SpyLook 👀

SpyLook es una aplicación Android desarrollada en `Kotlin` y utilizando el sistema de Views en `XML` para apuntar información sobre las personas de tu entorno. Desde sus datos personales hasta sus amistades, grupos, cuentas de *RRSS*, etc

Instalación

Puedes descargar cualquier versión de la aplicación directamente en formato APK desde la web oficial: https://cdominguezh06.github.io/spylook-web/

Estructura de la aplicación

```
spylook/
 ├── adapters/        # Adaptadores para listas y componentes de UI
(RecyclerView, etc.)
 ├── controller/      # Controladores para acceso a APIs REST de terceros
 ├── dao/             # Objetos de acceso a datos (DAOs) para la base de
datos local de Android Room
 ├── database/        # Definición y configuración de la base de datos de
Android Room
 ├── mappers/         # Conversores entre modelos de la base de datos y
'tarjetas' (elementos a mostrar en listas de RecyclerView)
 ├── model/           # Modelos de datos: entidades como Contacto, Grupo,
Suceso, etc.
 └── view/            # Activities, fragments y vistas principales
organizadas en subcarpetas
      ├── accounts/   # Vistas relativas al CRUD de cuentas
      ├── common/     # Vistas y fragments comunes o reutilizables
      ├── contacts/   # Vistas relativas al CRUD de contactos
      ├── groups/     # Vistas relativas al CRUD de grupos
      └── sucesos/    # Vistas relativas al CRUD de sucesos
```

Descripción de carpetas clave

adapters/

```
  adapters/
├── cards/
│    ├── AmigoCardAdapter.kt
│    ├── AnotacionCardAdapter.kt
│    ├── ContactoCardAdapter.kt
│    ├── ContactoMiniCardAdapter.kt
│    ├── CuentaCardAdapter.kt
│    ├── GrupoCardAdapter.kt
│    ├── GruposDeContactoCardAdapter.kt
│    └── SucesoCardAdapter.kt
```

```
├── search/
│    ├── BusquedaContactoCardAdapter.kt
│    ├── BusquedaGrupoCardAdapter.kt
│    ├── MultipleContactsCardSearchAdapter.kt
│    └── SingleContactCardSearchAdapter.kt
└── slider/
     ├── ContactSliderCardAdapter.kt
     ├── CuentaSliderCardAdapter.kt
     ├── GroupSliderCardAdapter.kt
     └── SucesoSliderCardAdapter.kt
```

Este paquete, dividido en tres paquetes, contiene todos los adaptadores de RecyclerView de la aplicación, cada uno para un caso de uso concreto donde se necesitan listar elementos distintos, con vistas distintas y comportamientos distintos

A su vez también almacena adaptadores para el comportamiento de los menús de los Fragments

- **cards/** Estos adapters se encargan de mostrar todas las tarjetas de la aplicación. Su función es redirigir al usuario a la activity del modelo en cuestión que maneje el adapter para visualizar los datos o mantener pulsada la tarjeta para acceder a información adicional
- **search/** Estos adapters muestran las tarjetas en un contexto de búsqueda (cuando el usuario utiliza el buscador para filtrar rápidamente elementos) y le asignan un comportamiento diferente al dibujarlas en pantalla y al interactuar con ellas
- **slider** Estos adaptadores se aplican a un ViewPager para asignar el comportamiento del TabLayout de una activity, definiendo sus pestañas y el fragmento que se inflará en cada una

controller/

```
controller/
└── GithubController.kt
```

Controlador para la integración con GitHub (gestión de releases, actualizaciones, etc).

dao/

```
dao/
├── AnotacionDAO.kt
├── ContactoDAO.kt
├── CuentaDao.kt
├── GrupoDAO.kt
└── SucesoDAO.kt
```

DAOs para todas las entidades principales: Anotación, Contacto, Cuenta, Grupo, Suceso.

El acceso a las operaciones de los DAOs se debe realizar de forma asíncrona, evitando así bloquear el hilo principal encargado de la UI

La elección de Kotlin como lenguaje de programación permite realizar esto de forma muy sencilla mediante el uso del patrón de diseño asíncrono de las **Corrutinas**

Las corrutinas son muy similares en concepto a los **hilos virtuales** de Java. Este concepto consiste en separar la equivalencia de un hilo de la máquina virtual de Java con un hilo del procesador, almacenandose en el heap en lugar de en el stack y permitiendo cambios de contexto con un coste mínimo de rendimiento (cuando se pausa la ejecución de una corrutina el hilo físico no se detiene, sino que se le asigna otra tarea)

En SpyLook se hace uso principalmente de `lifecycleScope.launch` para lanzar corrutinas y acceder a la base de datos. De esta forma se genera una corrutina ligada al ciclo de vida de la Activity o Fragment donde se haya creado (si la Activity se cierra la corrutina también) y permite acceder a la base de datos sin bloquear el hilo principal

```
    lifecycleScope.launch {
        val contact = contactoDAO.findContactoById(intent.getIntExtra("id",
0))
        setupContactDetails(contact)
        setupViewPager(contact)
    }
```

Sin embargo, en contextos ajenos a una Activity, como puede ser un adapter de RecyclerView, no se puede hacer uso de `lifecycleScope.launch`, por lo que se recurre al método `runBlocking` que genera una corrutina bloqueante, aunque todavía no ha demostrado ser perjudicial para el rendimiento de la app

```
    runBlocking {
        dao.deleteContactoWithAnotableById(cardItem.idAnotable)
        val index = cardItemList.indexOf(cardItem)
        recyclerAnimator.deleteItemWithAnimation(
            holder.itemView,
            index,
            onEmptyCallback = {
                cardItemList.add(ContactoCardItem.DEFAULT_FOR_EMPTY_LIST)
            },
            afterDeleteCallBack = {
                popupWindow.dismiss()
            })
    }
```

database/

```
database/
└── AppDatabase.kt
```

Definición de la base de datos Room. Esta clase cuenta con un patrón *Singleton* para acceder de forma global a la misma instancia de la base de datos. En ella nos encontramos una instancia de cada DAO declarado, la

declaración de todas las entidades que serán convertidas en tablas y una versión de la base de datos. Esta versión ha de cambiarse cada vez que modificamos el esquema de la base de datos (modificando entidades) para que Room rehaga la base de datos y no acceda a un esquema antiguo, provocando un fallo en el proceso

mappers/

```
mappers/
├── AnotacionToCardItem.kt
├── ContactoToCardItem.kt
├── ContactoToMiniCard.kt
├── CuentaToCardItem.kt
├── GrupoToCardItem.kt
└── SucesoToCardItem.kt
```

Mappers creados con *MapStruct* para convertir fácilmente entidades en tarjetas para mostrar en un RecyclerView

model/

```
model/
├── cards/
│   ├── AnotacionCardItem.kt
│   ├── ContactoCardItem.kt
│   ├── ContactoMiniCard.kt
│   ├── CuentaCardItem.kt
│   ├── GrupoCardItem.kt
│   └── SucesoCardItem.kt
├── entity/
│   ├── Anotable.kt
│   ├── Anotacion.kt
│   ├── Contacto.kt
│   ├── ContactoAmistadCrossRef.kt
│   ├── ContactoGrupoCrossRef.kt
│   ├── ContactoSucesoCrossRef.kt
│   ├── Cuenta.kt
│   ├── CuentaContactoCrossRef.kt
│   ├── Grupo.kt
│   └── Suceso.kt
├── github/
│   ├── GitHubAPI.kt
│   ├── GitHubRelease.kt
│   └── ReleaseAsset.kt
├── relations/
│   ├── AmigosDeContacto.kt
│   ├── AnotableConAnotaciones.kt
│   ├── CausanteSuceso.kt
│   ├── ContactoConCuentas.kt
│   ├── ContactosGrupos.kt
```

```
│     ├── ContactosSucesos.kt
│     ├── CreadorGrupo.kt
│     ├── CuentaConContactos.kt
│     ├── GruposContactos.kt
│     └── SucesosContactos.kt
└── utils/
      ├── animations/
      ├── converters/
      ├── decorators/
      ├── textWatchers/
      ├── ApplicationUpdater.kt
      ├── ForegroundShaderSpan.kt
      └── StringWithSpacesIndexRetriever.kt
```

En este paquete se encuentran la mayoría de clases de la aplicación

- **cards/**: Clases para cada tipo de tarjeta que se pueda mostrar en un RecyclerView
- **entity/**: Entidades Room, relaciones entre entidades y crossrefs para relaciones N:M.
- **github/**: Modelos para integración con la API de GitHub y tomar información de cada release de la aplicación
- **relations/**: Clases para relaciones entre entidades.
- **utils/**: Utilidades generales (actualización, decoradores, animaciones, etc), con subcarpetas especializadas.

  ```
  utils/
     ├── animations/
     │    └── RecyclerViewAnimator.kt
     ├── converters/
     │    └── DateConverters.kt
     ├── decorators/
     │    ├── RainbowTextViewDecorator.kt
     │    └── SpacingItemDecoration.kt
     ├── textWatchers/
     │    ├── actions/
     │    │    └── LongTextScrollerAction.kt
     │    ├── DateTextWatcher.kt
     │    ├── TextWatcherSearchBarContacts.kt
     │    ├── TextWatcherSearchBarGroups.kt
     │    ├── TextWatcherSearchBarGruposDeContacto.kt
     │    └── TextWatcherSearchBarMiembros.kt
     ├── ApplicationUpdater.kt
     ├── ForegroundShaderSpan.kt
     └── StringWithSpacesIndexRetriever
  ```

  - **animations/RecyclerViewAnimator.kt**: Animación al eliminar un elemento de un RecyclerView, aceptando funciones anónimas (lambda) como parámetro para definir el comportamiento después del borrado y cuando la lista queda vacía tras el borrado

    ```
    ```

```
    // Eliminar del dataSource inmediatamente
    if (dataSource.size == 1) {
        onEmptyCallback()
    }
    dataSource.removeAt(position)

    // Actualizar el adaptador inmediatamente
    adapter.notifyItemRemoved(position)

    // Crear y configurar la animación
    val animation = AnimationUtils.loadAnimation(rowView.context,
android.R.anim.slide_out_right).apply {
        duration = 300
    }
    rowView.startAnimation(animation)
    // Esperar a que termine la animación antes de interactuar con
el RecyclerView
    Handler(Looper.getMainLooper()).postDelayed({
afterDeleteCallBack() }, animation.duration)
```

- **converters/DateConverters.kt**: Singleton para generar conversores de fechas LocalDate o LocalDateTime a String, siguiendo un regex específico

```
    private val dateFormatter: DateTimeFormatter =
DateTimeFormatter.ISO_LOCAL_DATE
    private val dateTimeFormatter: DateTimeFormatter =
DateTimeFormatter.ofPattern("dd/MM/yyyy | HH:mm:ss")

    @JvmStatic
    @TypeConverter
    fun fromString(value: String): LocalDate {
        return LocalDate.parse(value, dateFormatter)
    }

    @JvmStatic
    @TypeConverter
    fun toString(date: LocalDate): String {
        return date.format(dateFormatter)
    }

    fun toCustomString(date: LocalDate, dateFormatter:
DateTimeFormatter): String {
        return date.format(dateFormatter)
    }

    @JvmStatic
    @TypeConverter
    fun toDateTimeString(dateTime: LocalDateTime): String {
        return dateTime.format(dateTimeFormatter)
    }
```

- **decorators/**: Objetos que alteran la apariencia de elementos de la UI:
  - **RainbowTextViewDecorator.kt**: Convierte el texto de un TextView para que sea de color arcoíris

```
private val drawableResourceId: Int =
R.drawable.rainbow_gradient

fun apply() {

textView.getViewTreeObserver().addOnGlobalLayoutListener(OnG
lobalLayoutListener {
        val gradientDrawable =
            AppCompatResources.getDrawable(context,
drawableResourceId) as GradientDrawable?

        var gradientColors = gradientDrawable!!.getColors()
        if (gradientColors == null) {
            gradientColors =
                intArrayOf(-0x10000, -0x100, -0xff0100,
-0xff0001, -0xffff01, -0xff01)
        }

        val linearGradient = LinearGradient(
            0f, 0f, textView.width.toFloat(),
textView.textSize,
            gradientColors,
            null,
            Shader.TileMode.CLAMP
        )

        textView.paint.setShader(linearGradient)
        textView.invalidate()
    })
}
```

  - **SpacingItemDecoration.kt**: Añade una pequeña separación entre elementos de un RecyclerView

```
class SpacingItemDecoration(private val context: Context) :
ItemDecoration() {
    override fun getItemOffsets(
        outRect: Rect,
        view: View,
        parent: RecyclerView,
        state: RecyclerView.State
    ) {
        val vertical =
context.getResources().getDimensionPixelSize(R.dimen.spacing
_vertical)
```

```
        val horizontal =
context.getResources().getDimensionPixelSize(R.dimen.spacing
_horizontal)
        outRect.set(horizontal, vertical, horizontal,
vertical)
    }
}
```

- **textWatchers/**: Aqui hay objetos que analizan constantemente los cambios de texto de un EditText con distintas finalizades
  - **actions/**: Acciones Runnable a aplicar por un TextWatcher sobre un elemento de la UI para alterar su estilo vía código Kotlin -**LongTextScrollerAction.kt**: Comprueba si el texto de la búsqueda no cabe en los límites de su respectivo TextView para desplazarlo hasta poder ver la coincidencia

```
override fun invoke() {
    val lineWidth =
         text.paint.measureText(text.text.toString())
    val viewWidth = text.width
    val endLine =
text.layout.getLineForOffset(lastScroll.toInt() + viewWidth)
    // Obtener el índice del último carácter visible en la
última línea visible
    val visibleText = text.layout.getOffsetForHorizontal(
        endLine.coerceAtMost(text.layout.lineCount - 1), //
Validar línea máxima
        lastScroll + viewWidth                      //
Última posición horizontal visible
    ).coerceAtMost(text.text.length)

    val endCharPosition =
visibleText.coerceAtMost(text.text.length)
    val startCharPosition = visibleText.coerceAtLeast(0)
    val actualLastCharPosition =
        text.paint.measureText(text.text.toString(), 0,
startIndex + busqueda.length - 1)
    if (lineWidth > viewWidth) {

        // Configuramos el TextView para manejar el
desplazamiento
        text.setHorizontallyScrolling(true)
        text.isHorizontalScrollBarEnabled = false
        text.isSingleLine = true
        text.ellipsize = null // Desactivar truncamiento
        text.textAlignment =
TextView.TEXT_ALIGNMENT_VIEW_START // Alineación desde el
inicio

        // Creando y asignando Scroller
        val scroller = Scroller(text.context)
        text.setScroller(scroller)
```

```
        if (actualLastCharPosition < startCharPosition ||
actualLastCharPosition > endCharPosition) {
            scroller.startScroll(lastScroll.toInt(), 0,
(actualLastCharPosition -lastScroll).toInt(), 0)
            lastScroll = text.x
        }
        text.invalidate()

    } else {
        text.isSelected = true
        text.isFocusable = true
        text.isFocusableInTouchMode = true
        text.ellipsize = TextUtils.TruncateAt.MARQUEE
        text.isHorizontalScrollBarEnabled = false
        text.setHorizontallyScrolling(false)
        text.scrollTo(0, 0)
    }
}
```

- **DateTextWatcher.kt**: Comprueba el valor introducido en el EditText asignado y delimita un valor mínimo (01/01/1970) y un valor máximo (LocalDate.now())

```
private var current = ""
private val ddmmyyyy = "DDMMYYYY"
private val cal: Calendar = Calendar.getInstance()

override fun beforeTextChanged(s: CharSequence?, start: Int,
count: Int, after: Int) {}

override fun onTextChanged(s: CharSequence, start: Int,
before: Int, count: Int) {
    if (s.toString() != current) {
        var clean = s.toString().replace("[^\\d]".toRegex(),
"")
        val cleanC = current.replace("[^\\d]".toRegex(), "")

        val cl = clean.length
        var sel = cl
        var i = 2
        while (i <= cl && i < 6) {
            sel++
            i += 2
        }
        if (clean == cleanC) sel--

        if (clean.length < 8) {
            clean = clean + ddmmyyyy.substring(clean.length)
        } else {
            var day = clean.substring(0, 2).toInt()
            var mon = clean.substring(2, 4).toInt()
```

```
            var year = clean.substring(4, 8).toInt()

            year = max(1970.0, min(year.toDouble(),
LocalDate.now().year.toDouble())).toInt()
            if (year == LocalDate.now().year) {
                mon = max(1.0, min(mon.toDouble(),
LocalDate.now().monthValue.toDouble())).toInt()
            } else {
                mon = max(1.0, min(mon.toDouble(),
12.0)).toInt()
            }
            cal.set(Calendar.YEAR, year)
            cal.set(Calendar.MONTH, mon - 1)
            if (mon == LocalDate.now().monthValue && year ==
LocalDate.now().year) {
                day = max(1.0, min(day.toDouble(),
LocalDate.now().dayOfMonth.toDouble())).toInt()
            }else{
                day = max(1.0, min(day.toDouble(),
cal.getActualMaximum(Calendar.DAY_OF_MONTH).toDouble())).toI
nt()
            }
            clean = String.format(Locale.getDefault(),
"%02d%02d%04d", day, mon, year)
        }
        clean = String.format(
            "%s/%s/%s", clean.substring(0, 2),
            clean.substring(2, 4),
            clean.substring(4, 8)
        )

        current = clean
        editText.setText(current)
        editText.setSelection(min(sel.toDouble(),
current.length.toDouble()).toInt())
    }
}
```

- **TextWatcherSearchBar%.kt**: Estos archivos asignan un comportamiento distinto a la barra de búsqueda según el resultado deseado

```
override fun onTextChanged(s: CharSequence?, start: Int,
before: Int, count: Int) {
   val busqueda = text.getText().toString().lowercase(
       Locale.getDefault()
   ).replace(" ", "")

   runBlocking {
       collect = db.contactoDAO()!!
           .getContactos()
           .map { c -> mapper.toCardItem(c) }
```

```
            .toMutableList()
    }
    busqueda.ifEmpty {

recyclerView!!.setAdapter(ContactoCardAdapter(collect,
context!!))
        retriever.contador = 0
        LongTextScrollerAction.lastScroll = 0.0f
        return@onTextChanged
    }
    collect = collect.filter { c ->
        c.alias.replace(" ",
"").lowercase(Locale.getDefault()).contains(busqueda)
    }.toMutableList()
    collect.ifEmpty {
        collect.add(ContactoCardItem.DEFAULT_FOR_NO_RESULTS)

recyclerView!!.setAdapter(ContactoCardAdapter(collect,
context!!))
        return@onTextChanged
    }
    val newAdapter = object : ContactoCardAdapter(collect,
context!!) {
        override fun onBindViewHolder(holder: CardViewHolder,
position: Int) {
            val cardItem = cardItemList[position]
            holder.name.text = cardItem.nombre
            holder.mostknownalias.text =
SpannableString(cardItem.alias).apply {
                cardItem.alias = cardItem.alias.let {
                    val spannable = SpannableString(it)
                    var startIndex =
retriever.getStartIndex(busqueda, cardItem.alias)
                    if (startIndex >= 0) {
                        val shader = LinearGradient(
                            0f, 0f,
holder.mostknownalias.textSize * 2, 0f,
                            intArrayOf(

context!!.getColor(R.color.red),

context.getColor(R.color.yellow),

context.getColor(R.color.green),

                                ),
                            floatArrayOf(0f, 0.5f, 1f),
                            Shader.TileMode.MIRROR
                        )

                        setSpan(
                            ForegroundShaderSpan(shader),
                            startIndex,
```

```
retriever.getSpanIntervalJump(busqueda, cardItem.alias,
startIndex),

Spannable.SPAN_EXCLUSIVE_EXCLUSIVE
                        )


holder.mostknownalias.post(LongTextScrollerAction(holder.mos
tknownalias, startIndex, busqueda))
                    }
                    spannable.toString()
                }

            }
            if (cardItem.idAnotable != -1) {

holder.careto.setImageResource(R.drawable.contact_icon)
                holder.careto.setColorFilter(
                    cardItem.colorFoto,
                    PorterDuff.Mode.MULTIPLY
                )
            } else {

holder.careto.setImageResource(R.drawable.notfound)
            }
            if (cardItem.clickable) {

holder.itemView.setOnClickListener(View.OnClickListener { l:
View? ->
                    val intent = Intent(context,
ContactoActivity::class.java)
                    intent.putExtra("id",
cardItem.idAnotable)
                    context!!.startActivity(intent)
                })
            }
        }
    }

recyclerView!!.setLayoutManager(LinearLayoutManager(context)
)
    recyclerView.setAdapter(newAdapter)
}
```

- **ApplicationUpdater.kt**: Singleton que genera la notificación de descarga y la petición de permisos de instalación de orígenes desconocidos

```
fun downloadAndInstallAPK(
    context: Context,
    url: String,
    fileName: String,
```

```
    unknownAppsPermissionLauncher: ActivityResultLauncher<Intent>
) {
    if (!context.packageManager.canRequestPackageInstalls()) {
        handleUnknownAppSourcesPermission(context,
unknownAppsPermissionLauncher)
        return
    }

    Log.d("GithubController", "URI de descarga: ${url.toUri()}")
    val request = DownloadManager.Request(url.toUri())
        .setTitle("Actualizacion de SpyLook")
        .setDescription("Espere mientras se descarga la
actualización.")

.setNotificationVisibility(DownloadManager.Request.VISIBILITY_VIS
IBLE_NOTIFY_COMPLETED)
        .setDestinationInExternalFilesDir(context,
Environment.DIRECTORY_DOWNLOADS, fileName)
        .setAllowedNetworkTypes(
            DownloadManager.Request.NETWORK_WIFI or
                    DownloadManager.Request.NETWORK_MOBILE
        )
        .setAllowedOverMetered(true)
        .setAllowedOverRoaming(true)

    val downloadManager =
context.getSystemService(Context.DOWNLOAD_SERVICE) as
DownloadManager
    downloadId = downloadManager.enqueue(request)
    Log.d("GithubController", "Solicitud de descarga realizada.
ID de descarga: $downloadId")
 }

 private fun handleUnknownAppSourcesPermission(
     context: Context,
     unknownAppsPermissionLauncher:
ActivityResultLauncher<Intent>
 ) {
     val intent = Intent(
         Settings.ACTION_MANAGE_UNKNOWN_APP_SOURCES,
         ("package:" + context.packageName).toUri()
     )
     Toast.makeText(context, UNKNOWN_APPS_PERMISSION_MSG,
Toast.LENGTH_LONG).show()
     unknownAppsPermissionLauncher.launch(intent)

}
```

- **ForegroundShaderSpan.kt**: Aplica el shader de texto arcoíris al Spannable al que entra como parámetro

```
//La clase
class ForegroundShaderSpan(private val shader: Shader) :
CharacterStyle(), UpdateAppearance {
    override fun updateDrawState(tp: TextPaint) {
        tp.shader = shader
    }
}
//En un TextWatcher

cardItem.alias = cardItem.alias.let {
    val spannable = SpannableString(it)
    var startIndex = retriever.getStartIndex(busqueda,
cardItem.alias)
    if (startIndex >= 0) {
        val shader = LinearGradient(
            0f, 0f, holder.mostknownalias.textSize * 2, 0f,
            intArrayOf(
                context!!.getColor(R.color.red),
                context.getColor(R.color.yellow),
                context.getColor(R.color.green),
            ),
            floatArrayOf(0f, 0.5f, 1f),
            Shader.TileMode.MIRROR
        )

        setSpan(
           ForegroundShaderSpan(shader),
           startIndex,
           retriever.getSpanIntervalJump(busqueda,
cardItem.alias, startIndex),
           Spannable.SPAN_EXCLUSIVE_EXCLUSIVE
        )


holder.mostknownalias.post(LongTextScrollerAction(holder.mostknow
nalias, startIndex, busqueda))
    }
    spannable.toString()
}
```

- **StringWithSpacesIndexRetriever.kt**: Permite que la búsqueda se pueda realizar sin escribir espacios manteniendo el shader de arcoíris bien aplicado

```
class StringWithSpacesIndexRetriever {

    var contador = 0
    var ultimaBusquedaLength = 0
    private var indicesDeEspacios = listOf<Int>()
    fun getSpanIntervalJump(busqueda: String, texto: String,
startIndex: Int): Int {
        contador = 0
```

```
            val indexUltimaLetra = busqueda.length - 1
            indicesDeEspacios = indicesDeEspacios.filter { it >
    startIndex }
            indicesDeEspacios.forEach {
                if (indexUltimaLetra+contador > 0 &&
    startIndex+indexUltimaLetra+contador >= it) {
                    contador += 1
                }
            }
            if (contador < 0) {
                contador = 0
            }
            ultimaBusquedaLength = busqueda.length
            var fin = startIndex + busqueda.length + contador
            if (fin > texto.length) {
                fin = texto.length
                contador -= 1
            }
            return fin
        }

        fun getStartIndex(busqueda: String, texto: String): Int {
            contador = 0
            val indexUltimaLetra = texto.replace(" ",
    "").lowercase().indexOf(busqueda)
            indicesDeEspacios = texto.mapIndexedNotNull { index, c ->
    index.takeIf { c == ' ' } }
            indicesDeEspacios.forEach {
                if (indexUltimaLetra+contador >= it) {
                    contador += 1
                }
            }
            return indexUltimaLetra+contador
        }
    }
```

view/

- **accounts/**

```
view/accounts/
├── CuentaActivity.kt
├── NuevaCuentaActivity.kt
└── fragments/
```

Activities y fragments para el CRUD de cuentas

- **common/**

```
view/common/
├── MainActivity.kt
└── fragments/
```

Activity principal (`MainActivity.kt`) y fragments comunes que pueden reutilizarse en un futuro.

- **contacts/**

```
view/contacts/
├── ContactoActivity.kt
├── NuevoContactoActivity.kt
└── fragments/
```

Activities y fragments para el CRUD de contactos.

- **groups/**

```
view/groups/
├── GrupoActivity.kt
├── NuevoGrupoActivity.kt
└── fragments/
```

Activities y fragments para el CRUD grupos.

- **sucesos/**

```
view/sucesos/
├── NuevoSucesoActivity.kt
├── SucesoActivity.kt
└── fragments/
```

Activities y fragments para el CRUD de sucesos/eventos.

Activities y vistas principales

Grupos

- `NuevoGrupoActivity.kt`: Permite crear un nuevo grupo, seleccionando a su creador y a sus miembros. El grupo debe tener por lo menos un creador y un miembro, además de un nombre
- `GrupoActivity.kt`: Muestra el TabLayout de grupos con un `GroupSliderAdapter.kt` aplicado para definir el Fragment asociado a cada pestaña.
    - **Fragments**:
        - `MiembrosFragment.kt`: Muestra al creador y a los miembros de un grupo en dos RecyclerView (uno para el creador, otro para los miembros) a los cuales les asigna un

ContactoCardAdapter.kt para acceder rápidamente a la información de cada miembro al pulsar sobre ellos. Si se mantiene pulsado sobre el miembro se permite eliminar al contacto permanentemente bajo previo aviso

Contactos

- NuevoContactoActivity.kt: Permite crear un nuevo contacto, asignandole nombre, alias más conocido, fecha de nacimiento, localidad, estado o provincia y país. El campo de fecha de nacimiento tiene el singleton de DateTextWatcher.kt asignado para asegurar que la fecha de nacimiento está comprendida entre el 1 de enero de 1970 y el dia de hoy La fecha de nacimiento es después almacenada en formato LocalDate, puesto que a la hora de instanciar un objeto de la entidad Contacto.kt se llama a un método estático de la clase para calcular la edad. El cálculo de la edad se basa en comprobar si el día de hoy es mayor que su fecha de cumpleaños reemplazando su año por el año actual y restándole un dia, de esta forma el dia del cumpleaños de un contacto su edad aumentará automáticamente
- ContactoActivity.kt: Muestra el TabLayout de grupos con un ContactSliderAdapter.kt aplicado para definir el Fragment asociado a cada pestaña.
    - **Fragments**:
        - AmigosFragment.kt: Muestra una lista de los amigos asociados al contacto y les aplica un AmigoCardAdapter.kt el cual permite abrir una ContactoActivity.kt en base al amigo pulsado y eliminar la amistad al mantener pulsada la tarjeta del amigo
        - ContactGroupsFragment.kt: Muestra la lista de grupos que ha creado o a los que pertenece el contacto. Le aplica un GruposDeContactoCardAdapter.kt para acceder a GrupoActivity.kt al pulsar sobre un grupo y eliminar la relación con el grupo al mantener pulsado En el caso de que el usuario sea
        - InformacionFragment.kt: Muestra los datos del contacto y un RecyclerView con las anotaciones asociadas a este. Las anotaciones en cualquier caso pueden ser editadas o eliminadas desde el pop up que aparece en pantalla al pulsar sobre ellas

Sucesos

- NuevoSucesoActivity.kt: Permite crear un nuevo suceso, detallando su título, descripción, fecha, causante e implicados Los implicados son opcionales, un suceso solo necesita un causante para ser creado El campo de descripción se ajusta de forma dinámica a la longitud del texto, cambiando su cantidad de líneas y ampliando la altura del campo La activity permite hacer scroll sobre su totalidad, exceptuando la parte superior con el nombre del suceso, si el contenido no cabe en la pantalla

- SucesoActivity.kt: Muestra el TabLayout de grupos con un SucesoSliderAdapter.kt aplicado para definir el Fragment asociado a cada pestaña.

    - **Fragments**:
        - ImplicadosFragment.kt: Muestra una lista de los contactos implicados en el suceso (sin contar al causante) y les aplica un ContactoCardAdapter.kt el cual permite abrir una ContactoActivity.kt en base al implicado pulsado y eliminar al contacto permanentemente bajo previo aviso al mantener pulsado El fragmento permite hacer scroll sobre su totalidad si el contenido no cabe en la pantalla
        - SucesoDataFragment.kt: Muestra los datos relativos a un suceso y al causante del mismo. El campo de descripción se ajusta dinámicamente al tamaño de la descripción,

separando el texto en varias líneas de la misma forma que lo hacía en `NuevoSucesoActivity.kt`. El fragmento permite hacer scroll sobre su totalidad si el contenido no cabe en la pantalla

- **Accounts**
  - `NuevaCuentaActivity.kt`: Permite crear una nueva cuenta con nick, link, nombre de la red social, propietario y usuarios Al igual que los sucesos, solo es necesario un propietario, los usuarios son opcionales y sirven para llevar un registro de cuantas personas tienen acceso a esa cuenta La activity permite hacer scroll sobre su totalidad, exceptuando la parte superior con el nombre de la cuenta, si el contenido no cabe en la pantalla
  - `CuentaActivity.kt`: Muestra el TabLayout de grupos con un `CuentaSliderAdapter.kt` aplicado para definir el Fragment asociado a cada pestaña.
    - **Fragments**:
      - `UsuariosCuentaFragment.kt`: Muestra los usuarios con acceso a la cuenta (sin contar al creador) en un RecyclerView con un `ContactoCardAdapter.kt` que permite abrir una `ContactoActivity.kt` en base al usuario pulsado y eliminar al contacto permanentemente bajo previo aviso al mantener pulsado El fragmento permite hacer scroll sobre su totalidad si el contenido no cabe en la pantalla
      - `CuentaDataFragment.kt`: Muestra los datos relativos a una cuenta y al propietario de la misma. El campo de link permite hacer click sobre él y abrir el enlace en la app correspondiente o, en su defecto, en el navegador. El fragmento permite hacer scroll sobre su totalidad si el contenido no cabe en la pantalla - **Common**
  - `MainActivity.kt`: Es la pantalla principal de la aplicación. Contamos con una barra de búsqueda en la parte superior de la pantalla, un botón con el texto *Nuevo* justo debajo y un RecyclerView donde se cargan las tarjetas. Por defecto al iniciar la app se muestran los usuarios, pero en la parte inferior de la pantalla hay un pequeño menú con dos opciones: Contactos y Grupos. Al pulsar en una de estas el RecyclerView se actualiza, cambiando el tipo de tarjetas que se muestran en el RecyclerView por el adecuado para cada entidad y se devuelve una respuesta háptica para simular la pulsación de un botón físico (cualquier elemento interactuable de la UI devuelve una respuesta háptica similar)

Al pulsar en alguno de los elementos nos llevará a su respectiva activity de datos, pero si pulsamos en *Nuevo* nos llevará a crear un nuevo elemento del tipo seleccionado

-**Fragments**:

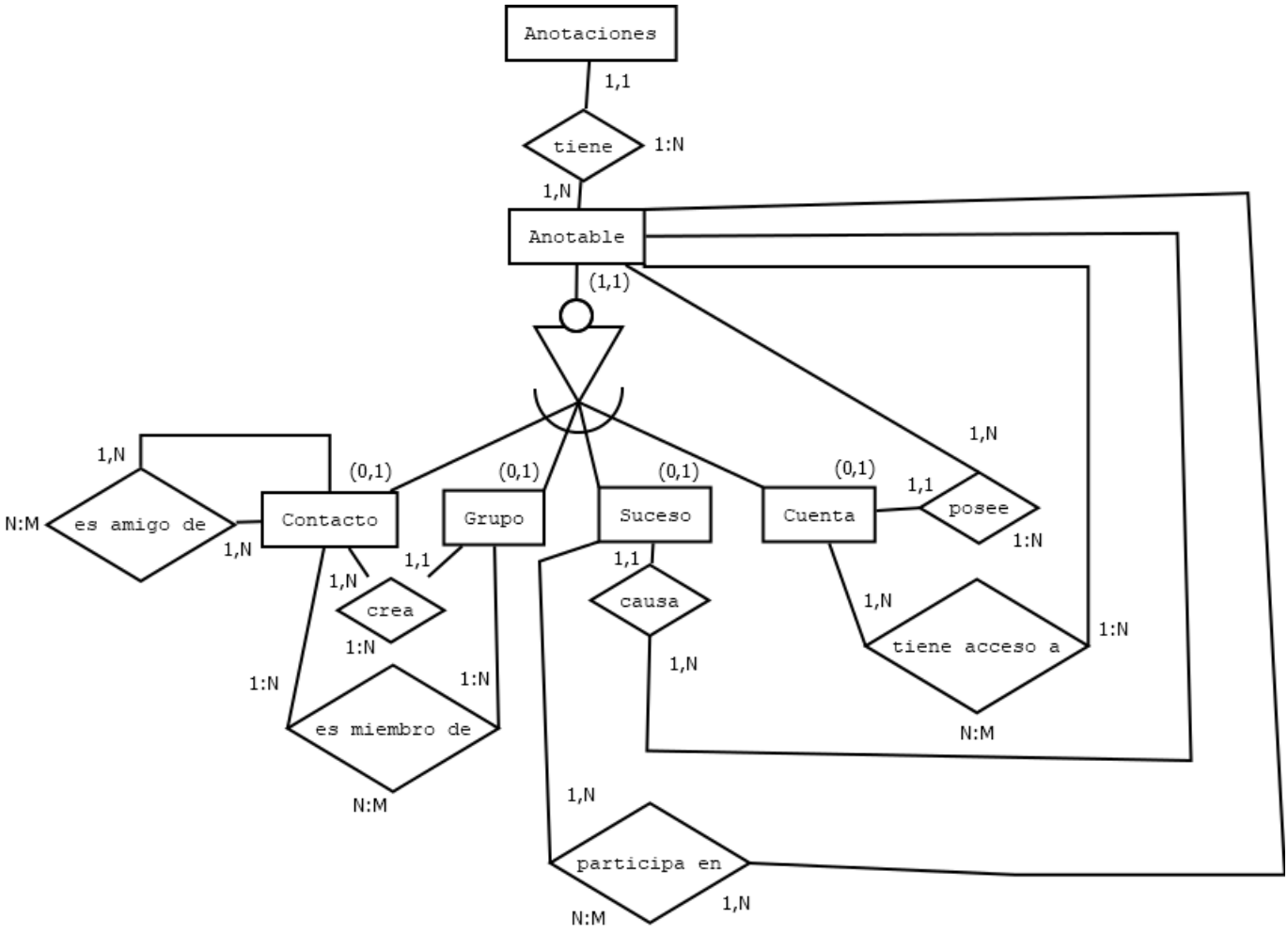
- `AnotacionesFragment.kt`: Fragment genérico para que cualquier elemento anotable pueda disponer rápidamente de un fragmento donde añadir, modificar y eliminar anotaciones. Una vez se pulsa sobre una anotación se infla un pop up desde el que podemos leer el titulo de la anotación, ver la fecha en la que se creó, leer el contenido de la anotación en un cuadro de texto adaptable similar al de `SucesoDataFragment.kt` con la diferencia de que este cuadro es scrolleable en vez de estirarse hasta mantener visible todo el texto Esta vista cuenta con un botón de *Cerrar* para cerrar el mensaje, *Borrar* para borrar la anotación y *Editar* para editar el mensaje en un nuevo pop up similar a este Implementado en `GroupSliderAdapter.kt`, `CuentaSliderAdapter.kt` y `SucesoSliderAdapter.kt`

- `CuentasFragment.kt`: Fragment genérico para que contactos y grupos puedan disponer rápidamente de un fragmento donde asociar y eliminar cuentas. Implementado en `ContactSliderAdapter.kt`
- `SucesosFragment.kt`: Fragment genérico para que contactos y grupos puedan disponer rápidamente de un fragmento donde asociar y eliminar sucesos. Implementado en `ContactSliderAdapter.kt`

Base de datos

La base de datos es una base de datos local SQLite manejada con la biblioteca de persistencia Android Room, que actúa como una capa de abstracción sobre SQLite y permite manejar todo el CRUD de los elementos mediante DAOs generados en tiempo de compilación a partir de interfaces

El diagrama E/R sería algo tal que así:



Como se puede ver en la imagen, todos los elementos a excepción de Anotación forman una jerarquía total exclusiva con Anotable, dotando a la aplicación de una flexibilidad muy amplia. Aunque el diagrama pueda parecer caótico y con alta tendencia a provocar fallos, la flexibilidad que otorga este diseño permite una interconexión entre los datos muy alta. Ejs:

- Se podría implementar que un suceso sea provocado por otro suceso, algo así como un *efecto Mariposa*
- Se podría implementar una característica que permita asociar a una cuenta un grupo como creador/propietario, de esta forma se podría hacer referencia a un conjunto de personas (creador y miembros del grupo) como dueños de esta en vez de simplemente tener un unico dueño

Cosas aprendidas durante el desarrollo / curiosidades

Acceso a atributos de objetos

Según el propio *linter* integrado en Android Studio, en Kotlin no se accede a las propiedades de un objeto mediante el getter, sino accediendo a la propiedad expuesta directamente (sin modificador de acceso) Por este motivo se puede encontrar a lo largo del código muchas asignaciones de atributos sin acceder a getters/setters o anotaciones como `@JvmField` sobre los atributos, que le indican al compilador de Kotlin que no genere getters/setters para ese atributo y lo exponga

Tuberías y programacion funcional en Kotlin

En Kotlin, al ser un lenguaje interoperable con Java, se puede hacer uso de los Streams y todo lo que esto conlleva. No obstante, Kotlin cuenta con funciones propias de programación funcional que aportan una flexibilidad muy alta a la hora de programar

Este es el caso de funciones como por ejemplo `isEmpty{}`, que permite en muchos casos reemplazar por completo a estructuras if-else

```
val nombre = "Carlos"
var saludo : String = ""
saludo.isEmpty{
   saludo = "Hola ${nombre}, ¿qué tal?"
}
```

Otra de las funciones que más potencial tienen es `.apply{}`. Este método se puede usar desde cualquier objeto y devuelve el mismo objeto después de aplicar el contenido de la función anónima en su interior, a la cual pasa el objeto como parámetro De esta forma podemos ampliar cadenas de tuberías con acciones que de otra forma no sería posible, como por ejemplo:

```
var lista = listOf(1,2)
lista.filter{ it != 1}
  .toMutableList()
  .apply{
    add(3)
    add(4)}
  .forEach{ print("${it} ") } //Resultado: 2 3 4
```

Uso de Optional y tipos no-nulos

En Kotlin, aunque se pueda usar la clase Optional igual que en Java, se opta por utilizar la verificación de valores nulos (null safety) propia del sistema de tipos de Kotlin.

Todos los valores son por defecto no-nulos, es decir, se requiere asignar un valor al momento de la declaración y este nunca puede ser nulo Mediante el uso de palabras clave como "lateinit" delante del

nombre de la variable permiten separar declaración de inicialización solo en objetos (los tipos primitivos no son compatibles con esta palabra clave)

```
private lateinit var inicializacionSeparada : String
```

En caso de que queramos que un objeto sea nulo se le asigna una interrogación al tipo, que se tiene que declarar explicitamente

```
var nula : String? = null
var nula2: String?
```

Esto introduce un nuevo requerimiento a la hora de usar la variable, asegurarse de que al momento de acceder a la variable no es nula Se puede hacer de dos formas:

```
var nula : String? = "Hola mundo"

print(nula?.length) //Con ? se imprime la longitud solo si la variable no
es nula
print(nula!!.charAt(0)) //Con !! se fuerza a la JVM a asumir que la
variable no es nula nunca, pudiendo resultar en un NullPointerException
```