CÓMO FUNCIONA STATE

CÓMO FUNCIONA STATE

Recomposición:

- Proceso que se encarga de actualizar la pantalla, en concreto, los componentes que admiten composición.

- Para lanzar la recomposición es indispensable tener una implementación de **State** para cada componente composable, al menos para los que tienen un estado que cambia a lo largo del tiempo.

CÓMO FUNCIONA STATE

State:

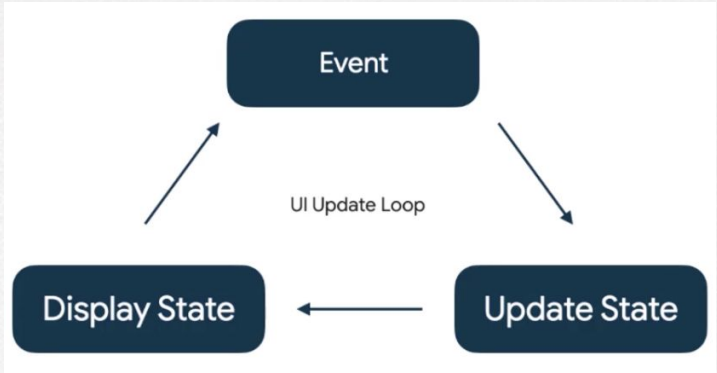
- El **State** de una aplicación se puede definir como ***cualquier valor que puede cambiar a lo largo del tiempo***.

- En Jetpack Compose **State** es un componente más del propio componente composable.

CÓMO FUNCIONA STATE

Flujo de datos unidireccional:

- Bucle en el que se dispara un evento que actualiza un **State** (un click a un botón que desencadena la actualización de una lista).
- Este nuevo valor de **State** pasa por todo el árbol de la UI de elementos que deben tener en cuenta sus posibles valores y actualizar la UI.



CÓMO FUNCIONA STATE

Flujo de datos unidireccional (Ventajas):

- **Mayor testeabilidad: State** está desacoplado de la UI, es muy fácil hacer tests de ambas partes de forma aislada.

- **Mayor consistencia en la UI:** Este flujo obliga a que todos los **State** sean reflejados en la UI de forma continua eliminando las posibles inconsistencias entre los componentes visuales y los estados.

CÓMO FUNCIONA STATE

Controlar **State** en una lista:

- Partimos de un componente **MainScreen** que contiene una lista **StudentList** de componentes **StudentText** y un **Button** que añade nuevos elementos a la lista de estudiantes.

CÓMO FUNCIONA STATE

MainScreen

```
@Composable
fun MainScreen() {
    Surface(
        color = Color.LightGray,
        modifier = Modifier.fillMaxSize()
    ) {
        StudentList()
    }
}
```

CÓMO FUNCIONA STATE

StudentList

```
@Composable
fun StudentList() {
    val students = mutableListOf("Juan", "Victor", "Esther", "Jaime")
    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {  this: ColumnScope
        for (student in students) {
            StudentText(name = student)
        }
        Button(
            onClick = { students.add("Miguel") },
        ) {  this: RowScope
            Text(text = "Add new student")
        }
    }
}
```

CÓMO FUNCIONA STATE

StudentText

```
@Composable
fun StudentText(name: String) {
    Text(
        text = name,
        style = MaterialTheme.typography.h5,
        modifier = Modifier.padding(10.dp)
    )
}
```

CÓMO FUNCIONA STATE

- Si activamos el modo interactivo y pulsamos el botón añadir podemos observar cómo la lista no añade el nuevo valor aunque modifiquemos la lista de estudiantes

- Esto es debido a que no se ha implementado ningún **State** a la lista de datos que dispare la recomposición.

CÓMO FUNCIONA STATE

- Para añadir **State** a la lista es necesario crear la lista del tipo **SnapshotStateList** a través del método **mutableStateListOf**:
    - **val studentsState = mutableStateListOf("Esther", "Jaime")**

CÓMO FUNCIONA STATE

- Observamos que el compilador nos obliga a utilizar el bloque **remember**. Permite que el estado sea recordado durante la recomposición y que no desaparezca después:
    - **val studentsState = remember { mutableStateListOf("Esther", "Jaime") }**

CÓMO FUNCIONA STATE

Finalmente, **StudentList** queda de esta forma:

```
@Composable
fun StudentList() {
    val studentsState = remember { mutableStateListOf("Juan", "Victor", "Esther", "Jaime") }
    Column(
        modifier = Modifier.fillMaxSize(),
        horizontalAlignment = Alignment.CenterHorizontally
    ) { this: ColumnScope
        for (student in studentsState) {
            StudentText(name = student)
        }
        Button(
            onClick = { studentsState.add("Miguel") },
        ) { this: RowScope
            Text(text = "Add new student")
        }
    }
}
```