Proyecto Integrado

mi lista de la compra

Autor: Alfonso Carlos Serrano Martín

INDICE

1. INTRODUCCIÓN

Este proyecto cubre una necesidad de compartir una lista con diferentes tipos de permisos a alguien y este pueda ejercer esos permisos. También tiene un lector de códigos de barras para añadir artículos. En comparación de otras apps es más completo y pido menos información.

2. OBJETIVOS DEL PROYECTO

El objetivo final es tener una aplicación que contenga una colección de lista de la compra. Objetivos adicionales son tener un lector de código de barra y poder dar permisos a los demás. Las listas se almacenará en una base de datos SQLite o en Firebase Firestore de forma que podrá ser consultada por el usuario. El inicio de sesión será a trabes de Firebase Auth.

3. PLANIFICACIÓN DEL PROYECTO

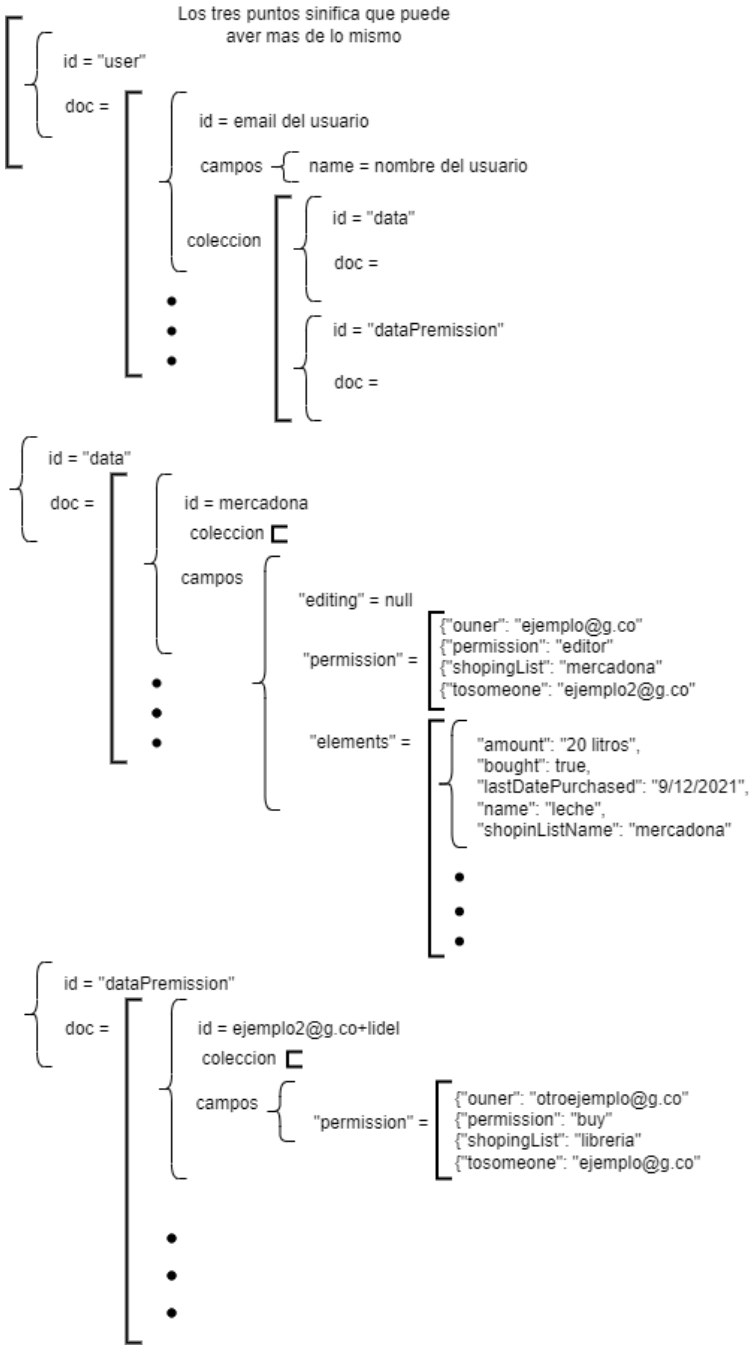
- ⚫ Registro y login de usuario.
- ⚫ Menú
  - ○Mis listas
    - ○ Botón para añadir más listas con su respectivo formulario
    - ○ por cada lista aparecerán un nombre y una cantidad de artículo que faltan por compra (se cambió por donde está guardado la lista)
    - ○ ~~También aparecer aun botón en modo de ">" para un sub menú tendrá las opciones de comprar e historial.~~ → al pulsar te llevará a una página que te dirá lo que contiene y te dará opciones de historial, comprar y editar. En caso de que la lista esté alojada en el Sever podrás administrar permisos.
    - ○ ~~Si clicas te saldrá la zona para añadir más~~ (se cambió por la acción editar del apartado anterior.)
  - ○Listas de otros
    - ○ Por cada lista aparecer un nombre ~~y una cantidad de artículo que faltan por comprar~~ y los permisos que tienes → (se añadió) y el dueño de la lista
    - ○ Al pulsar te llevará a una página que te dirá lo que contiene y te dará opciones de historial, comprar y editar de pendiendo del permiso que tengas
  - ○ Opciones
  - ○ Acerca de
  - ○ Cerrar sesión

No pongo tiempo un cronograma porque también tiene una parte de investigación que ha sido importan. Como: Kotlin, Firebase (auth, auth Google, Firestore) como leer un código de barras, como interpretar código de barras, buscar APIs para esto, ver que esas APIs cuestan dinero y poner una solución. Como manejar permisos con Firestore y si sale más rentable Firestore versus base de datos convencional

4. ANÁLISIS Y DISEÑO DEL SISTEMA

4.1. Modelo conceptual (diagrama entidad-relación)

Firebase Firestore tiene similitud con XML al guardar datos en forma de árbol.
Para un ejemplo ir al Anexo I: Firestore en XML.json



4.2. Creación de la base de datos y consultas principales.

Realmente cada vez que se guarda algo se crea la collection y doc necesario.
Las consultas son el mismo código. Un ejemplo al meter datos:

```
fun setProfileData(userName: String) {
   db.collection(usersCollection).document(userEmail)
   .set(hashMapOf(NAMEKEY to userName))
}
```

Al llamar es esta función le pasaremos el nombre del usuario para guardarlo en la base de datos. usersCollection = "users", NAMEKEY = "name", userEmail ace referencia al email del usuario

Un ejemplo al sacar datos:

```
fun getProfileData(): Task<DocumentSnapshot> {
    return db.collection(usersCollection).document(userEmail!!).get()
}
getProfileData().addOnSuccessListener {
            val name = it.get(fbFirestoreService.NAMEKEY) as String  //Aqui se devuelve el valor
        }
```

Un ejemplo para listar todos los usuarios seria:

```
fun getAllProfile(): Task<QuerySnapshot> {
    return db.collection(usersCollection).get()
}
   getAllProfile().addOnSuccessListener { value ->

       value .documents // esta es una lista con todos los usuarios.
       value .documents[0].id // este es el email del usuario 1

    }
```

Si quiere más ejemplos busca el fichero en el código fuente de ruta: *"\app\src\main\java\com\acasema\listadelacompra\service\FirebaseFirestoreService.kt"*

5. ESPECIFICACIONES DEL SISTEMA

5.1. Justificar el uso de la tecnología y el software empleado

“Android Studio”: facilidad a la hora de crear una aplicación
"Firebase Auth”: facilidad a la hora de iniciar sesión
"Firebase Firestore”: facilidad de implementación
"room”: Ya lo sabía (aunque me ha dado problemas al usar la nueva versión)
"journeyapps.barcodescanner”: facilidad de trabajo

5.2. Instalación y configuración de la aplicación

La instalación y configuración: no hace falta nada más que ejecutar el APK en un teléfono Android.

Librerías de terceros:

- Firebase (Versión 29.0.0)
- FireBase Analytics (Versión 20.0.0)
- Firebase Auth (Versión 8.0.0)
- Firebase Auth Google (Versión 19.0.0)
- Firebase Firestore(Versión 24.0.0)
- Room (room-runtime, room-ktx, room-compiler, version 2.3.0)
- journeyapps:zxing-android-embedded (Versión 4.3.0)

5.3. Elaboración de las especificaciones hardware del sistema

Sistema Android con SDK 24 a 31. (De Nougat a Snow Cone.)
Una cámara para el lector código de barra.

6. ESPECIFICACIONES DEL SOFTWARE

6.1. Descripción de las operaciones

• Operaciones de registro de usuario.

Actor → Usuario.
Descripción → Aquí van a registrarse.
Precondición → Ninguna.
Parámetros → Email, nombre, contraseña.
Postcondición → Email tiene que tener formato email,
nombre tiene como mínimo 3 caracteres,
contraseña min 8 char, una minúscula y un número.
Resultado → Inicia sesión e ir a colección de lista
Excepción → si precondición no se cumple salte cuál es el error

• Operaciones de identificación y acceso de un usuario.

Actor → Usuario.
Descripción → Aquí va a iniciar sesión.
Precondición → Ninguna.
Parámetros → Email, contraseña.
Postcondición → Credencial válida.
Resultado → Inicia sesión e ir a colección de lista.
Excepción → Credencial erróneas.

• Operaciones de crear o editar una lista

Actor → usuario.
Descripción → Aquí se crea o edita una lista
Precondición → Si hay lista, esta se edita esta bloqueando los campo nombre y si es online.
Si no hay lista, se crea una.
Parámetro → nombre y si es online. Por cada elemento está un nombre y cantidad
Postcondición → Que el nombre de la lista y el del elemento no sea nulo
Resultado → se guardan los datos en servidor si online está marcado y si no en Room
Excepción → si hay un elemento con nombre nulo o vacío se elimina de la lista y se prosigue.
Si está el nombre vacío se te notifica.

• Operaciones de comprar

Actor → Usuario.

Descripción → Aquí aparece una lista de elementos que no se han comprado todavía. Hay un botón en grande para marcar que se ha comprado esa cosa. Aparecerá en rojo y el botón se usa para desmarcarlo por si te equivocaste.

Precondición → tener una lista de elementos con el campo bought a falso

Parámetros → si está comprado y la fecha de hoy.

Postcondición → marcar si está comprado.

Resultado → la información se guardará en el instante que pulsas al botón por lo que podrás ir para tras que ya se habrá guardado los datos.

Excepción → Credencial erróneas.

• Operaciones de dar permiso

Actor → Usuario.

Descripción → Aquí es donde se le dan permisos a los demás usuarios.

Precondición → tener una lista online.

Parámetros → buscar un usuario

Postcondición → que el usuario exista.

Resultado → el usuario aparecerá con el permiso mínimo, que es ver.

Excepción → se avisará que el usuario no existe.

• Operaciones de cambiar o eliminar permiso

Actor → Usuario.

Descripción → Aquí es donde se cambia o se elimina el permiso.

Precondición → tener un usuario válido.

Parámetros → hay dos botones uno es para aumentar el permiso y otro para quitárselo

Postcondición → si ya tienes el permiso editar volverá al permiso ver

Resultado → el usuario aparecerá con el permiso mínimo, que es ver.

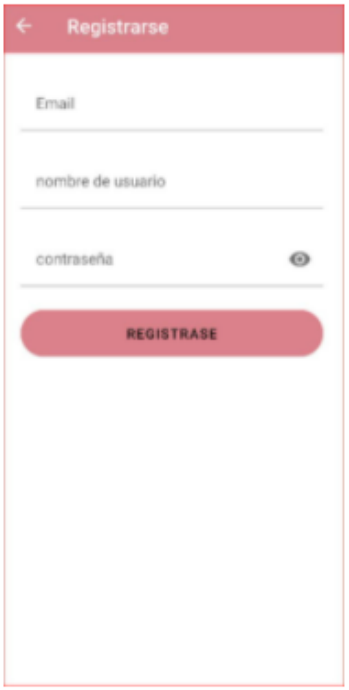
Excepción → Ninguna.

6.2. Interfaz y Navegación de la aplicación

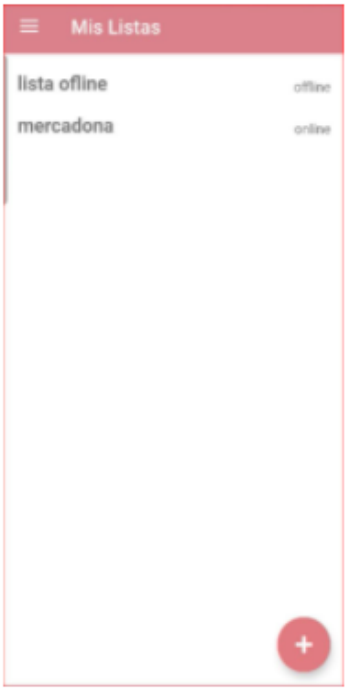
IMÁGENES:

1.-Inicio de sesión

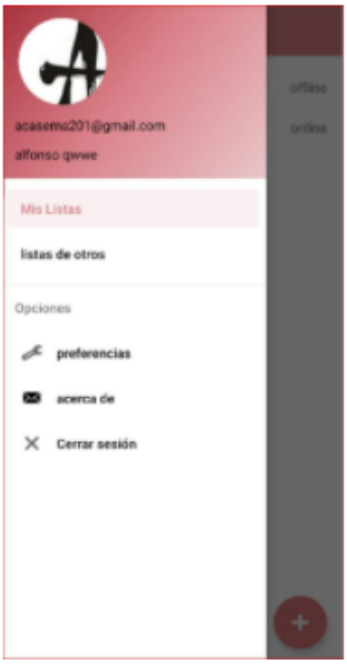
2.- Registrarse



3.- colección de listas



4.- Menú



5.- Listas de otros

6.- Ver

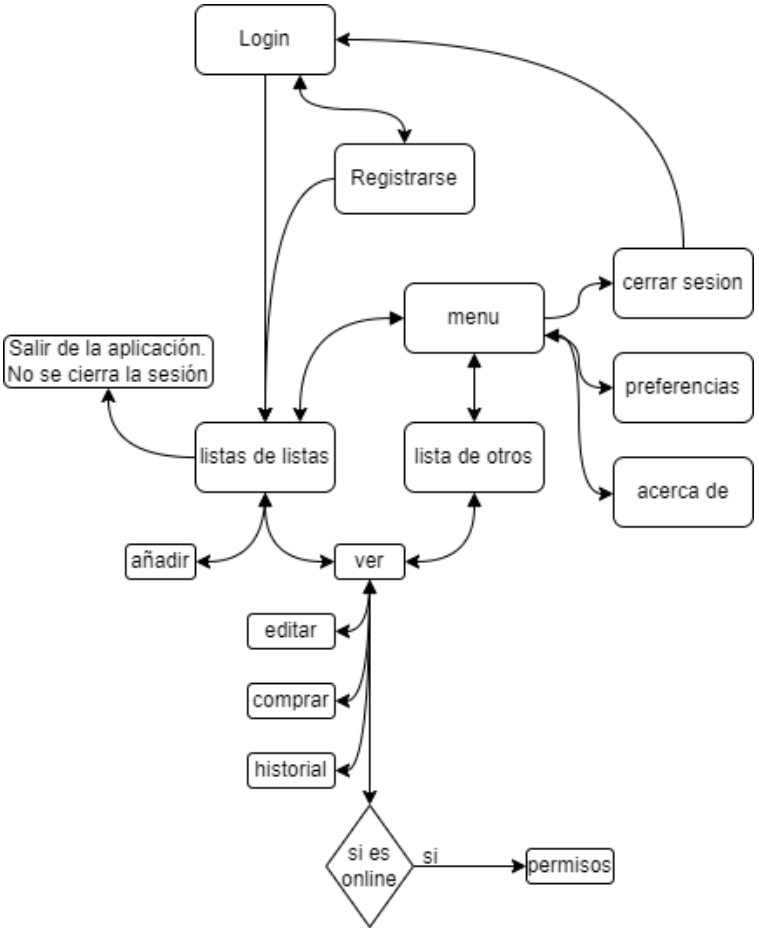
7.- Añadir/ editar

8.- Historial

9.- Comprar

10.- Permisos

Esquema de navegación



7. CÓDIGO FUENTE RELEVANTE

Código relacionado con Firebase Auth.

```
package com.acasema.listadelacompra.service

import com.google.android.gms.auth.api.signin.GoogleSignInAccount
import com.google.android.gms.tasks.Task
import com.google.firebase.auth.AuthResult
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.FirebaseUser
import com.google.firebase.auth.GoogleAuthProvider

class FirebaseAuthService {

    private val auth = FirebaseAuth.getInstance()

    fun login(email: String, password: String): Task<AuthResult> {
        return auth.signInWithEmailAndPassword(email, password)
    }
    fun signIn(email: String, password: String): Task<AuthResult> {
        return auth.createUserWithEmailAndPassword(email, password)
    }

    fun loginGoogle(account: GoogleSignInAccount): Task<AuthResult> {
        val credential= GoogleAuthProvider.getCredential(account.idToken,  accessToken: null)
        return auth.signInWithCredential(credential)
    }
    fun getUser(): FirebaseUser {
        return auth.currentUser!!
    }
    fun signOff() {
        auth.signOut()
    }
}
```

Creación de la base de datos.

```
@Database(entities = [ShopingList::class, Element::class], version = 2, exportSchema = false)
abstract class AppDatabase: RoomDatabase() {

    abstract fun ShopingListDao(): ShopingListDao
    abstract fun ElementDao(): ElementDao

    companion object{

        private  var database: AppDatabase? = null

        fun create(context: Context){
            database = Room.databaseBuilder(
                context.applicationContext,
                AppDatabase::class.java,  name: "database-app"
            ).allowMainThreadQueries().fallbackToDestructiveMigrationOnDowngrade().build()
        }

        fun getInstance(): AppDatabase{
            return database!!
        }
    }
}
```

Relación de la base de datos.

```
data class ShoppingListWithElement(
    @Embedded
    val shopingLis: ShopingList,

    @Relation(parentColumn = "name", entityColumn = "shopingListName")
    val elements:List<Element>
)
```

Tenía un problema y la solución fácil hubiera sido
recyclerview.setItemViewCacheSize(mDataset.size())
al final fue:

```
override fun onBindViewHolder(holder: AdapterListCreation.Viewholder, position: Int) {

    holder.nameElementEditTextListener.updatePosition(holder.adapterPosition)
    holder.amountElementEditTextListener.updatePosition(holder.adapterPosition)

    holder.etName.text =Editable.Factory().newEditable(list[holder.adapterPosition].name)
    holder.etAmount.text =Editable.Factory().newEditable(list[holder.adapterPosition].amount)
}
```

```
inner class Viewholder(view: View ): RecyclerView.ViewHolder(view){

    lateinit var nameElementEditTextListener : NameElementEditTextListener
    lateinit var amountElementEditTextListener : AmountElementEditTextListener
    var etName: EditText = itemView.findViewById(R.id.etName)
    var etAmount: EditText = itemView.findViewById(R.id.etAmount)

    constructor(view: View, nameElementEditTextListener : NameElementEditTextListener,
                amountElementEditTextListener : AmountElementEditTextListener) : this(view) {

        this.nameElementEditTextListener = nameElementEditTextListener
        this.amountElementEditTextListener = amountElementEditTextListener
        etName.addTextChangedListener(nameElementEditTextListener)
        etAmount.addTextChangedListener(amountElementEditTextListener)
    }
}
inner class NameElementEditTextListener : TextWatcher {
    private var position = 0
    fun updatePosition(position: Int) {
        this.position = position
    }
    override fun beforeTextChanged(charSequence: CharSequence, i: Int, i2: Int, i3: Int) {}
    override fun onTextChanged(charSequence: CharSequence, i: Int, i2: Int, i3: Int) {
        list[position].name = charSequence.toString()
    }
    override fun afterTextChanged(editable: Editable) {}
}
inner class AmountElementEditTextListener : TextWatcher {
    private var position = 0

    fun updatePosition(position: Int) {
        this.position = position
    }
    override fun beforeTextChanged(charSequence: CharSequence, i: Int, i2: Int, i3: Int) {}
    override fun onTextChanged(charSequence: CharSequence, i: Int, i2: Int, i3: Int) {
        list[position].amount = charSequence.toString()
    }
    override fun afterTextChanged(editable: Editable) {}
```

setItemViewCacheSize hubiera sido suficiente si la lista es pequeña, pero al no saber las magnitudes de dicha lista lo que hago es refrescar la vista desde fuera de la clase interna para que los valores estén bien puestos. Antes lo que pasaba era que al añadir un nuevo elemento si había hueco te lo ponía al final y si no abia espacio te lo ponía al principio y cuando eliminabas un objeto siempre borraba el último. Además, el editText conservaba el valor anterior y no se actualizaba a una nueva posición.

Conexión con la clase para leer códigos de barras.

```
private fun initScanner() {
    val options = ScanOptions()
    options.setDesiredBarcodeFormats(ScanOptions.ONE_D_CODE_TYPES)

    options.setOrientationLocked(true)
    options.setPrompt("Scan a barcode")
    options.setBeepEnabled(true)

    val barcodeFormats : MutableList<String> = mutableListOf()
    barcodeFormats.add(ScanOptions.EAN_13)
    barcodeFormats.add(ScanOptions.EAN_8)
    barcodeFormats.add(ScanOptions.UPC_A)
    barcodeFormats.add(ScanOptions.UPC_E)

    options.setDesiredBarcodeFormats(barcodeFormats)

    barcodeLauncher.launch(options)
}

private val barcodeLauncher = registerForActivityResult(ScanContract()) {
        result: ScanIntentResult →
    if (result.contents == null) {
        Toast.makeText(requireContext(), text: "Cancelled", Toast.LENGTH_LONG).show()
    } else {
        viewModel.resultReadCodePost(result.contents)
    }
}
//endregion
```

Dependencias del proyecto.

```
def room_version = "2.3.0"
implementation "androidx.room:room-runtime:$room_version"
implementation "androidx.room:room-ktx:$room_version"
kapt "androidx.room:room-compiler:$room_version"

//FireBase
implementation platform('com.google.firebase:firebase-bom:29.0.0')
//FireBase analytics
implementation 'com.google.firebase:firebase-analytics-ktx:20.0.0'
//FireBase Authentication
implementation 'com.firebaseui:firebase-ui-auth:8.0.0'
//FireBase Authentication Google
implementation 'com.google.android.gms:play-services-auth:19.2.0'
//FireBase Firestore database
implementation 'com.google.firebase:firebase-firestore:24.0.0'
// When using the BoM, you don't specify versions in Firebase library dependencies
//implementation 'com.google.firebase:firebase-crashlytics-ktx'

//Zxing: used to read barcode and code qr https://github.com/journeyapps/zxing-android-embedded
//licencia: Con licencia de Apache License 2.0
implementation 'com.journeyapps:zxing-android-embedded:4.3.0'
```

8. CONCLUSIONES DEL PROYECTO

Yo cambiaría la base de datos de Firebase Firestore por una base de datos convencional. No se implementó esto porque cuando me di cuenta ya era tarde y ya estaba implementado con Firebase. También cambiaria el sitio donde saco la información del código de barra, el cómo las busco y contratar el servicio. O quitar esta función. Reorganizaría las clases y vería si se pueden reutilizar o agrupar ViewModel. En mi opinión le invertiría más tiempo en la organización al principio del proyecto y no metería idead nuevas durante el desarrollo. Y por último me ha gustado Kotlin más por lo que repetiría lenguaje en un siguiente proyecto personal.

9. APÉNDICES

En este apartado se pueden incluir anexos como las pruebas ejecutadas al programa, informes que se actúen en el programa, ejemplos de facturación, listados….

10. BIBLIOGRAFÍA

Firebase:
https://firebase.google.com/docs?authuser=0
https://youtube.com/playlist?list=PLNdFk2_brsRcaGhfeeiVkW72qTYcn_nfQ
Kotlin:
https://developer.android.com/kotlin/learn
URL útil:
https://material.io/resources/color/#!/?view.left=0&view.right=0
https://romannurik.github.io/AndroidAssetStudio/index.html
https://www.flaticon.es/