero **build.gradle** permitiendo el uso del método **observeAsState** encargado de la conversión a **State**:

    - **implementation "androidx.compose.runtime:runtime-livedata:$compose_version"**

PRÁCTICA DE VIEWMODEL Y LIVEDATA

State en ViewModel:

- Los eventos de **TextField** recogidos en la lambda son enviados ahora a nuestro **MainViewModel** y a su vez notificados a **LiveData** a través del método **onTextChange**.