APAC 3 INCIDENCIAS

Entidades

La clase que representa una incidencia tiene como valores los siguientes_

```
package com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3.model.db

import java.io.Serializable
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity
data class Incidencia(
    @PrimaryKey(autoGenerate = true)var id:Long = 0,
    var assumpte: String,
    var descripcio: String,
    var ubicacio: String,
    var servei: String, // Jardineria, Infraestructures, Obres
    var img: Int,
    var resolt: Boolean
): Serializable
```

La clase Incidencia El vol de entero será long autogenerado con valor inicial de 0

Clase DAO

Inserción de una incidencia, como un método de suspensión, y de manera que si existe conflicto a

la base de datos, reemplazar el registro,

- Actualización de una incidencia, como un método de suspensión,

- borrado de una incidencia, con un método de suspensión,

- Consulta de todas las incidencias. Este método devolverá un LiveData, con la lista de incidencias

de incidencias.

```
package com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3.model.db

import androidx.lifecycle.LiveData
import androidx.room.*

/* Classe d'Accés a Dades */

@Dao
interface IncidenciaDAO {

    /* Mètodes de conveniència */

    // Afegir un incidencia
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun addIncidencia(incidencia: Incidencia)

    // Actualitzar un incidencia
    @Update
    suspend fun updateIncidencia(incidencia: Incidencia)

    // Eliminar un incidencia
    @Delete
    suspend fun deleteIncidencia(incidencia: Incidencia): Int

    /* Mètodes de cerca: Consultes */

    // Obtindre la llista de incidencia
    @Query("SELECT * from Incidencia")
    fun getAll(): LiveData<List<Incidencia>>


}
```

Creación Base de datos

```kotlin
package com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3.model.

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@Database(entities = [Incidencia::class], version = 1, )
abstract class IncidenciasDB : RoomDatabase() {
    abstract fun incidenciaDAO(): IncidenciaDAO
}
object DatabaseBuilder {
    private var INSTANCE: IncidenciasDB? = null

    fun getInstance(context: Context): IncidenciasDB? {
        if (INSTANCE == null) {
            synchronized(IncidenciasDB::class) {
                INSTANCE = buidRoomDB(context)
            }
        }
        return INSTANCE!!
    }
    // Constructor privat
    private  fun buidRoomDB(context: Context) =
        Room.databaseBuilder(
            context.applicationContext,
            IncidenciasDB::class.java,
            name: "incidencias.db"
        ).build()
}
```

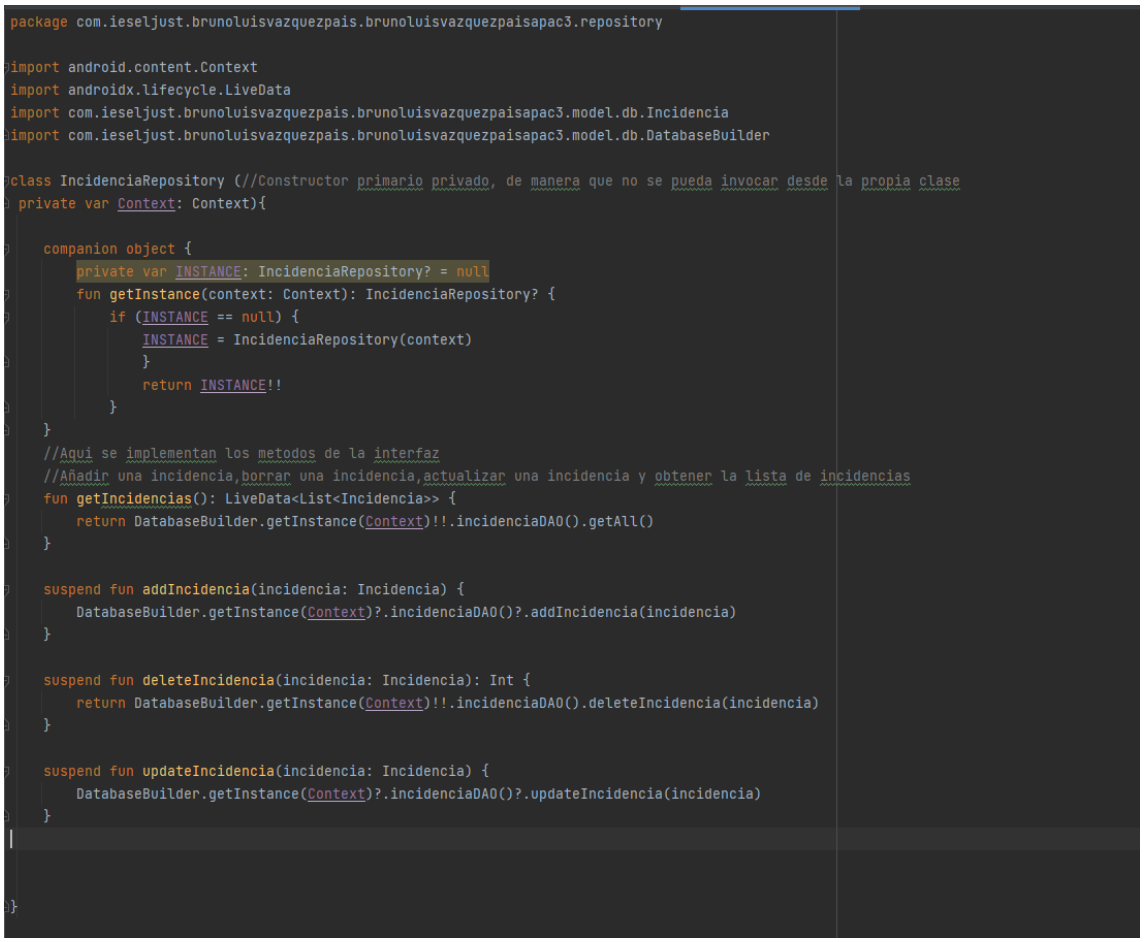
• Crearemos una clase que derive de RoomDatabase, y que definirá las entidades y la versión, así

como un método abstracto para obtener la clase DAO, llamado IncidenciaDAO

• Creará el objeto DataBaseBuilder, que nos devuelva una instancia única de esta clase siguiendo el patrón Singleton getInstance. Para ello hará uso de un constructor privado que sea quien invoque al método buildRoomDB

correspondiente de Room.

Implementación del Repositorio



```
package com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3.repository

import android.content.Context
import androidx.lifecycle.LiveData
import com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3.model.db.Incidencia
import com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3.model.db.DatabaseBuilder

class IncidenciaRepository (//Constructor primario privado, de manera que no se pueda invocar desde la propia clase
 private var Context: Context){

    companion object {
        private var INSTANCE: IncidenciaRepository? = null
        fun getInstance(context: Context): IncidenciaRepository? {
            if (INSTANCE == null) {
                INSTANCE = IncidenciaRepository(context)
                }
                return INSTANCE!!
            }
    }
    //Aqui se implementan los metodos de la interfaz
    //Añadir una incidencia,borrar una incidencia,actualizar una incidencia y obtener la lista de incidencias
    fun getIncidencias(): LiveData<List<Incidencia>> {
        return DatabaseBuilder.getInstance(Context)!!.incidenciaDAO().getAll()
    }

    suspend fun addIncidencia(incidencia: Incidencia) {
        DatabaseBuilder.getInstance(Context)?.incidenciaDAO()?.addIncidencia(incidencia)
    }

    suspend fun deleteIncidencia(incidencia: Incidencia): Int {
        return DatabaseBuilder.getInstance(Context)!!.incidenciaDAO().deleteIncidencia(incidencia)
    }

    suspend fun updateIncidencia(incidencia: Incidencia) {
        DatabaseBuilder.getInstance(Context)?.incidenciaDAO()?.updateIncidencia(incidencia)
    }


}
```

Implementamos los métodos de la interfaz

Añadir,borrar,actualizar, obtener la lista de incidencias

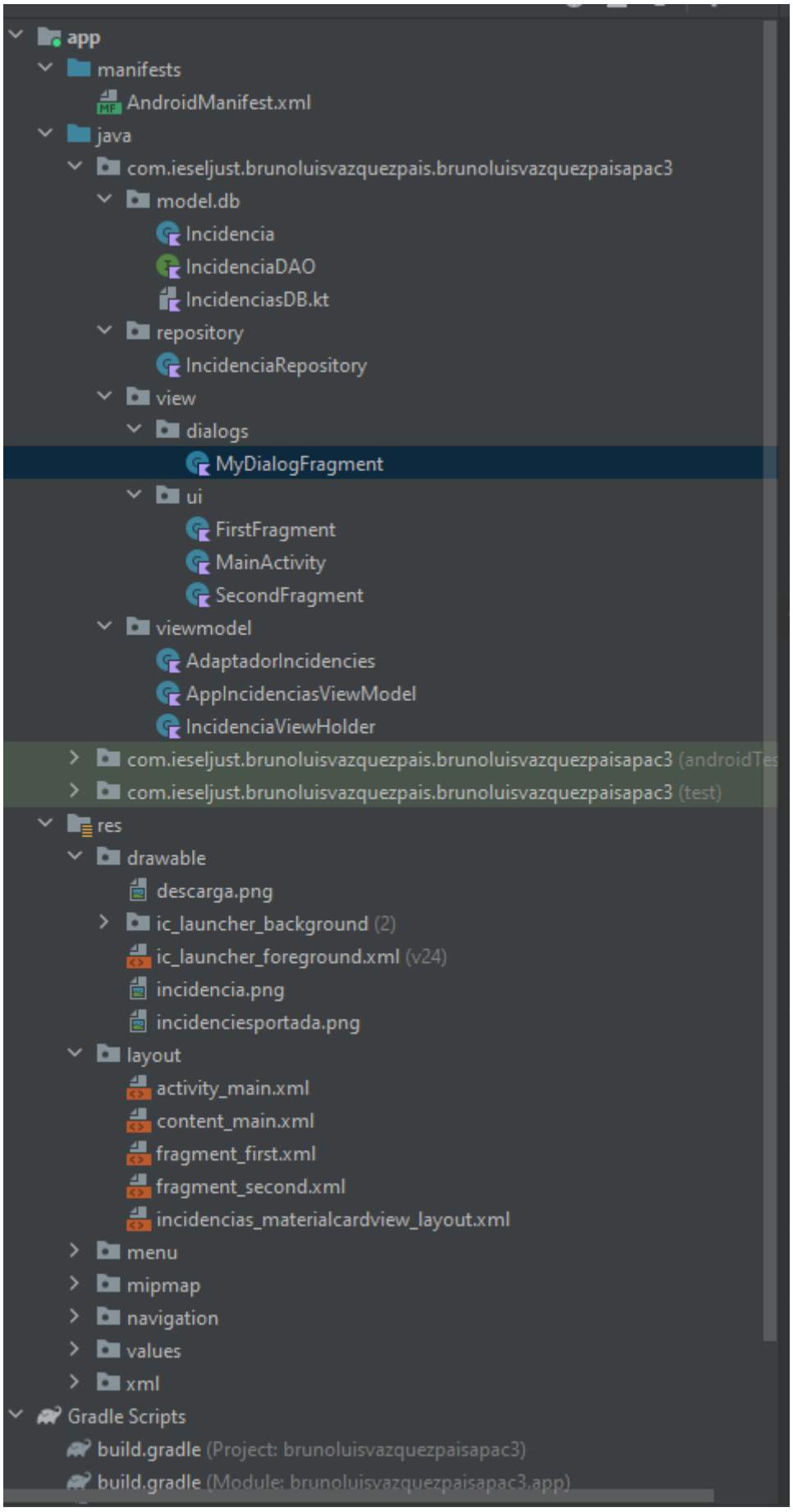
Implementacion ViewModel y las vistas

```kotlin
package com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3.view.dialogs

import android.app.Dialog
import android.content.Context
import android.os.Bundle
import androidx.appcompat.app.AlertDialog
import androidx.fragment.app.DialogFragment

class MyDialogFragment(): DialogFragment() {
    private var title="Títol per defecte"
    private var content="Contingut per defecte"

    // Definim el listner (qui invoca el diàleg) com a
    // una propietat més, amb lateinit.
    private lateinit var listener: OkOrCancelDialogable

    constructor (title:String, content:String) : this() {
        this.title=title
        this.content=content

    }

    // Definim la interfície que hauran d'implementar
    // els fragments que vulguen utilitzar el diàleg.
    // Aquesta defineix dos mètodes per a les classes que la
    // implementen, i que es corresponen a les accions del
    // Clic en OK o en Cancel.
    interface OkOrCancelDialogable {
        fun onPositiveClick()
        fun onCancelClick()
    }


    // onAttach: Mètode del cicle de vida que es dispara
    // quan el fragment s'adjunta a l'activitat.
    // En aquest moment, inicialitzem el listener.

    override fun onAttach(context: Context) {
        super.onAttach(context)
        // Verifiquem que qui ens invoca implementa la interfície
        // per fer possibles els callbacks
```

```kotlin
    override fun onAttach(context: Context) {
        super.onAttach(context)
        // Verifiquem que qui ens invoca implementa la interfície
        // per fer possibles els callbacks
        try {
            // Instanciem el diàleg per poder invocar els callbacks
            // parentFragment és el fragment pare (NavHostFragment)
            // des del parent Fragment accedim als fragments del fill
            // i obtenim el que està en la posició 0.
            // Amb això obtenim el fragment que es troba actualment
            // en el host de navegació, que no és altre que FirstFragment.
            listener = parentFragment?.childFragmentManager?.fragments?.get(0) as OkOrCancelDialogable

        } catch (e: ClassCastException) {
            // Si el fragment que ens invoca no implementa la interfície
            // llancem una excepció.
            throw ClassCastException(
                (listener.toString() +
                        " must implement OkOrCancelDialogable")
            )
        }
    }


    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {

        if(savedInstanceState!=null) {
            title=savedInstanceState.getString( key: "title")?:title
            content=savedInstanceState.getString( key: "content")?:content
        }

        return activity?.let {  it: FragmentActivity
            val builder: AlertDialog.Builder=AlertDialog
                .Builder(requireActivity())
            // Establim el títol i el contingut del diàleg
            builder.setTitle(title).setMessage(content)
                .setPositiveButton(android.R.string.ok) { _, _ ->
                    // Invoquem al callback per al click en acceptar
                    // del listener.
                    listener.onPositiveClick()
```

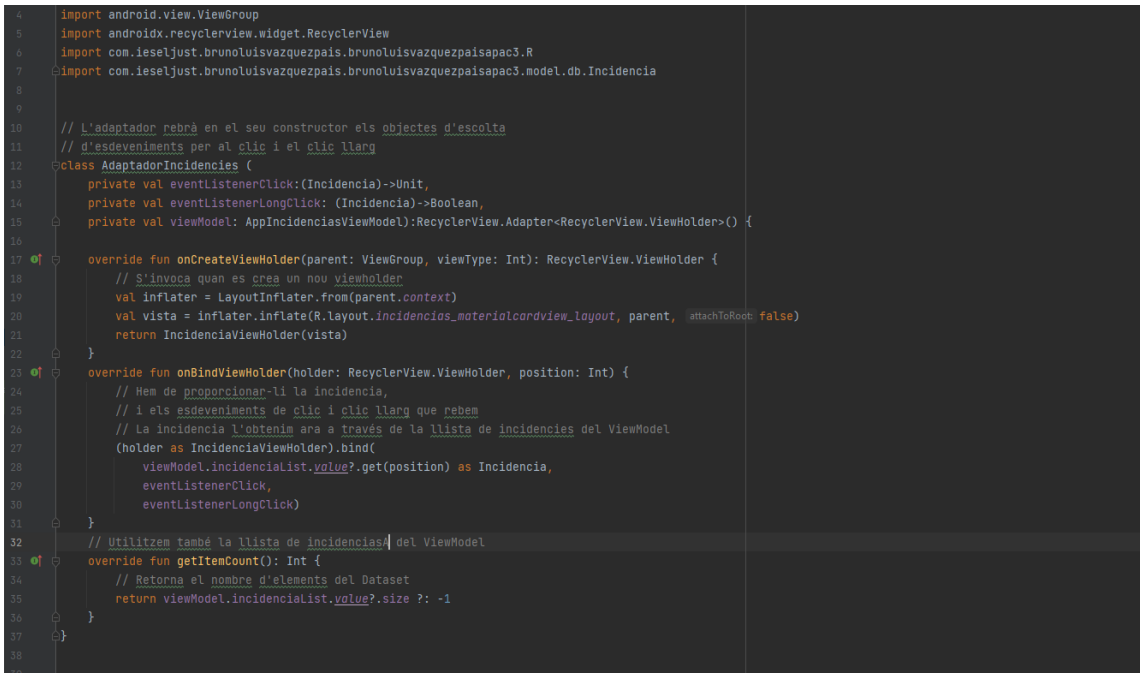
```
override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {

    if(savedInstanceState!=null) {
        title=savedInstanceState.getString( key: "title")?:title
        content=savedInstanceState.getString( key: "content")?:content
    }

    return activity?.let {  it: FragmentActivity
        val builder: AlertDialog.Builder=AlertDialog
            .Builder(requireActivity())
        // Establim el títol i el contingut del diàleg
        builder.setTitle(title).setMessage(content)
            .setPositiveButton(android.R.string.ok) { _, _ ->
                // Invoquem al callback per al click en acceptar
                // del listener.
                listener.onPositiveClick()
            }
            .setNegativeButton(android.R.string.cancel) { _, _ ->
                // Invoquem al callback per al click en cancel·lar
                // del listener.
                listener.onCancelClick()
            }
        // Retornen l'AlertDialog,
        return builder.create()} ?: throw IllegalStateException("El fragment no està associat a cap fragment")
}


override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)
    outState.putString("title", title)
    outState.putString("content", content)
}
}
```

Tendremos que crear un diolog para cuando creamos un nuevo usuario y el proyecto tendrá la siguiente distribución:


app
manifests
AndroidManifest.xml
java
com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3
model.db
Incidencia
IncidenciaDAO
IncidenciasDB.kt
repository
IncidenciaRepository
view
dialogs
MyDialogFragment
ui
FirstFragment
MainActivity
SecondFragment
viewmodel
AdaptadorIncidencies
AppIncidenciasViewModel
IncidenciaViewHolder
com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3 (androidTes
com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3 (test)
res
drawable
descarga.png
ic_launcher_background (2)
ic_launcher_foreground.xml (v24)
incidencia.png
incidenciesportada.png
layout
activity_main.xml
content_main.xml
fragment_first.xml
fragment_second.xml
incidencias_materialcardview_layout.xml
menu
mipmap
navigation
values
xml
Gradle Scripts
build.gradle (Project: brunoluisvazquezpaisapac3)
build.gradle (Module: brunoluisvazquezpaisapac3.app)

En el viewModerl crearemos tres clase AdaptadorIncidencias, AppIncidenciaViewModel y IncidenciaViewHolder.

AdaptadorIncidencias



```
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3.R
import com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3.model.db.Incidencia


// L'adaptador rebrà en el seu constructor els objectes d'escolta
// d'esdeveniments per al clic i el clic llarg
class AdaptadorIncidencies (
    private val eventListenerClick:(Incidencia)->Unit,
    private val eventListenerLongClick: (Incidencia)->Boolean,
    private val viewModel: AppIncidenciasViewModel):RecyclerView.Adapter<RecyclerView.ViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
        // S'invoca quan es crea un nou viewholder
        val inflater = LayoutInflater.from(parent.context)
        val vista = inflater.inflate(R.layout.incidencias_materialcardview_layout, parent, attachToRoot: false)
        return IncidenciaViewHolder(vista)
    }
    override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {
        // Hem de proporcionar-li la incidencia,
        // i els esdeveniments de clic i clic llarg que rebem
        // La incidencia l'obtenim ara a través de la llista de incidencies del ViewModel
        (holder as IncidenciaViewHolder).bind(
            viewModel.incidenciaList.value?.get(position) as Incidencia,
            eventListenerClick,
            eventListenerLongClick)
    }
    // Utilitzem també la llista de incidenciasA del ViewModel
    override fun getItemCount(): Int {
        // Retorna el nombre d'elements del Dataset
        return viewModel.incidenciaList.value?.size ?: -1
    }
}
```

AppIncidenciaViewModel

Se comunicara con el adaptador y lo live data para mostrar, modificar, borrar los datos del recycler view

```kotlin
package com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3.viewmodel

import android.app.Application
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.viewModelScope
import com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3.model.db.Incidencia
import com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3.repository.IncidenciaRepository
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

// Definim el ViewModel com a AndroidViewModel, per tal de poder accedir al context
class AppIncidenciasViewModel(application: Application): AndroidViewModel(application) {

    // Definición d'atributs

    // Referència al repositori
    val repository = IncidenciaRepository.getInstance(application.applicationContext)

    // Incidencias que s'està editant actualment
    // Observeu que el Incidencieas pot ser nul

    private val _incidenciaActual=MutableLiveData<Incidencia?>() // Atribut de suport privat
    val incidenciaActual: LiveData<Incidencia?> = _incidenciaActual // Accés públic

    // Setter
    fun setIncidenciaActual(incidencia: Incidencia){
        _incidenciaActual.value=incidencia.copy()
    }

    // Setter a null
    fun cleanIncidenciaActual(){
        _incidenciaActual.value = null
    }



    // Livedata per a la llista de incidencias
    val incidenciaList: LiveData<List<Incidencia>> by lazy {
        repository!!.getIncidencias()
```

IncidenciaViewHolder

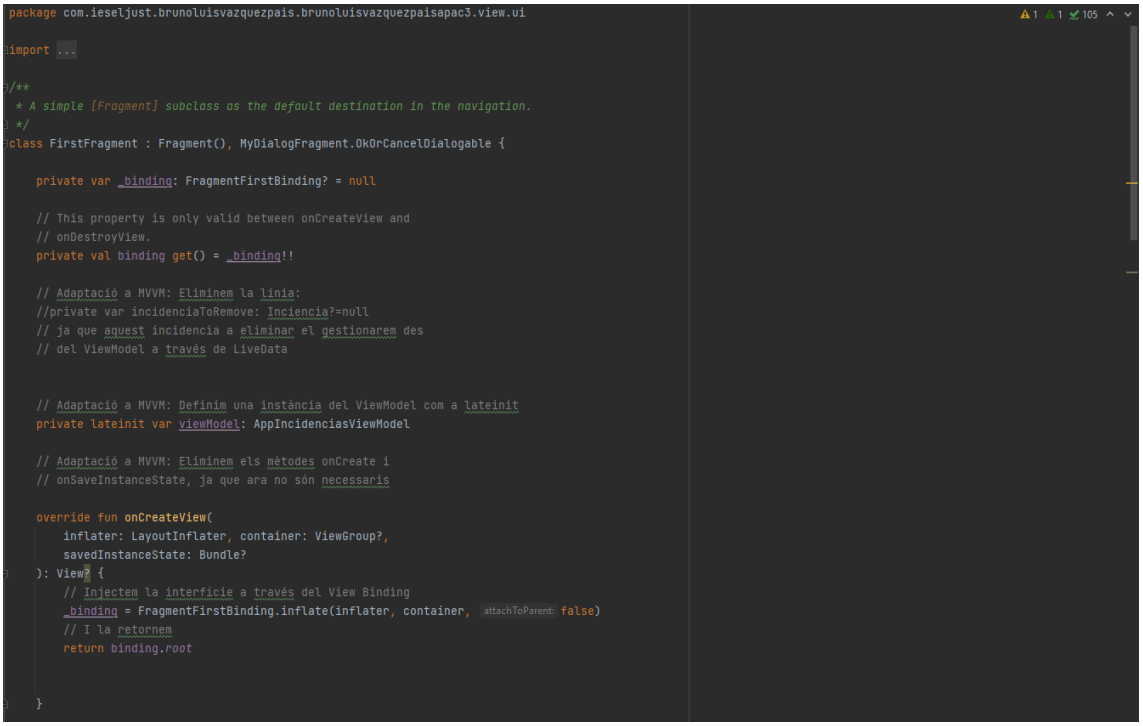
```kotlin
package com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3.viewmodel

import android.view.View
import android.widget.ImageView
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3.R
import com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3.model.db.Incidencia

class IncidenciaViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView){

    val img = itemView.findViewById(R.id.imageView2) as ImageView
    val asumpte = itemView.findViewById(R.id.textView) as TextView
    val descripcio = itemView.findViewById(R.id.textView2) as TextView

    // Enllacem les dades de la incidencia amb la vista
    // i amb els seus gestors d'esdeveniments, com a funcions Lambda
    fun bind(
        incidencia: Incidencia,
        eventListenerClick: (Incidencia) -> Unit,
        eventListenerLongClick: (Incidencia) -> Boolean
    ) {
        asumpte.text = incidencia.assumpte
        descripcio.text = incidencia.descripcio
        img.setImageResource(incidencia.img)

        /* Ara capturem els esdeveniments i invoquem el callback corresponent */

        // Click normal
        itemView.setOnClickListener{ it: View!
            eventListenerClick(incidencia)
            }

        // Click llarg
        itemView.setOnLongClickListener{ it: View!
            eventListenerLongClick(incidencia)
        }
    }
```

En las Clases FirstFragment, MainActivity, SecondFragment

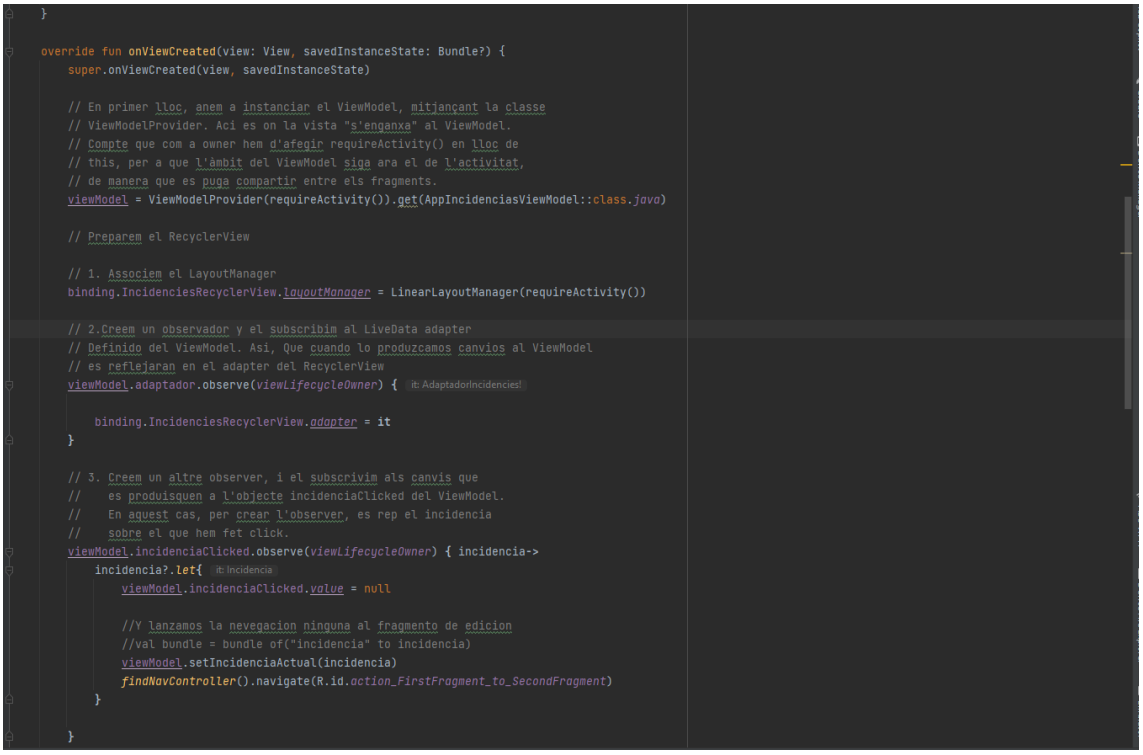
Lo adaptaremos el mvvm los métodos item y itemlongclicked,

Llamaremos a las demás clases del proyecto

FIRSTFRAGMENT



```
package com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3.view.ui

import ...

/**
 * A simple [Fragment] subclass as the default destination in the navigation.
 */
class FirstFragment : Fragment(), MyDialogFragment.OkOrCancelDialogable {

    private var _binding: FragmentFirstBinding? = null

    // This property is only valid between onCreateView and
    // onDestroyView.
    private val binding get() = _binding!!

    // Adaptació a MVVM: Eliminem la línia:
    //private var incidenciaToRemove: Inciencia?=null
    // ja que aquest incidencia a eliminar el gestionarem des
    // del ViewModel a través de LiveData


    // Adaptació a MVVM: Definim una instància del ViewModel com a lateinit
    private lateinit var viewModel: AppIncidenciasViewModel

    // Adaptació a MVVM: Eliminem els mètodes onCreate i
    // onSaveInstanceState, ja que ara no són necessaris

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Injectem la interfície a través del View Binding
        _binding = FragmentFirstBinding.inflate(inflater, container,  attachToParent: false)
        // I la retornem
        return binding.root


    }
```



```
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        // En primer lloc, anem a instanciar el ViewModel, mitjançant la classe
        // ViewModelProvider. Ací es on la vista "s'enganxa" al ViewModel.
        // Compte que com a owner hem d'afegir requireActivity() en lloc de
        // this, per a que l'àmbit del ViewModel siga ara el de l'activitat,
        // de manera que es puga compartir entre els fragments.
        viewModel = ViewModelProvider(requireActivity()).get(AppIncidenciasViewModel::class.java)

        // Preparem el RecyclerView

        // 1. Associem el LayoutManager
        binding.IncidenciesRecyclerView.layoutManager = LinearLayoutManager(requireActivity())

        // 2.Creem un observador y el subscribim al LiveData adapter
        // Definido del ViewModel. Así, Que cuando lo produzcamos canvios al ViewModel
        // es reflejaran en el adapter del RecyclerView
        viewModel.adaptador.observe(viewLifecycleOwner) {  it: AdaptadorIncidencies!

            binding.IncidenciesRecyclerView.adapter = it
        }

        // 3. Creem un altre observer, i el subscrivim als canvis que
        //    es produisquen a l'objecte incidenciaClicked del ViewModel.
        //    En aquest cas, per crear l'observer, es rep el incidencia
        //    sobre el que hem fet click.
        viewModel.incidenciaClicked.observe(viewLifecycleOwner) { incidencia->
            incidencia?.let{  it: Incidencia
                viewModel.incidenciaClicked.value = null

                //Y lanzamos la nevegacion ninguna al fragmento de edicion
                //val bundle = bundle of("incidencia" to incidencia)
                viewModel.setIncidenciaActual(incidencia)
                findNavController().navigate(R.id.action_FirstFragment_to_SecondFragment)
            }

        }
```

MainActivity

```kotlin
package com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3.view.ui

import ...

class MainActivity : AppCompatActivity() {

    private lateinit var appBarConfiguration: AppBarConfiguration
    private lateinit var binding: ActivityMainBinding
    //Adaptació a MVVM: Referència al ViewModel
    private lateinit var viewModel: AppIncidenciasViewModel
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        // Adaptació a MVVM: Instanciem el ViewModel, utilitzant la classe ViewModelProvider
        viewModel = ViewModelProvider( owner: this)[AppIncidenciasViewModel::class.java]

        setSupportActionBar(binding.toolbar)

        val navController = findNavController(R.id.nav_host_fragment_content_main)
        appBarConfiguration = AppBarConfiguration(navController.graph)
        setupActionBarWithNavController(navController, appBarConfiguration)
        //Modificació per anar al segon fragment
        binding.fab.setOnClickListener { it: View!
            viewModel.cleanIncidenciaActual()
            navController.navigate(R.id.action_FirstFragment_to_SecondFragment)
        }
        navController.addOnDestinationChangedListener { _, destination, _ ->
            when ((destination as FragmentNavigator.Destination).className) {
                // show fab in recipe fragment
                FirstFragment::class.qualifiedName -> {
                    binding.fab.visibility = View.VISIBLE
                }
                else -> {
                    binding.fab.visibility = View.GONE
                }
            }
        }
    }
```

```kotlin
        }
        navController.addOnDestinationChangedListener { _, destination, _ ->
            when ((destination as FragmentNavigator.Destination).className) {
                // show fab in recipe fragment
                FirstFragment::class.qualifiedName -> {
                    binding.fab.visibility = View.VISIBLE
                }
                else -> {
                    binding.fab.visibility = View.GONE
                }
            }
        }
    }

    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        // Inflate the menu; this adds items to the action bar if it is present.
        menuInflater.inflate(R.menu.menu_main, menu)
        return true
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        return when (item.itemId) {
            R.id.action_settings -> true
            else -> super.onOptionsItemSelected(item)
        }
    }

    override fun onSupportNavigateUp(): Boolean {
        val navController = findNavController(R.id.nav_host_fragment_content_main)
        return navController.navigateUp(appBarConfiguration)
                || super.onSupportNavigateUp()
    }
}
```

SecondFragment

```kotlin
package com.ieseljust.brunoluisvazquezpais.brunoluisvazquezpaisapac3.view.ui

import ...

class SecondFragment : Fragment() {

    private var _binding: FragmentSecondBinding? = null
    private val binding get() = _binding!!

    // Adaptació a MVVM: Referència al ViewModel
    private lateinit var viewModel: AppIncidenciasViewModel

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentSecondBinding.inflate(inflater, container, attachToParent: false)

        // La funcionalitat per inicialitzar la vista l'hem afegir el mètode onViewCreated

        // Finalment retornem l'arrel del binding

        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        // Adaptació a MVVM: Instanciem el ViewModel, utilitzant la classe ViewModelProvider
        viewModel = ViewModelProvider(requireActivity())[AppIncidenciasViewModel::class.java]

        // Adaptació a MVVM: Afegim observadors al LiveData
        // incidenciaActual per a ser notificats dels canvis

        binding.imageViewIncidencia.setImageResource(R.drawable.incidencia)
        viewModel.incidenciaActual.observe(viewLifecycleOwner) {  it: Incidencia?
            // I quan aquests es produïsquen, actualitzem les vistes de la interfície
            it?.let {  it: Incidencia
                // Actualitzem les vistes de la interfície
                // I preparem els TextViews  per actualitzar el ViewModel quan canvien de valor
                binding.imageViewIncidencia.setImageResource(it.img)
```

```kotlin
                // I preparem els TextViews  per actualitzar el ViewModel quan canvien de valor
                binding.imageViewIncidencia.setImageResource(it.img)

                // TextView per al assumpte
                binding.editTextAssumpteIncidencia.setText(it.assumpte)
                binding.editTextAssumpteIncidencia.doOnTextChanged { text, _, _, _ ->
                    viewModel.incidenciaActual.value?.assumpte = text.toString()
                }

                // TextView per al Descripció
                binding.editDescpIncidencia.setText(it.descripcio)
                binding.editDescpIncidencia.doOnTextChanged { text, _, _, _ ->
                    viewModel.incidenciaActual.value?.descripcio=text.toString()
                }


                // TextView per al Ubicació
                binding.editubincidencia.setText(it.ubicacio)
                binding.editubincidencia.doOnTextChanged { text, _, _, _ ->
                    viewModel.incidenciaActual.value?.ubicacio=text.toString()
                }

                // Switch pera si esta resolt
                binding.switch1.isChecked = it.resolt
                binding.switch1.setOnCheckedChangeListener { _, isChecked ->
                    viewModel.incidenciaActual.value?.resolt = isChecked
                }



                // Actualització de l'spinner
                // Recorrem els diferents valors d'aquest, i comparem
                // amb el valor guardat. Si coincideix, deixem seleccionat el valor.

                for (i in 0 ≤ until < binding.spinnerServei.adapter.count)
                    if (binding.spinnerServei.adapter.getItem(i).equals(it.resolt)) {
                        binding.spinnerServei.setSelection(i)
                        break
                    }
            }
        }
```

```
        // Observadors per saber quan s'ha guardat o modificat un contacte

        viewModel.incidenciaSaved.observe(viewLifecycleOwner) { saved ->
            saved?.let {  it: Boolean
                // Netegem el valor per a no entrar en un bucle infinit
                viewModel.incidenciaSaved.value=null
                Snackbar.make(
                    binding.root,
                    "Incidencia has been saved",
                    Snackbar.LENGTH_LONG
                ).setAction(resources.getString(android.R.string.ok)) {  it: View!
                    findNavController().navigateUp()
                }.show()
            }
        }


        viewModel.incidenciaUpdated.observe(viewLifecycleOwner) { updated ->
            updated?.let {  it: Boolean
                // Netegem el valor per a no entrar en un bucle infinit
                viewModel.incidenciaUpdated.value=null
                Snackbar.make(
                    binding.root,
                    "Contact has been updated",
                    Snackbar.LENGTH_LONG
                ).setAction(resources.getString(android.R.string.ok)) {  it: View!
                    findNavController().navigateUp()
                }.show()
            }
        }

        // Capturem el click sobre el botó de guardar:
        binding.buttonSave.setOnClickListener {  it: View!
            guardaIncidencia()
        }

    }
    override fun onDestroyView() {
```

```
    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }


    private fun guardaIncidencia(){

        // Creem un nou contacte a partir de la informació de les vistes
        // Recordeu que de moment agafem la imatge actual, que ara s'ubica
        // al contacte emmagatzemat al ViewModel

        // Utilitzem el constructor mitjançant arguments per nom en lloc de posicionals
        val nou= Incidencia(
            assumpte = binding.editTextAssumpteIncidencia.text.toString(),
            descripcio = binding.editDescpIncidencia.text.toString(),
            ubicacio = binding.editubincidencia.text.toString(),
            servei = binding.spinnerServei.selectedItem.toString(),
            resolt = binding.switch1.isChecked,
            img = viewModel.incidenciaActual.value?.img?:R.drawable.incidencia //La imagen sera la misma
            )

        viewModel.guardaIncidencia(nou)
    }
}
```