Desafío - MonsterCreator App2

- Para realizar este desafío debes haber estudiado previamente todo el material disponibilizado correspondiente a la unidad.
- Una vez terminado el desafío, comprime la carpeta y sube el `.zip`

Descripción

A continuación vamos a poner a prueba tus conocimientos en la construcción de una aplicación utilizando un patrón de arquitectura MVVM, Room, Livedata y también Databinding.

En esta oportunidad continuaremos con la aplicación que realizamos en el desafío Anterior. Si por algún motivo no lograste terminar el desafío anterior, te proporcionaremos el proyecto en un estado inicial para este Desafío.

La idea principal de MonsterCreator es ser una aplicación que permite obtener un listado de monstruos creados. Mostrando una imagen, nombre y monsterPoints. En la MainActivity se mostrará a través de un RecyclerView esta información.

Ahora falta añadir el código necesario para que los usuarios puedan añadir sus monstruos por su cuenta, y que al ser añadidos se guarden en Room y se actualicen las interfaces de usuario a través de LiveData.

Instrucciones

Para la realización de este desafío, debes descargar el .zip que se encuentra en plataforma, llamado Apoyo Desafío - MonsterCreator App2 donde encontrarás, el material base para poder desarrollar los siguientes puntos:

1. Tu primera tarea es revisar el código facilitado, para verificar si el proyecto puedes ejecutarlo en tu máquina, resolver los posibles problemas de compatibilidad y descargar las versiones más nuevas de las librerías utilizadas.

   Revisa con cuidado la Actividad **MonsterCreatorActivity**, como puedes observar ya tiene creado su Layout, también existe un package llamado **monsteravatars** y en su interior contiene la implementación de un bottomDialog, con su respectivo adapter y recyclerView. Esto lo ocuparemos más adelante.

2. Debes mostrar la segunda actividad (MonsterCreatorActivity) para ello debes crear el intent en la acción del floating action Button de MainActivity. No olvides indicar en el manifest quien es la actividad padre. También debes activar Databinding en build.gradle(module: app).

```
dataBinding {

   enabled = true

}
```

3. Utilizar MutableLiveData en el ViewModel. Como puedes haber notado ya está creada la clase **MonsterCreatorViewModel**, pero debemos cambiar sus parámetros para adaptarlo a la forma que trabajamos. A diferencia del anterior viewModel que solo mostraba datos desde Room, ahora vamos a utilizar una clase que va a generar nuestro monstruo y calcular los puntos que tendrá cada uno. Esta clase estaba creada con anterioridad se llama “MonsterGenerator”. Debemos añadirla como parámetro a nuestro viewModel. (**2 Puntos**)

> TIP: En esta oportunidad Utilizaremos MutableLiveData(). No olvides utilizar la interface del repositorio en tu ViewModel.

4. Leer los datos desde la UI a través de ObservableField y crear las variables que te permitirán crear un objeto a través de la clase MonsterGenerator, para luego añadirlo a LiveData con el metodo PostValue().

Ejemplo:

```
var name = ObservableField<String>("")

var intelligence = 0

var ugliness = 0

var evilness = 0

var drawable = 0



fun updateCreature() {

   val attributes = MonsterAttributes(intelligence, ugliness, evilness)

   monster = generator.generateMonster(attributes, name.get() ?: "", drawable)

   monsterLiveData.postValue(monster)

}
```

5. Completa el ViewModel con los métodos necesarios para poder mostrar información en los spinners y otros elementos,

> TIP: Crea un archivo de kotlin en el package model conviertelo en un objeto:

```
object AttributeStore {

   val INTELLIGENCE: List<AttributeValue> by *azy
    {

       val monsterImages = *mutableListOf*<AttributeValue>()

       monsterImages.add(AttributeValue("Ninguno"))

       monsterImages.add(AttributeValue("Listillo", 3))

       monsterImages.add(AttributeValue("Vivaracho", 7))

       monsterImages.add(AttributeValue("Einstein", 10))

       monsterImages

     }

   val UGLINESS: List<AttributeValue> by lazy
    {



       val monsterImages = *mutableListOf*<AttributeValue>()

       monsterImages.add(AttributeValue("None"))

       monsterImages.add(AttributeValue("Feo", 3))

       monsterImages.add(AttributeValue("Adefesio feo", 7))

       monsterImages.add(AttributeValue("Feo feo", 10))

       monsterImages
   }

   val EVILNESS: List<AttributeValue> by lazy
   {

       val monsterImages = *mutableListOf <AttributeValue>()

       monsterImages.add(AttributeValue("None"))

       monsterImages.add(AttributeValue("malito", 3))

       monsterImages.add(AttributeValue("malulo", 7))

       monsterImages.add(AttributeValue("Mephistofeles", 10))

       monsterImages
```

```
    }

}
```

La idea es que podamos seleccionar estos atributos, para ellos debemos crear un método que pueda manejar los datos de nuestro archivo objeto attributeStore, este método debe ser así:

```
un attributeSelected(attributeType: AttributeType, position: Int) {
   when(attributeType) {
       AttributeType.INTELLIGENCE -> {
           intelligence = AttributeStore.INTELLIGENCE[position].value
       }
       AttributeType.UGLINESS -> {
           ugliness = AttributeStore.UGLINESS[position].value
       }
       AttributeType.EVILNESS -> {
           evilness = AttributeStore.EVILNESS[position].value
       }
   }
   updateCreature()
}
```

Esto permitirá que se seleccione lo que necesitamos en base a las clases y objetos que hemos creado. El resto de nuestro viewModel, deberá incorporar lo siguiente:

```
fun drawableSelected(drawable: Int) {
   this.drawable = drawable
   updateCreature()
}

fun saveCreature() {
   monsterRepository.saveMonster(monster)
   saveLiveData.postValue(true)
}
```

Estos métodos los usaremos más adelante para seleccionar la imagen y para guardar nuestro monstruo. El botón de nuestro layout está unido a este método gracias a DataBinding.

6. Debes añadir el ViewModel en la actividad **MonsterCreatorActivity** y completar la implementación.

a. Primero une la vista a la actividad con el layout, existe un error de nombre en este y no apunta a la actividad correcta, corrigelo.

> TIP: recuerda hacer Rebuild al proyecto si no aparece la clase que necesitas.

b. Ahora debes terminar la implementación del viewModel y los observadores en la actividad. Te dejare los métodos para controlar los spinners y añadirle los datos necesarios, pero tu tienes que terminar el resto.

```
private fun configureUI() {
    supportActionBar?.setDisplayHomeAsUpEnabled(true)
    title = "Añade una Monstruo"
    if (viewModel.drawable != 0) hideTapLabel()
}

private fun configureSpinnerAdapters() {
    intelligence.adapter = ArrayAdapter<AttributeValue>(
        this,
        android.R.layout.simple_spinner_dropdown_item,
AttributeStore.INTELLIGENCE
    )
    strength.adapter = ArrayAdapter<AttributeValue>(
        this,
        android.R.layout.simple_spinner_dropdown_item, AttributeStore.UGLINESS
    )
    endurance.adapter = ArrayAdapter<AttributeValue>(
        this,
        android.R.layout.simple_spinner_dropdown_item, AttributeStore.EVILNESS
    )
}

private fun configureSpinnerListeners() {
    intelligence.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
        override fun onItemSelected(
            parent: AdapterView<*>?,
            view: View?,
            position: Int,
            id: Long
        ) {
            viewModel.attributeSelected(AttributeType.INTELLIGENCE, position)
        }

        override fun onNothingSelected(parent: AdapterView<*>?) {}
    }
    strength.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
        override fun onItemSelected(
            parent: AdapterView<*>?,
            view: View?,
            position: Int,
            id: Long
        ) {
```

```
            viewModel.attributeSelected(AttributeType.UGLINESS, position)
        }

        override fun onNothingSelected(parent: AdapterView<*>?) {}
    }
    endurance.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
        override fun onItemSelected(
            parent: AdapterView<*>?,
            view: View?,
            position: Int,
            id: Long
        ) {
            viewModel.attributeSelected(AttributeType.EVILNESS, position)
        }

        override fun onNothingSelected(parent: AdapterView<*>?) {}
    }
}

private fun configureClickListeners() {
    avatarImageView.setOnClickListener {
        val bottomDialogFragment = MonsterBottomDialogFragment.newInstance()
        bottomDialogFragment.show(supportFragmentManager,
"AvatarBottomDialogFragment")
    }
}
```

Imágenes de referencia: