

Evidencia dia 05 - Semana 15

Leonardo Rodenas Escobar

Reflexión:

Clase de finalización de presentacion del tema hilos - corrutinas. Se presenta y se da la clase para realizar el ejercicio de la Lección 12 - Consolidación. Por mi parte usé una Api distinta a la del ejercicio para dar un poco de variación a los proyectos. Estuvo buena la clases, pero la modalidad de preguntar una cosas y que me respondan con 5 y quedar con más dudas de las que tenia no me agrada del todo.

Ejercicio:

Ejercicio de consolidación (Requerimientos generales)

1. El proyecto debe estar en un repositorio en la plataforma GitHub.
 2. Debe construir el proyecto con base en los requerimientos específicos solicitados, pero puede añadir su capa de customización.
 3. Está considerado construirlo en horas de clases, para responder las consultas, pero si trabaja fuera del horario no existe problema.
 4. En cada clase debe subir un commit y push hasta donde haya avanzado.
-
1. Debe utilizar algun patron de diseño recomendado (MVP, MVVM).
 2. Debe utilizar una clase Repositorio para tener una única fuente de datos.
 3. Debe implementar persistencia de datos en una base de datos utilizando la biblioteca Room.
 4. Debe implementar una interfaz para...

4. Debe conectarse a un servicio web y consumir datos, se recomienda un servicio específico.
5. Debe mostrar al menos tres pantallas, un RecyclerView con un listado de elementos.
6. Al hacer click sobre un elemento, debe mostrar un nuevo Fragmento con otro RecyclerView mostrando imágenes.
7. Al hacer un onLongClick sobre un elemento, debe guardar el objeto en la persistencia de datos como favoritos.
8. Debe haber un fragmento que muestre los objetos favoritos seleccionados en un RecyclerView. (Implementar un fab o menu).
9. En el fragmento de favoritos, debe ser posible eliminar un elemento o todos los

Por mi parte usaré la siguiente API:

[Últimos 15 Sismos en Chile - Centro Sismológico U de Chile](#)

Muestro acá parte del avance del proyecto, para ver el avance completo revisar en el siguiente [link](#).

Modelo:

```
@Entity (tableName = "tabla_sismos")
data class SismosModel(

    @PrimaryKey(autoGenerate = true)
    val id: Int, //Agregada por mi para llevar una id
    val horaLocal: String,
    val horaUtc: String,
    val latitud: String,
    val longitud: String,
    val magnitud: String,
    val mapa: String,
    val profundidad: String,
    val referencia: String,
    val favorito: Boolean //agregado por mí para guardar en favorito
```

```
) : Serializable
```

Service:

```
interface SismoService {  
  
    @GET(".")  
    fun obtenerSismos(): Call<List<SismosModel>>  
  
}
```

Dao:

```
@Dao  
interface SismoDao {  
  
    @Insert(onConflict = OnConflictStrategy.REPLACE)  
    suspend fun insertarTodosLosSismos(listaDeSismos: List<SismosModel>)  
  
    @Query("SELECT * FROM tabla_sismos")  
    fun obtenerTodosLosSismosDeLaBD(): LiveData<List<SismosModel>>  
  
    //1 = true || 0 = false --> revisar las comillas o no  
    @Query("SELECT * FROM tabla_sismos where favorito='1'")  
    fun obtenerLosSismosFavoritosDeLaBD(): LiveData<List<SismosModel>>  
  
    //ejecutado en consecuencia de si es favorito o no  
    @Update  
    fun cambiaEstadoFavorito (favorito: SismosModel)  
  
    //este borra  
    @Query("UPDATE tabla_sismos set favorito='0' where favorito='1'")  
    fun borraFavorito ()  
  
}
```

Repositorio:

```
class SismosRepositorio(private val sismosDao: SismoDao) {
```

```
private val service = SismoCliente.obtenCliente()
val miLiveData = sismosDao.obtenerTodosLosSismosDeLaBD()

//De Api a la base de datos
fun obtenerDataDelServer() {
    val call = service.obtenerSismos()
    call.enqueue(object : Callback<List<SismosModel>> {
        override fun onResponse(
            call: Call<List<SismosModel>>,
            response: Response<List<SismosModel>>
        ) {
            CoroutineScope(Dispatchers.IO).launch {
                response.body()?.let {
                    Log.v("logenrepo", response.body().toString())
                    sismosDao.insertarTodosLosSismos(it)
                }
            }
        }
    })

    override fun onFailure(call: Call<List<SismosModel>>, t: Throwable) {
        call.cancel()
    }
}

//para mostrar los sismos en la base de datos
fun exponeSismosDeLaBaseDeDatos(): LiveData<List<SismosModel>> {
    return sismosDao.obtenerTodosLosSismosDeLaBD()
}

//para mostrar los FAVORITOS en la base de datos
fun exponeFavoritosDeLaBaseDeDatos(): LiveData<List<SismosModel>> {
    return sismosDao.obtenerLosSismosFavoritosDeLaBD()
}

//este cambia el estado de si es favorito o no, pero tendria que cambiarlo acá
//no? o en el viewmodel?
fun cambiaeEstadoDeFavorito(sismo:SismosModel){
    sismosDao.cambiaEstadoFavorito(sismo)
}

//para BORRAR los FAVORITOS en la base de datos
fun borrarFavoritosDeLaBaseDeDatos() {
    sismosDao.borraFavorito()
}
}
```

Leonardo Rodenas Escobar