**Sesión 2-06 Clase del 13 de noviembre**

Se tratará de aprender a usar el patrón MVVM. Trata de aislar la lógica de negocio de la aplicación del modelo y de la interfaz. Las vistas (**view**) se limitan a escribir datos y a recibir datos. Cuando una vista recibe datos se los pasa a un **ViewModel** que, si es necesario, les aplica la lógica de negocio. En el **ViewModel** se modificarán algunos de sus datos que implicará que las vistas que los observen, escriban sus valores.

Diagrama, Escala de tiempo

Descripción generada automáticamente

MVVM se divide en tres partes:

* Modelo de datos usado (model)
* Diseño de la vista (view)
* Lógica de la vista (ViewModel)

1.- Partimos de un proyecto base en el que haremos todas las modificaciones necesarias para darle funcionalidad y que use el patrón MVVM.

Abrimos el proyecto **Sesion2\_06\_Contador\_Base.**

Vemos que tiene un diseño muy sencillo que incluye un Texto (contador) y un botón de acción flotante (incrementar).

No tiene funcionalidad actualmente. Implementaremos MVVM para que cuando se pulse el botón, se incremente el valor del contador.

2.- Creamos un paquete **viewmodels** y dentro, una nueva clase para implementar el **ViewModel**. A la clase le damos el nombre **ContadorViewModel** y debe extender la clase **ViewModel.**



3.- Vamos a añadir en **ContadorViewModel** un **LiveData** para controlar el valor del contador que vamos a usar**.**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

*Hemos definido un* ***LiveData*** *para contador. El que la variable private se nombre comenzando por guión bajo y que las dos variables se nombren igual es por convención de nombrado y caridad de código.*

*La variable* ***\_contador*** *se maneja dentro de la clase* ***ContadorViewModel.*** *La variable* ***contador,*** *al ser pública, se podrá manejar desde el exterior de la clase pero sólo para acceder a su valor.*

4.- El **ViewModel** se deberá usar ya desde **MainActivity** y pasarse hacia abajo a las clases y funciones que lo usen.

Texto

Descripción generada automáticamente

Y ahora haya que modificar **ContadorView** para que reciba ese parámetro **viewModel**.



Ahora en **ContadorView** adjuntamos un **observador** para la variable **contador** del **LiveData. el** (vamos, que cuando cambie el valor en el **ViewModel**, se va a modificar aquí). Establecemos también que esa variable observador se comporte como variable de estado.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

*Debemos añadir dependencia* ***runtime-livedata*** *para que se pueda trabajar con observadores de un* ***viewModel****.*

5.- Ahora pasaremos el observador **contador** al componente Text (*Enganchamos el componente* ***Text*** *con el* ***ViewModel*** *a través de una variable observador)*.

Texto

Descripción generada automáticamente

6.- Añadimos a **ContadorViewModel** una función de lógica de negocio que necesitamos. La función incrementará el valor de \_**contador** en la clase **ContadorViewModel**.

Texto

Descripción generada automáticamente

Quizás te resulte más sencillo hacerlo así:

Texto

Descripción generada automáticamente

Ya puedes probar la funcionalidad de la app. Al ejecutarlo se está produciendo lo siguiente:

a.- En **MainActivity** se construye el **ContadorViewModel** y se pasa a **ContadorView.**

b.- Al construirlo, se está asignando a la variable **\_contador de ContadorViewModel** el valor 0.

c.- Al construirlo el valor de **\_contador** se está asignando a la variable **contador** del **LiveData.**

d.- Como esa variable es observada por la variable **contador** de **ContadorView** entonces **contador de LiveData** notifica a **contador** de **ContadorView** de que se ha actualizado con el valor 0.

e.- Como **contador** de **ContadorView** es también variable de estado, se notifica al componente **Text** su valor (ya que lo usa). Entonces se escribe el valor 0 en el Text.

f.- Cuando se pulsa el botón +, se estará llamando a la función **add** de **ContadorViewModel.** En ese método se incrementa **\_contador** y la variable **contador** del **LiveData** a valor 1.

g.- Con la nueva modificación se notifica al observador **contador** de **ContadorView** y el cambio hace que se vea el Text con el nuevo valor.

7.- Vamos ahora a realizar otra aplicación en la que se generan aleatoriamente los números de alumnos que participan en un grupo de trabajo. Se va a usar MVVM y te indico los pasos a seguir:

7.1.- Descarga el proyecto **Sesion2\_06\_A\_Grupo\_Base.** Abre el proyecto, analiza el diseño de su pantalla y comprueba que es ejecutable, pero sin funcionalidad.

7.2.- Crea una clase **ViewModel** con un nombre apropiado. Como se va a generar una lista de 4 números que será la que se pinta en la pantalla, añade a la clase ViewModel un **LiveData** para una lista de números enteros. La variable privada se debe inicializar a una lista vacía.

7.3.- Añade a la clase **ViewModel** el siguiente método que permite generar 4 números aleatorios y añadirlos a una lista.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

7.4.- Haz que desde **MainActivity** se pase el **ViewModel** a **GrupoAleatView.**

7.5.- En **GrupoAleatView:**

* Declara una variable observador para la lista de números.
* Pasa la lista al componente que pinta la lista de números.
* Pasa la función **generateGrupoNumbers** al **onClick** del botón **Generar.**