Documento Explicativo Tarea 5

1. Introducción

En esta tarea se ha desarrollado una aplicación Android llamada "ContactosApp" que permite gestionar una agenda de clientes. Puedes hacer CRUD con los contactos y exportarlos a un archivo en formato Json.

La aplicación cumple con los requisitos de:

-Persistencia de datos utilizando **SQLite** para el almacenamiento local

-**SharedPreferences** para la gestión de sesiones de usuario,

-Uso de ficheros externos (**JSON**) para la exportación de datos con la librería externa (**Gson**) .

-Además, se han integrado librerías externas para la carga de imágenes (**Coil**), en su versión para Kotlin y Jetpack Compose

2. Desarrollo de la tarea paso a paso

A continuación se describen los pasos seguidos para la implementación:

2.1. Configuración del proyecto y Dependencias

Se ha creado un proyecto en Android Studio utilizando la plantilla "Empty Activity" y lenguaje Kotlin. Se han añadido las siguientes dependencias en el build.gradle:

- **Gson :** Para la exportación de objetos a formato JSON.
- **Coil:** Para la carga asíncrona de imágenes desde URLs.
- **LazyColumn:** Para el listado eficiente de contactos.

2.2. Base de Datos (SQLite)

Se ha creado la clase ContactosDB que hereda de SQLiteOpenHelper.

- **Tabla creada:** contactos con los campos id, nombre, telefono, email e imagenid.
- **Métodos implementados:** Se han desarrollado los métodos CRUD (Crear, Leer, Actualizar, Borrar) necesarios para la gestión de la información.

2.3. Gestión de Sesión (SharedPreferences)

Se ha implementado la LoginScreen.kt. Al iniciar sesión, se guardan las preferencias en modo PRIVATE. Si el usuario marca "Recordar sesión", la app guarda el estado y, en el siguiente inicio, salta directamente a la pantalla principal.

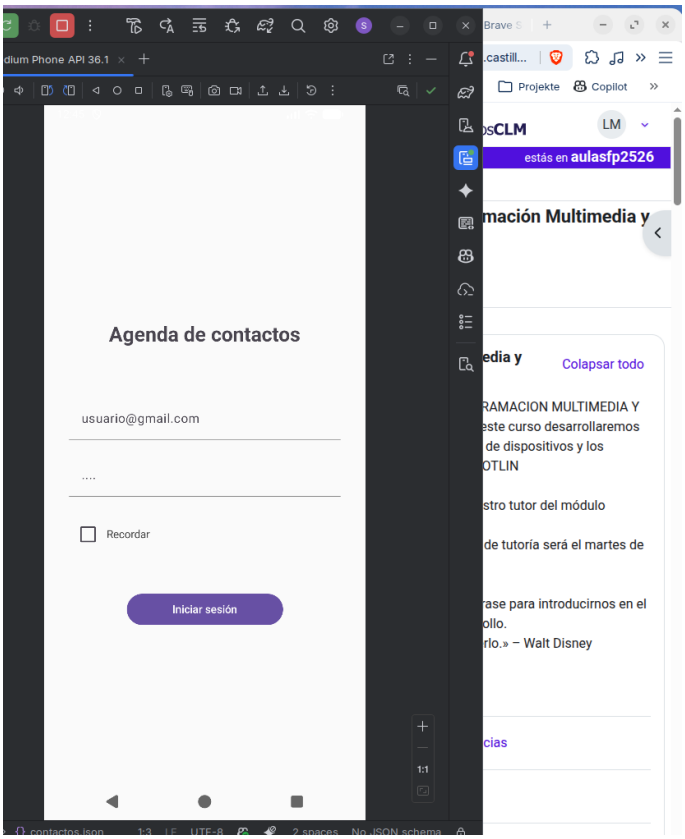
2.4. Interfaz de Usuario y Lógica

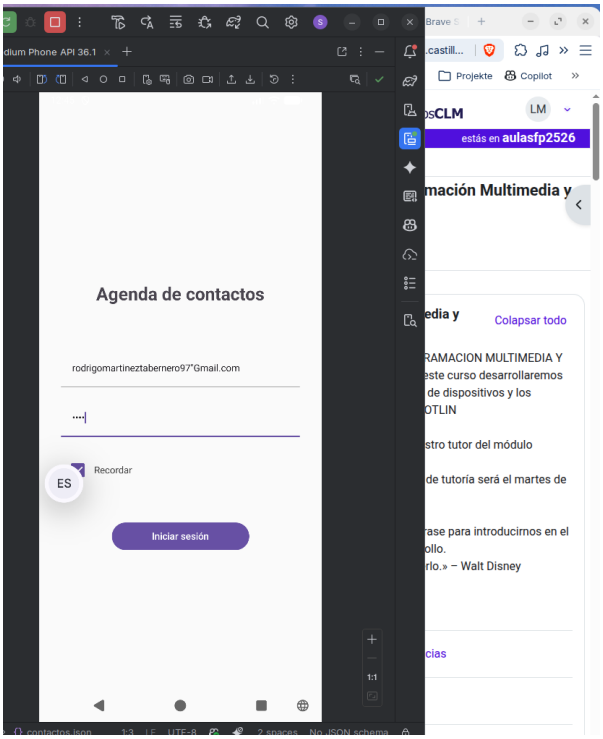
- **MainActivity:** (Cambiar)Contiene el RecyclerView y gestiona la comunicación con la base de datos. Implementa un menú superior para la exportación y un menú Popup contextual para cada elemento de la lista.
- **LoginScreen.kt:** Comentada en el punto anterior
- **ListaContactosScreen.kt:** Renderiza la lista de contactos con una LazyColumn. Tiene una TopAppBar que usamos como menú principal con la opción de Exportar a JSON, Eliminar todos los contactos y cerrar sesión. También los items individuales te permiten crear nuevos contactos, editar o eliminarlos uno a uno.
- **DetalleContactoScreen.kt:** Permite tanto el alta de nuevos contactos como la edición de los existentes, cargando la imagen mediante Coil si existe una URL válida.

3. Pruebas (Capturas de pantalla)

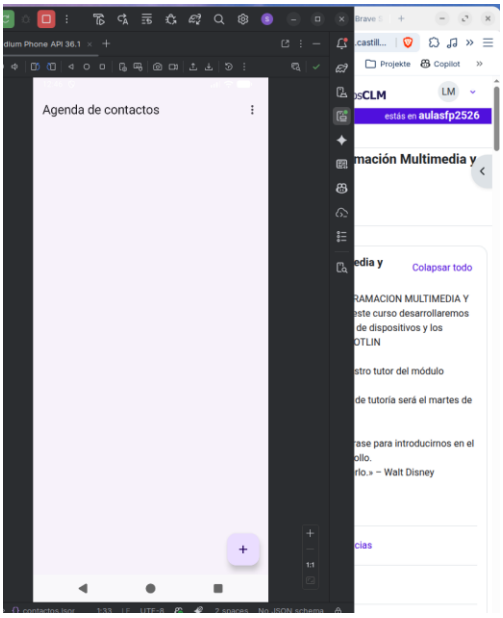
EMULADOR API 36

3.1. Pantalla de Login (solo volvemos aquí si cerramos sesión o si no le dimos a recordar)

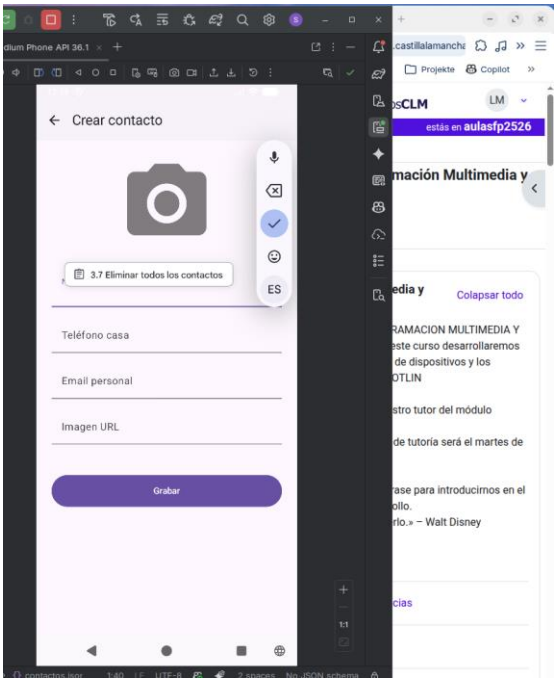


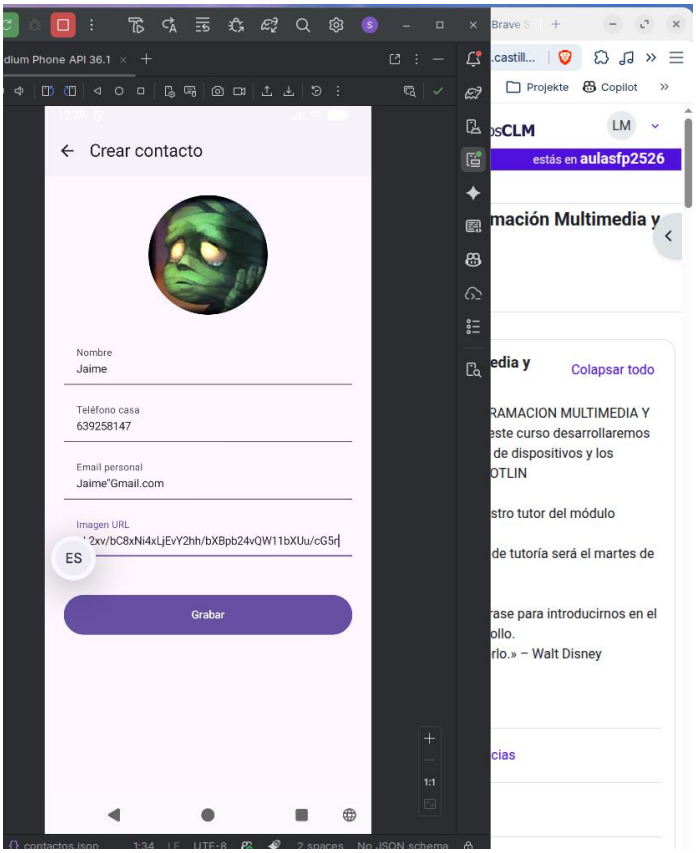


3.2. Lista de Contactos (Estado inicial / Vacía)

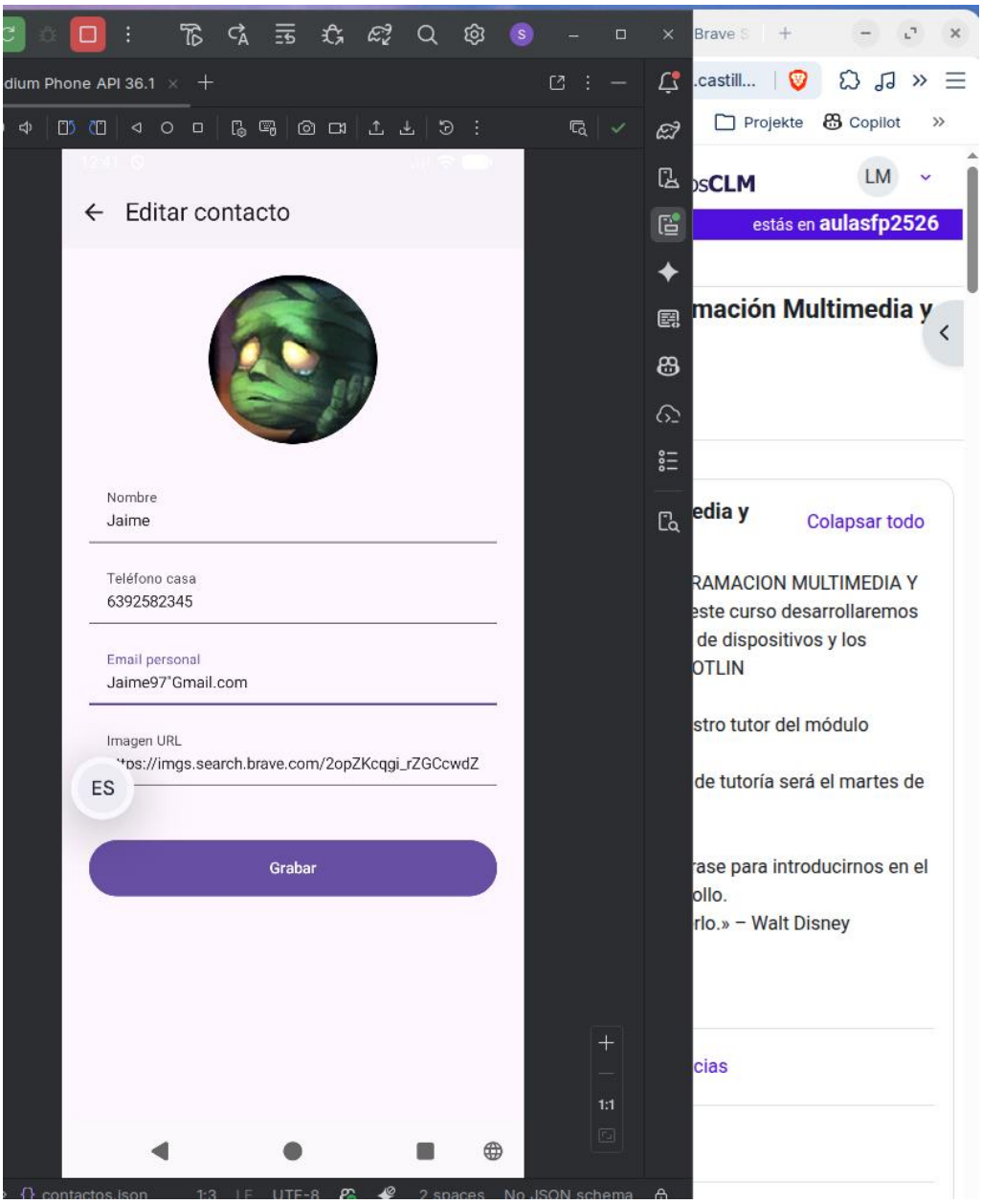


3.3. Formulario de Alta/Edición

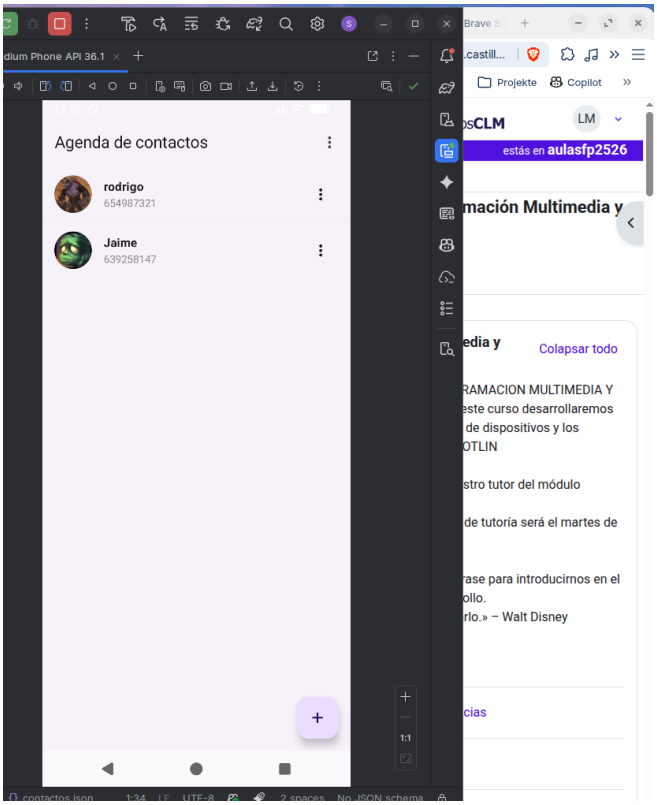



Crear contacto
Nombre
Jaime
Teléfono casa
639258147
Email personal
Jaime"Gmail.com
Imagen URL
Grabar
ES
Colapsar todo
estás en aulasfp2526

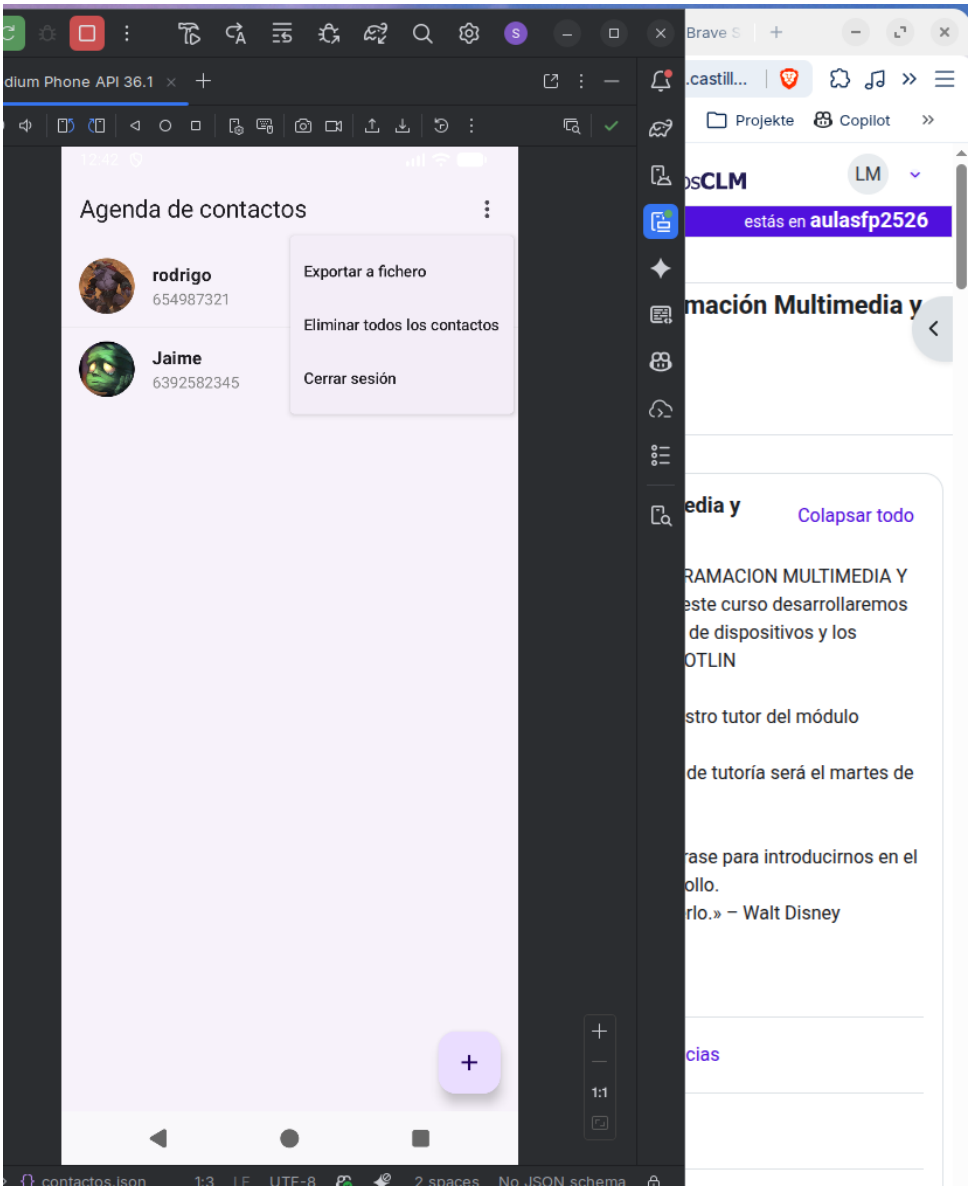
Edición

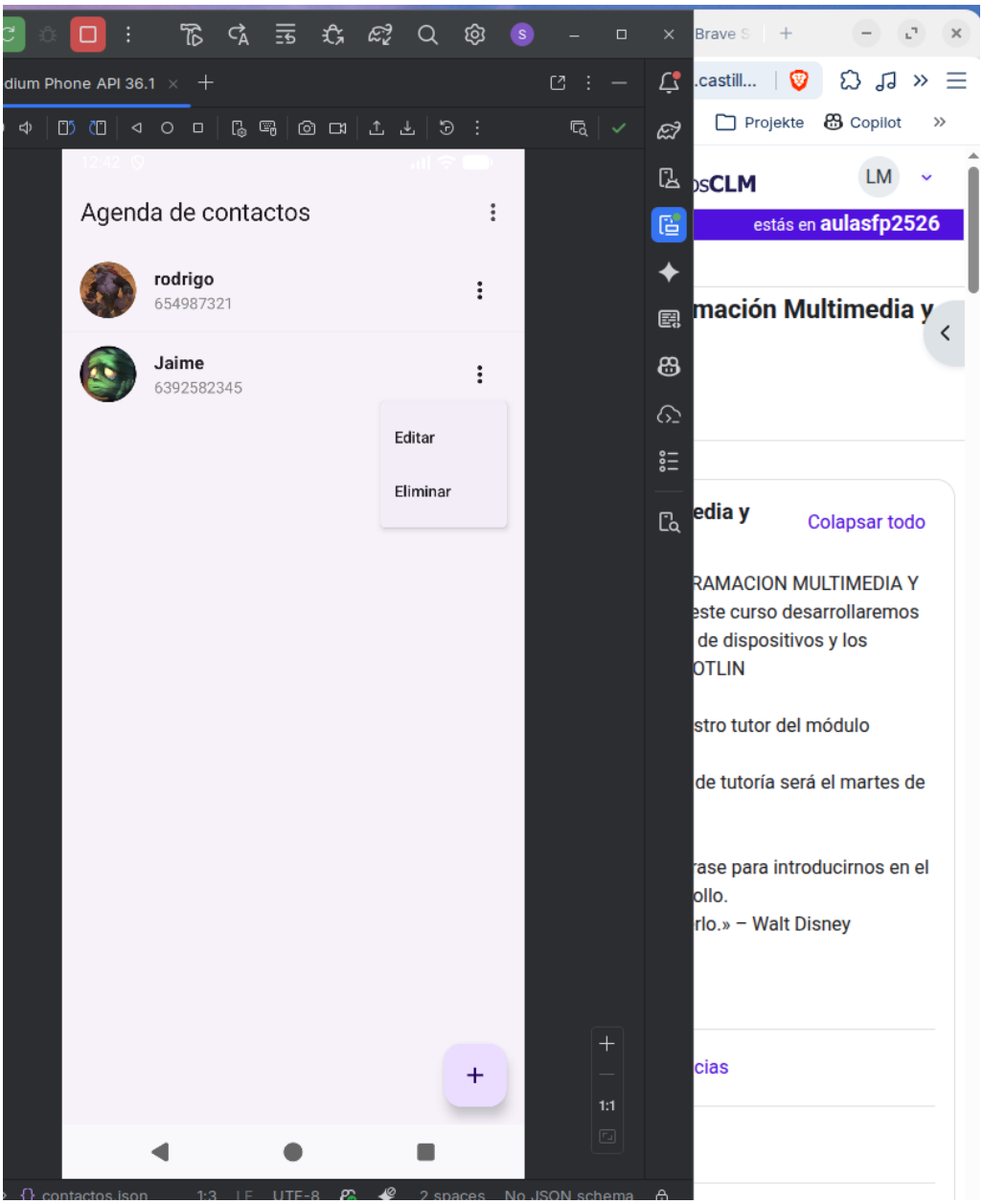


3.4. Lista de Contactos con datos

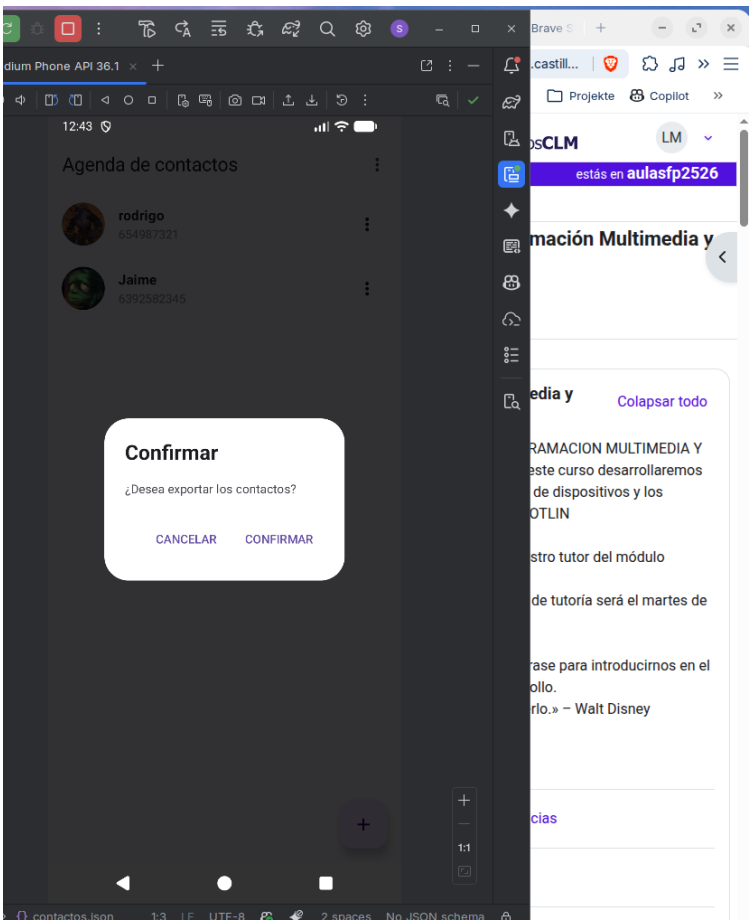


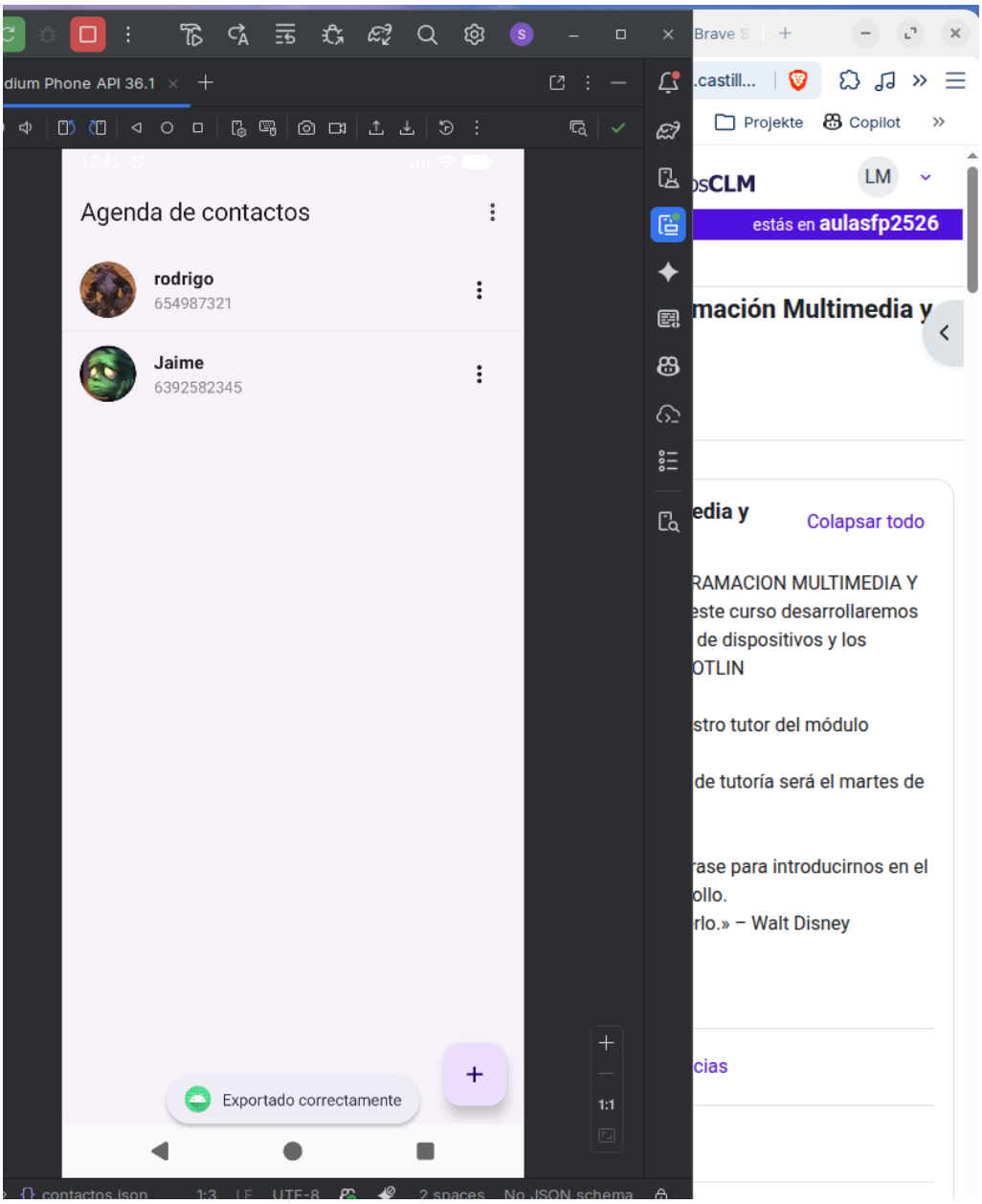
3.5. Menú Popup y Funcionalidades

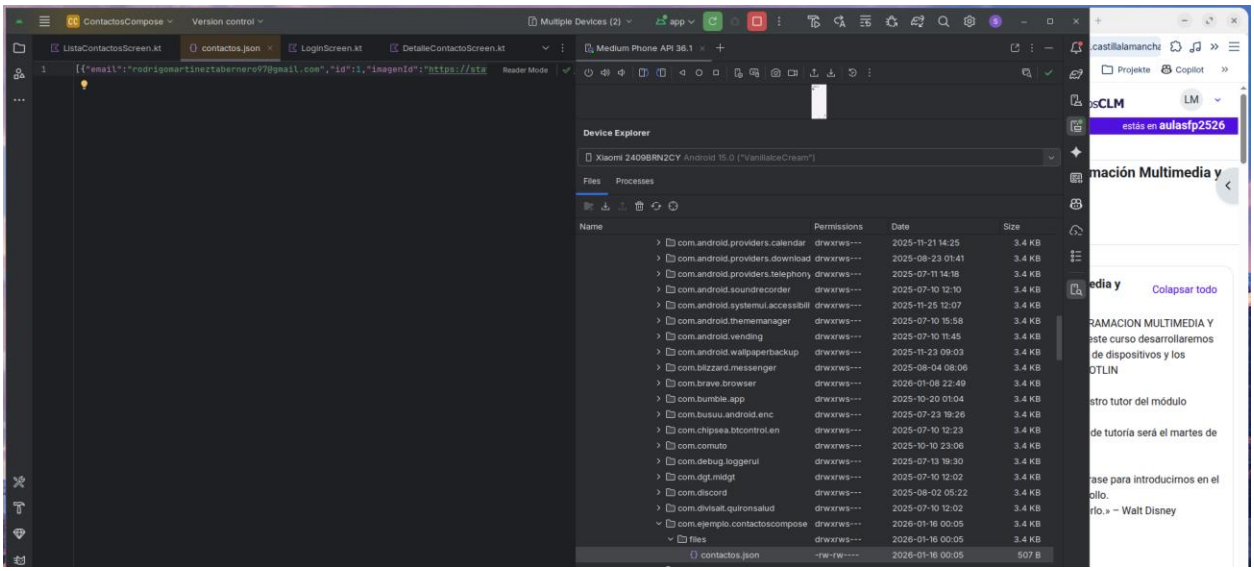

dium Phone API 36.1
Agenda de contactos
rodrigo
654987321
Jaime
6392582345
Exportar a fichero
Eliminar todos los contactos
Cerrar sesión
Brave
Projekte
Copilot
LM
estás en aulasfp2526
Colapsar todo
Walt Disney
contactos.json



3.6. Exportación a JSON

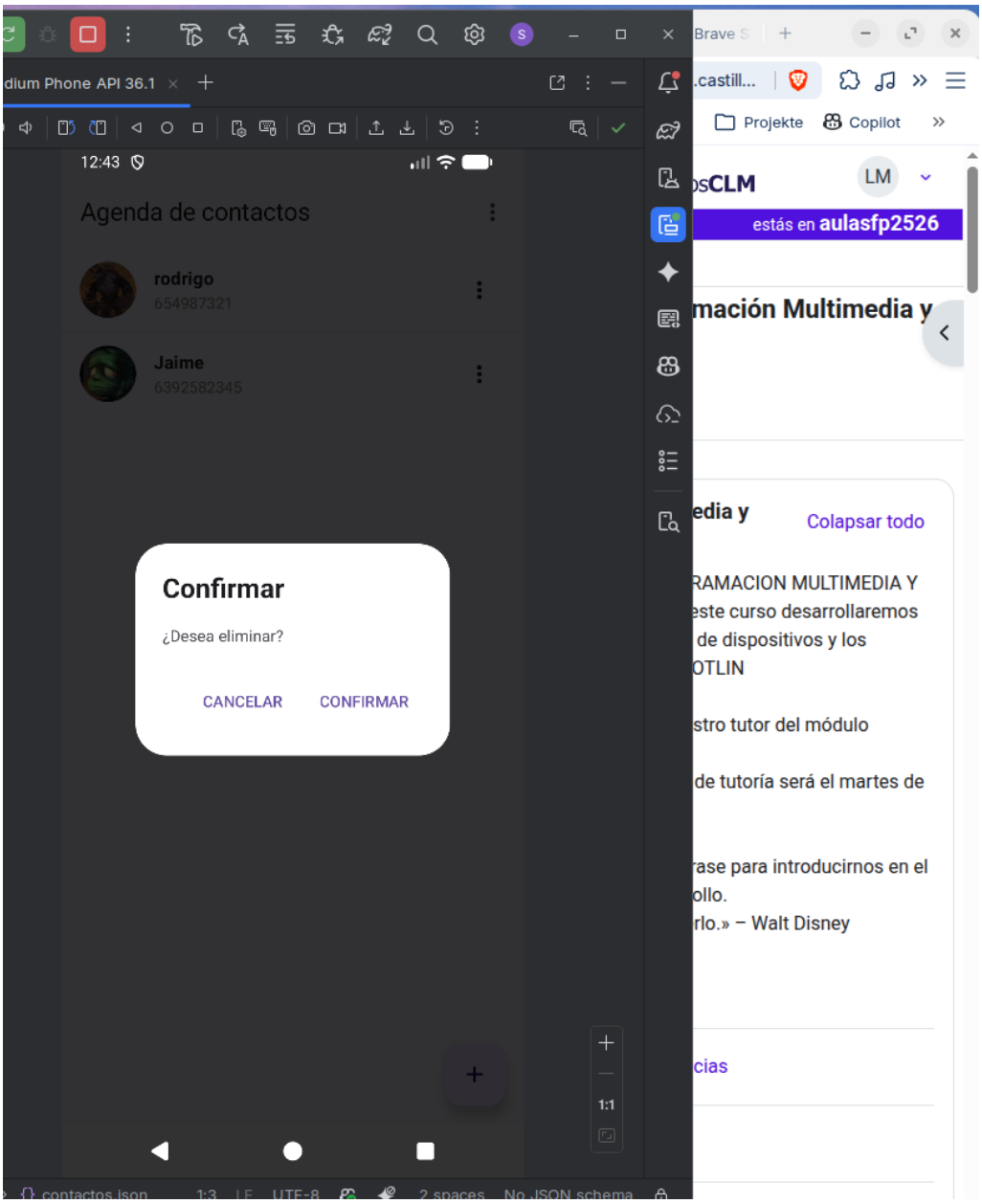

dium Phone API 36.1
12:43
Agenda de contactos
rodrigo
654987321
Jaime
6392582345
Confirmar
¿Desea exportar los contactos?
CANCELAR
CONFIRMAR
1:1
contactos.json
1:3
LF
UTF-8
2 spaces
No JSON schema
Brave
.castill...
Projekte
Copilot
osCLM
LM
estás en aulasfp2526
mación Multimedia y
edia y
Colapsar todo
RAMACION MULTIMEDIA Y
este curso desarrollaremos
de dispositivos y los
OTLIN
stro tutor del módulo
de tutoría será el martes de
rase para introducirnos en el
ollo.
rlo.» – Walt Disney
cias

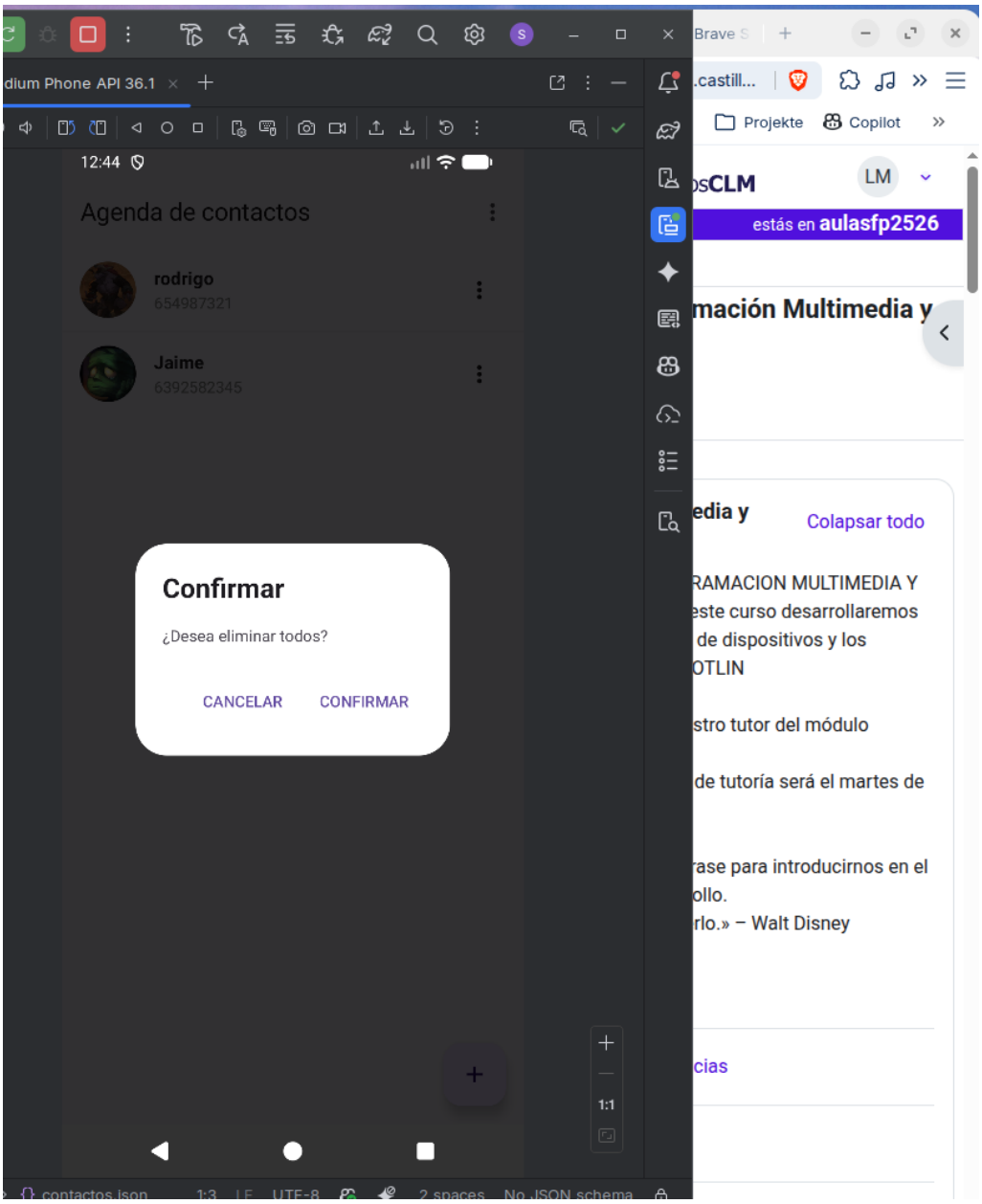

dium Phone API 36.1
Agenda de contactos
rodrigo
654987321
Jaime
6392582345
Exportado correctamente
1:1
contactos.json
1:3
LF
UTF-8
2 spaces
No JSON schema
Brave
.castill...
Projekte
Copilot
LM
estás en aulasfp2526
mación Multimedia y
edia y
Colapsar todo
RAMACION MULTIMEDIA Y
este curso desarrollaremos
de dispositivos y los
OTLIN
stro tutor del módulo
de tutoría será el martes de
rase para introducirnos en el
ollo.
rlo.» – Walt Disney
cias

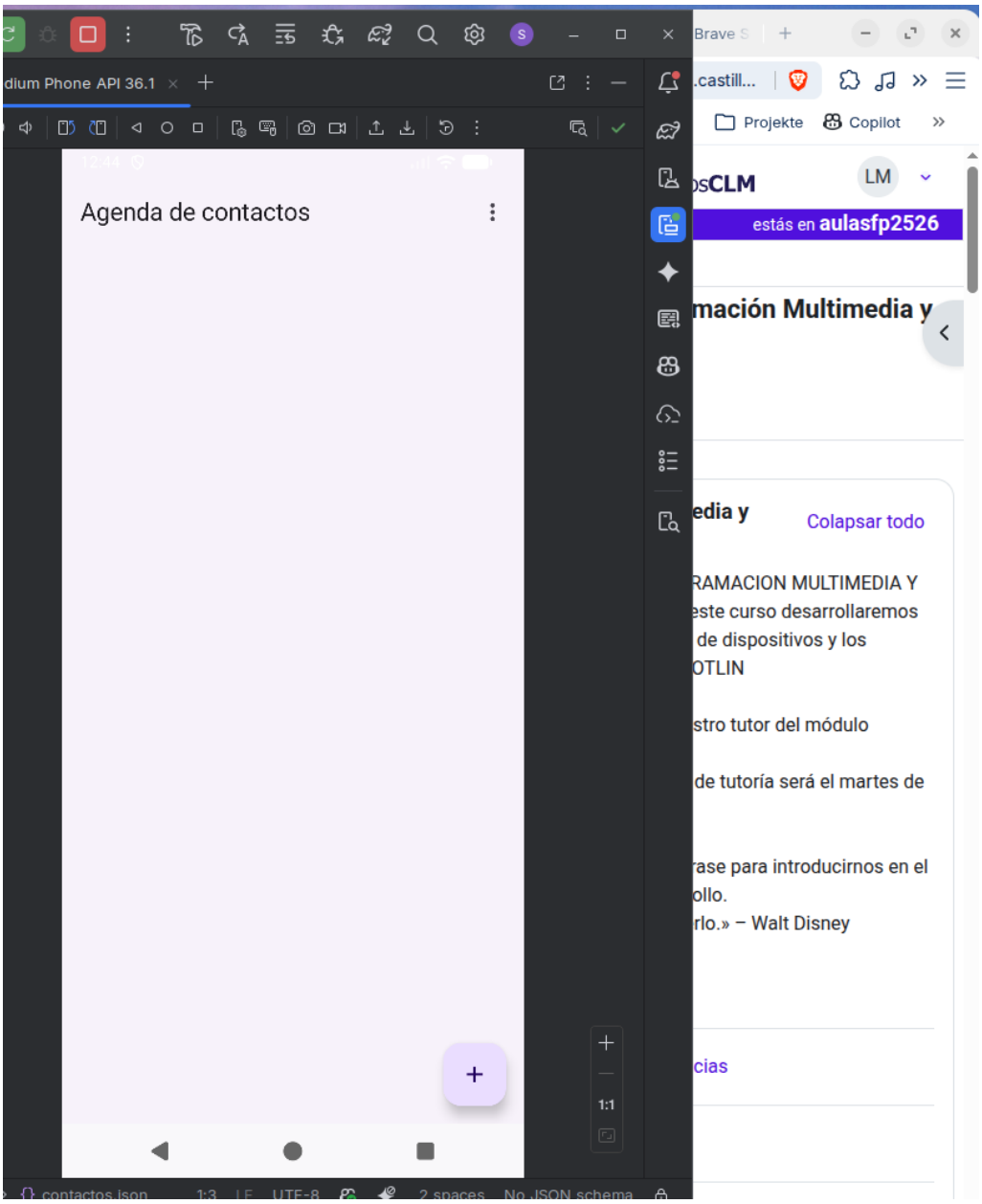



```
[{"email":"rodri
arts\"Gmail.com","id":2,"imagenId":"https://imgs.search.brave.com/DrIVsNemqNx2b0annjSrT2T6GJymKg0PfHdemw3BxrQ/rs:fit:860:0:0:0/g:ce/aHR0cHM6Ly93aWtp/LmxlYWd1ZW9mbGVn/ZW5kcy5jb20vZW4t/dXMvaW1hZ2VzL3Ro/dW1iL0FsaXN0YXJf/UmVuZGVyXzMucG5n/LzIxOHB4LUFsaXN0/YXJfUmVuZGVyXzMu/cG5nP2YwMmQ2","nombre":"rodri","telefono":"1234"},{"email":"maria\"Gmail.com","id":3,"imagenId":"https://imgs.search.brave.com/Br0u2XrUPbGgCncXIjQJaXBFAfPQ6aVCsfY9V6xP79s/rs:fit:500:0:1:0/g:ce/aHR0cHM6Ly9pLnBp/bmltZy5jb20vb3Jp/Z2luYWxzLzQ2LzZk/LzlhLzQ2NmQ5YWJl/NTRhMjdjZTFmYzlj/ZjA3Yzc1OTU3NTdl/LmpwZw","nombre":"Mara",
"telefono":"456789123"}]
```

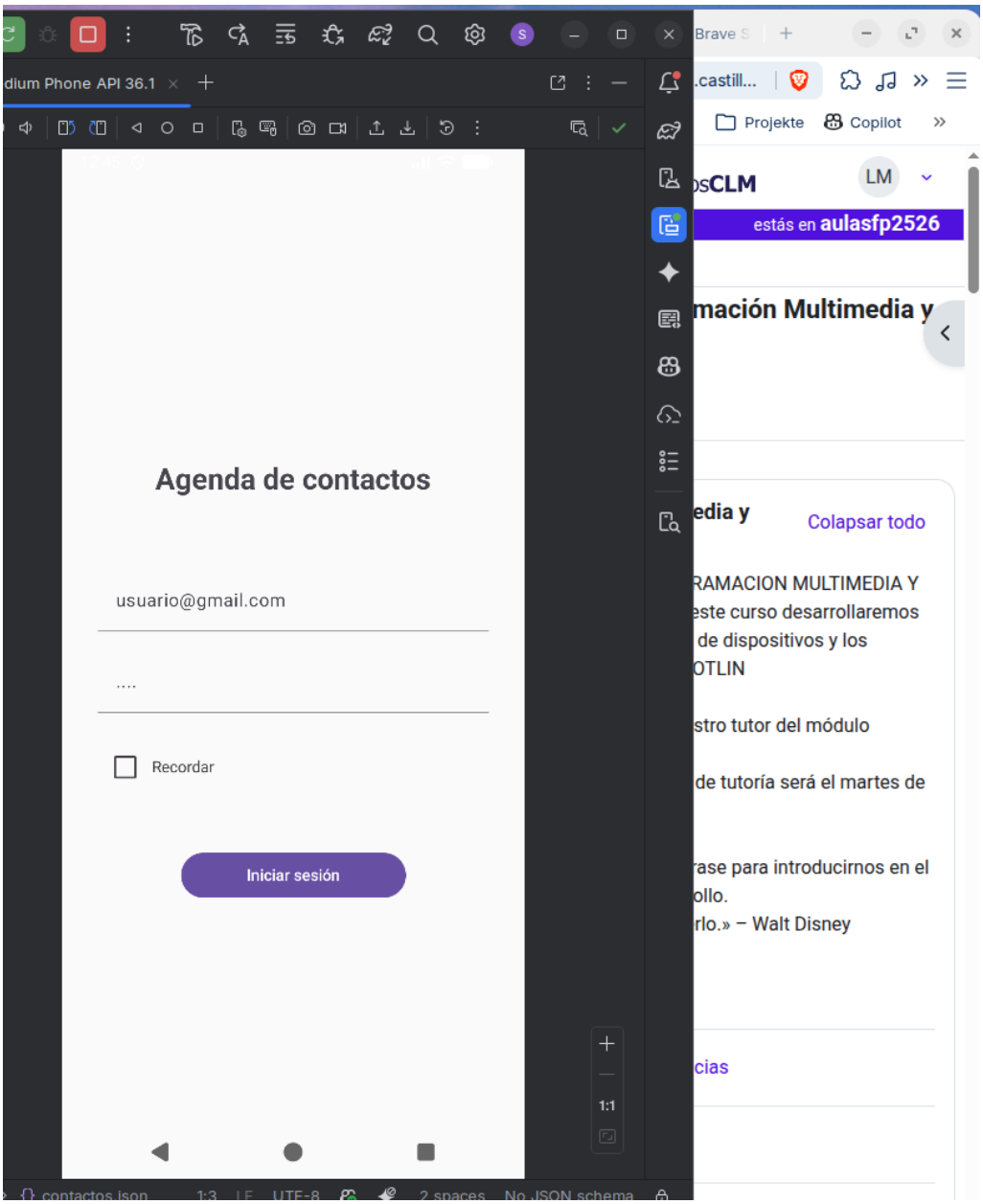

3.7 Eliminar todos los contactos/Un contacto


dium Phone API 36.1
12:43
Agenda de contactos
rodrigo
654987321
Jaime
6392582345
Confirmar
¿Desea eliminar?
CANCELAR
CONFIRMAR
1:1
contactos.json
1:3
LF
UTF-8
2 spaces
No JSON schema
Brave
.castill...
Projekte
Copilot
LM
estás en aulasfp2526
mación Multimedia y
edia y
Colapsar todo
RAMACION MULTIMEDIA Y
este curso desarrollaremos
de dispositivos y los
OTLIN
stro tutor del módulo
de tutoría será el martes de
rase para introducirnos en el
ollo.
rlo.» – Walt Disney
cias


dium Phone API 36.1
12:44
Agenda de contactos
rodrigo
654987321
Jaime
6392582345
Confirmar
¿Desea eliminar todos?
CANCELAR
CONFIRMAR
1:1
contactos.json
1:3
LF
UTF-8
2 spaces
No JSON schema
Brave
.castill...
Projekte
Copilot
LM
estás en aulasfp2526
mación Multimedia y
edia y
Colapsar todo
RAMACION MULTIMEDIA Y
este curso desarrollaremos
de dispositivos y los
OTLIN
stro tutor del módulo
de tutoría será el martes de
rase para introducirnos en el
ollo.
rlo.» – Walt Disney
cias



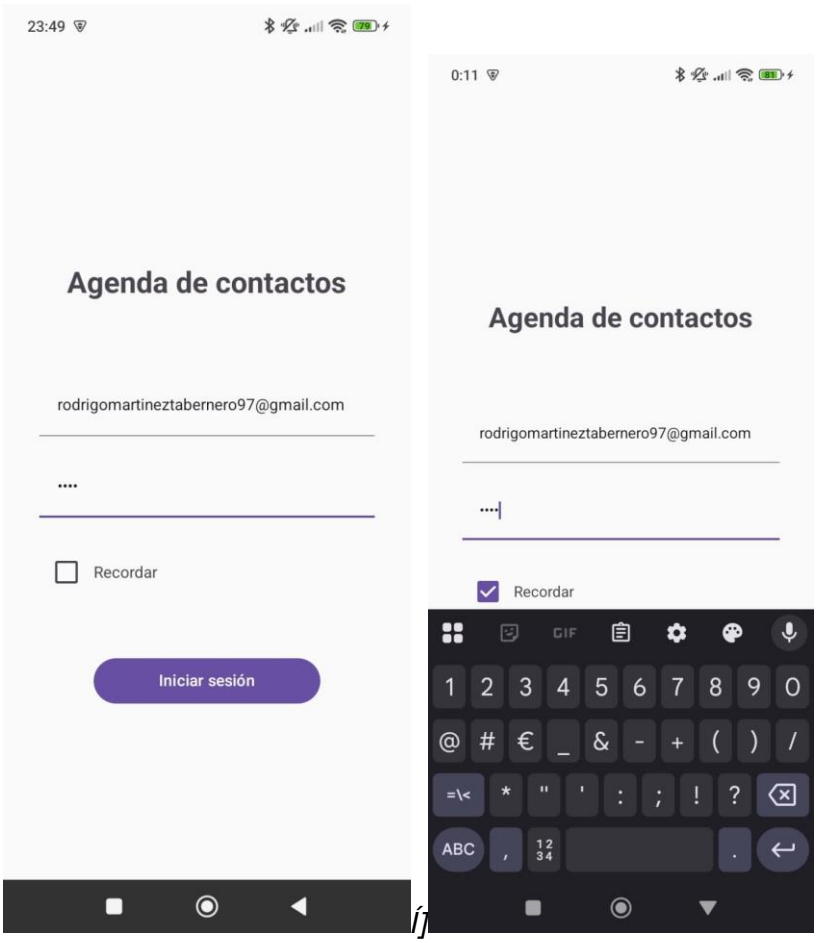
Y cerramos sesión, volvemos al inicio



DISPOSITIVO REAL (XIAOMI REDMI 14C).

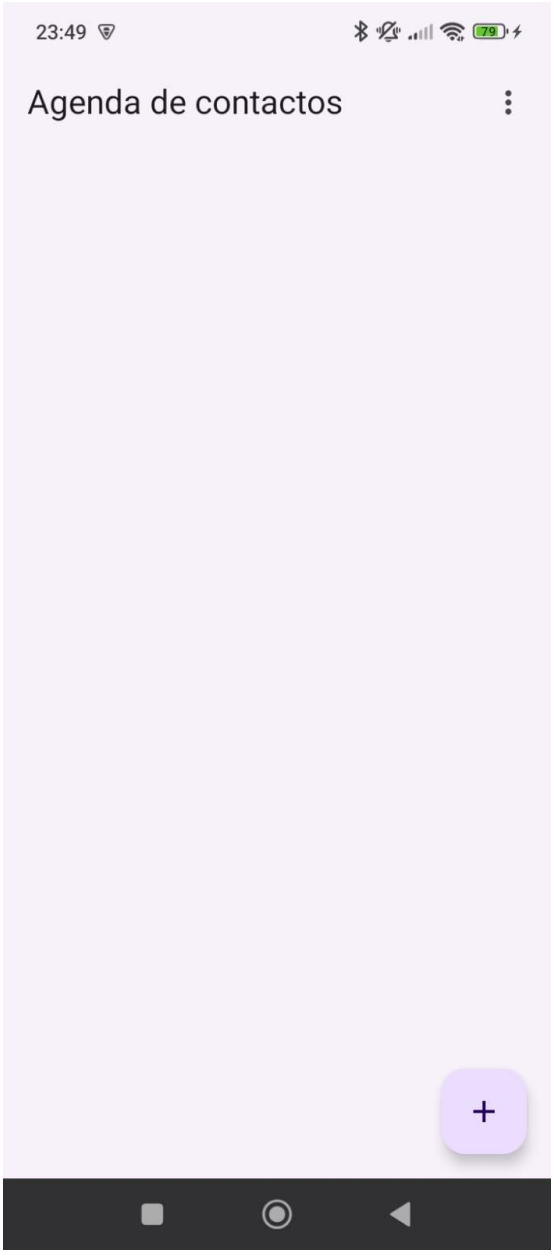
No he sabido bien como poner mi perfil de Educamos aquí para demostrar mi autoría.

3.1. Pantalla de Login : No se ve con capturas pero la funcionalidad de recordar o no, o cerrar sesión funciona correctamente



íj

3.2. Lista de Contactos (Estado inicial / Vacía)



3.3. Formulario de Alta/Edición

0:05

← Crear contacto

Nombre

Teléfono casa

Email personal

Imagen URL

Grabar

0:05

← Editar contacto

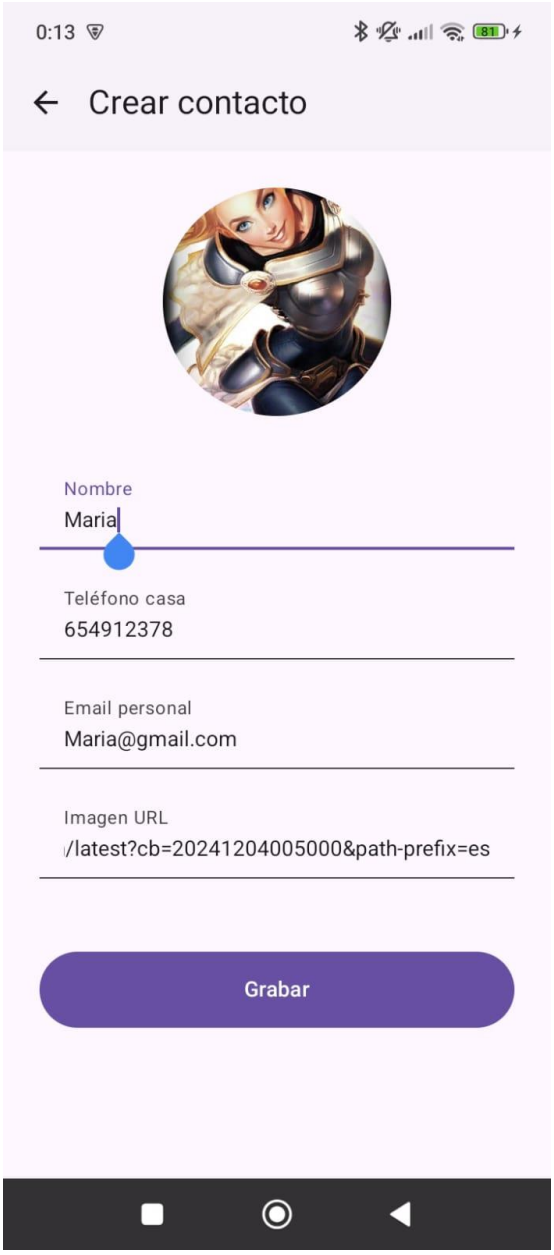
Nombre
Maria

Teléfono casa
698523147

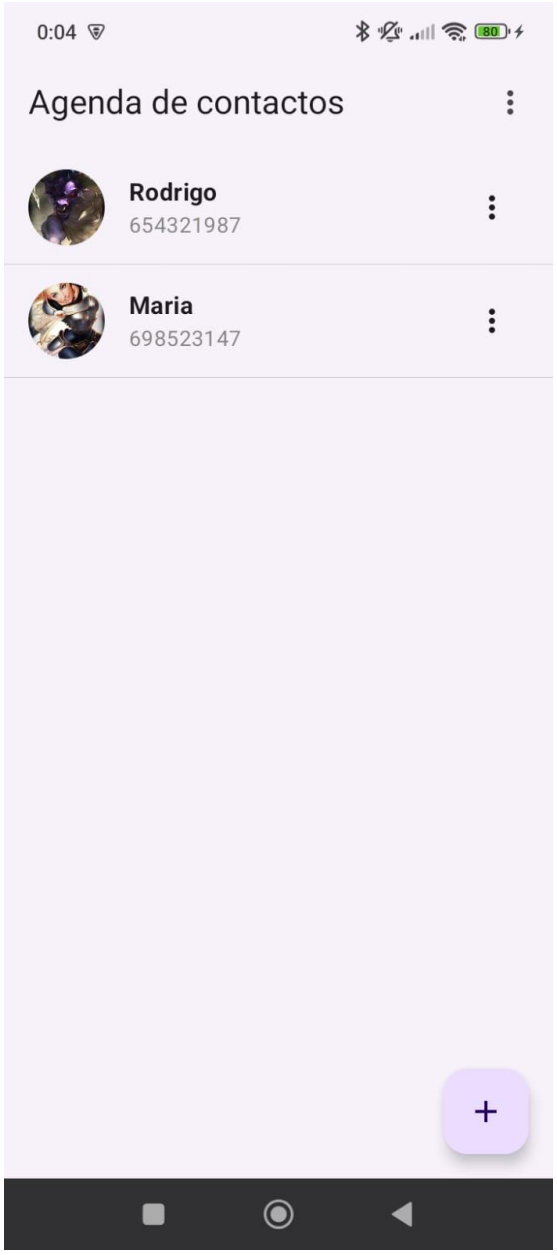
Email personal
rodrigomartineztabernero97@gmail.com

Imagen URL
https://static.wikia.nocookie.net/doblaje/ima

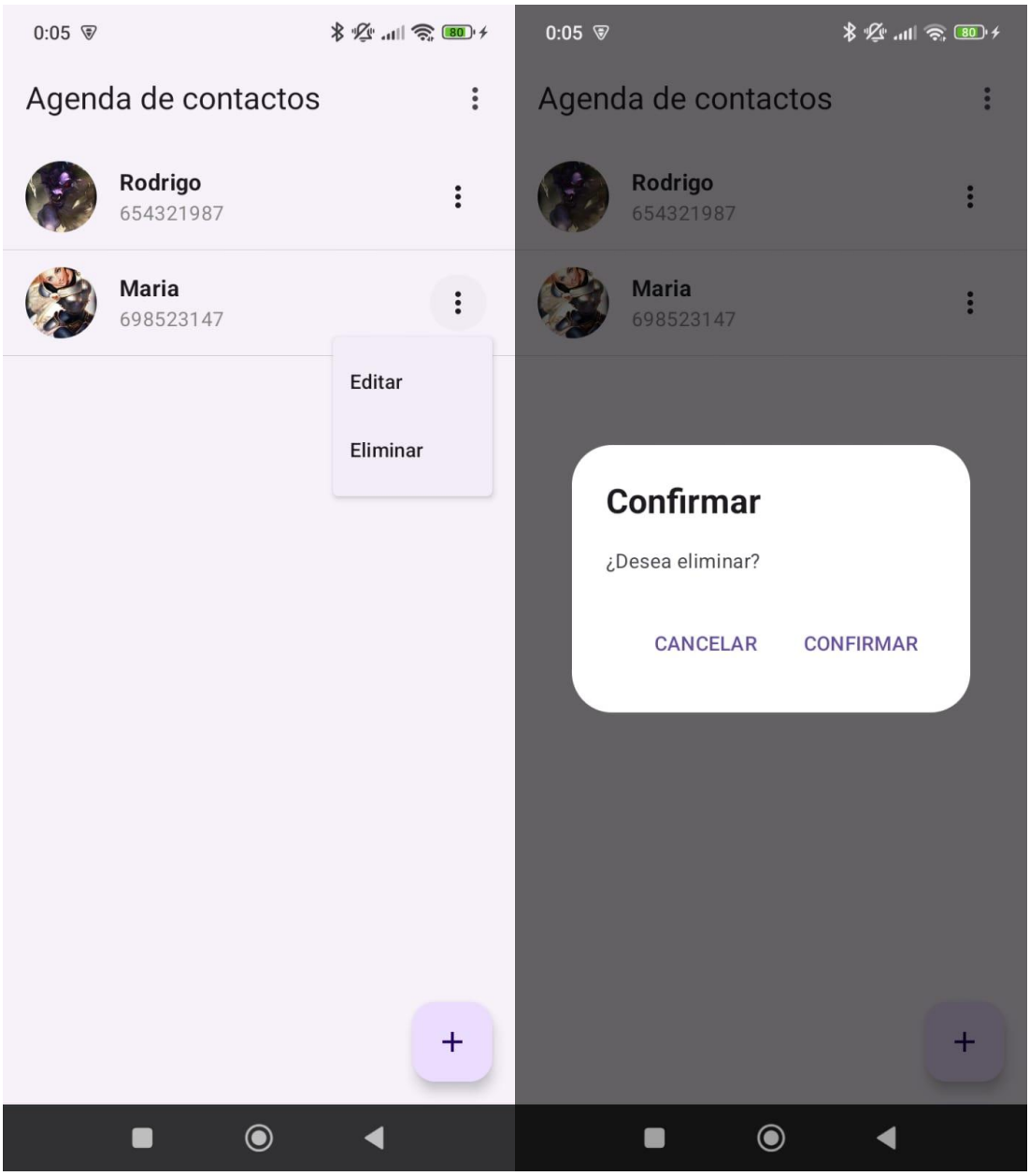
Grabar

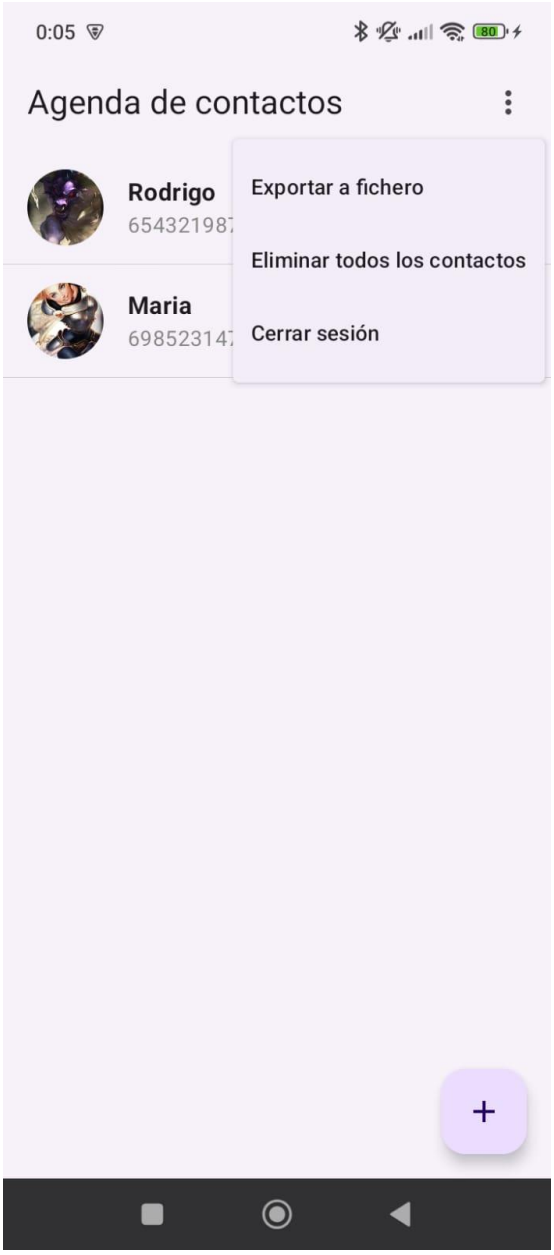


3.4. Lista de Contactos con datos

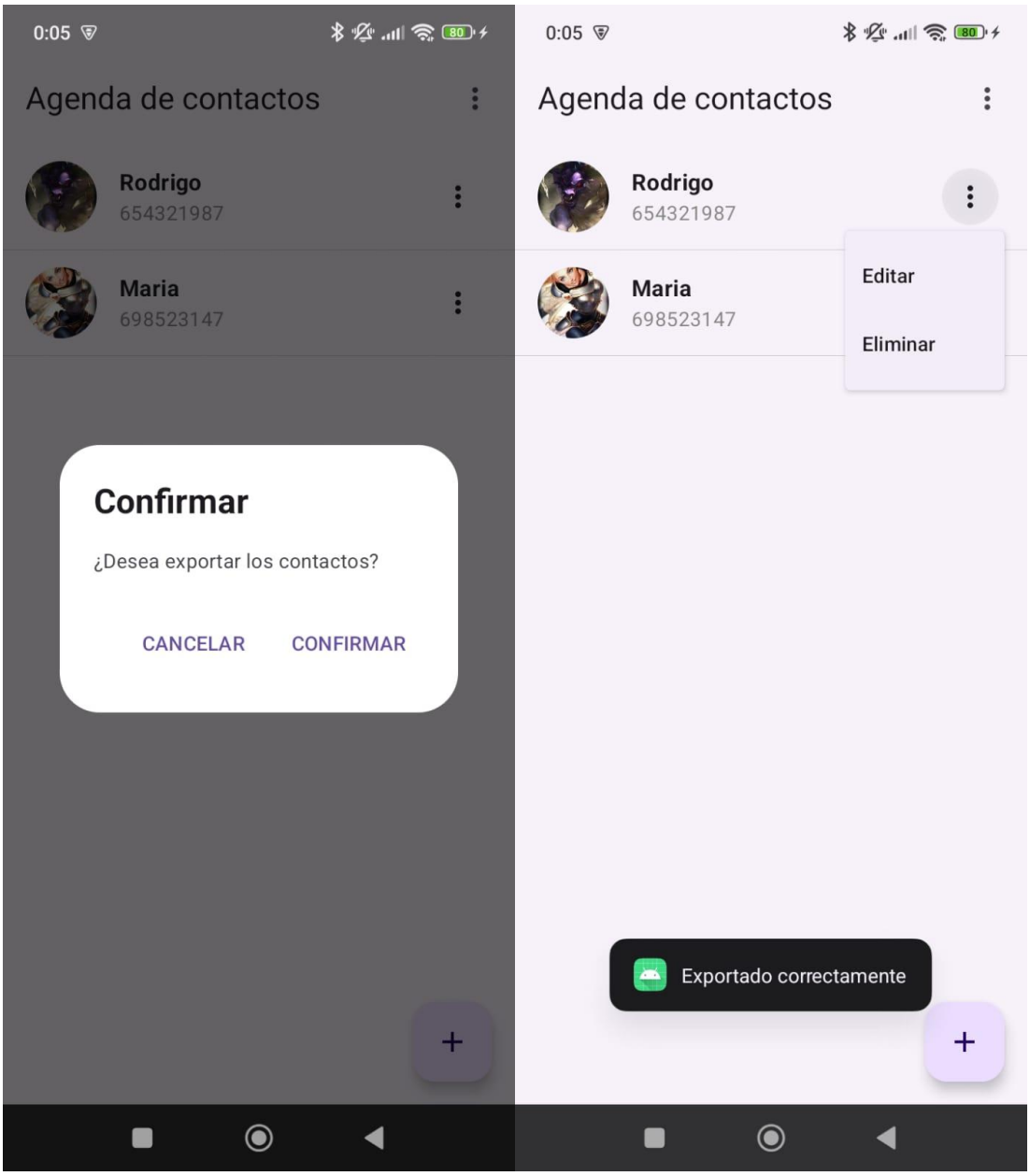


3.5. Menú Popup y Funcionalidades


0:05
Agenda de contactos
Rodrigo
654321987
Maria
698523147
Editar
Eliminar
0:05
Agenda de contactos
Rodrigo
654321987
Maria
698523147
Confirmar
¿Desea eliminar?
CANCELAR
CONFIRMAR



3.6. Exportación a JSON



3.7 Eliminar todos los contactos

4. Dificultades encontradas y soluciones

- **A la hora de encontrar el .json, no lo he podido abrir desde el propio emulador. Al final he conseguido hacerlo pulsando** Ctrl + Shift + A (Uso Linux) y escribiendo Device Explorer y accediendo a la ruta /storage/emulated/0/Android/data/com.ejemplo.contactoscompose/files/contactos.json desde Android Studio, fuera del emulador.

- **No sabía si implementar esta funcionalidad: que la lista de contactos cambie cuando hacía logins con cuentas diferentes. He acabado implementando un login 'fake', y por eso, aparece la misma lista para diferentes cuentas.**
- **Problemas de versiones de dependencias elegidas demasiado antiguas, aunque el propio Android Studio ya me sugería como actualizar y sincronizar.**
- **Me pasé hora y media de reloj intentado compilar el proyecto hasta que me di cuenta de que tenía un '>' de más en un .xml. Todo para luego rehacer el proyecto con Compose.**

5. Consideraciones al profesor

La aplicación la he probado en un emulador Pixel con API 36. Mi dispositivo real es un Xiaomi Redmi 14C.

La exportación del fichero JSON se realiza en el directorio privado de la aplicación (getExternalFilesDir)

He tenido que rehacer el proyecto porque había empezado haciéndolo con XML, pero luego me he pasado a Compose. Por lo tanto, me he pasado a Empty Activity. Después he tenido que cambiar Adapter por un loop de items, y borrado archivos .xml y menu. He cambiado el RecyclerView por LazyColumn.

6. Documentación del Código Fuente

Voy a saltarme poner los imports . Porque si no, queda un PDF enorme.

Añadimos estas dependencias en build.gradle.kts

```
implementation("com.google.code.gson:gson:2.13.2")
implementation("io.coil-kt:coil-compose:2.7.0")
implementation("androidx.navigation:navigation-compose:2.9.6")
implementation("androidx.compose.material:material-icons-extended:1.7.8")
```

Clase: AndroidManifest.xml

Hemos añadido esta línea

```
<uses-permission android:name="android.permission.INTERNET"/>
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.ContactosCompose">
        <activity
            android:name=".MainActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:theme="@style/Theme.ContactosCompose">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Clase: MainActivity.kt

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            AppNavigation()
        }
```

```
    }
}

@Composable
fun AppNavigation() {
    val navController = rememberNavController()
    val context = LocalContext.current

    // Comprobamos sesión guardada en SharedPreferences
    val prefs = context.getSharedPreferences("preferencias_file", Context.MODE_PRIVATE)
    val isLogueado = prefs.getBoolean("logueado", false)
    val startDestination = if (isLogueado) "lista" else "login"

    // Definimos las rutas
    NavHost(navController = navController, startDestination = startDestination) {

        composable("login") {
            LoginScreen(navController)
        }

        composable("lista") {
            ListaContactosScreen(navController)
        }

        composable(
            route = "detalle/{id}",
            arguments = listOf(navArgument("id") { type = NavType.IntType })
        ) { backStackEntry ->
            val id = backStackEntry.arguments?.getInt("id") ?: 0
            DetalleContactoScreen(navController, id)
        }
    }
}
```

Clase: Contacto.kt (Modelo de datos)

```
package com.ejemplo.contactoscompose.model

data class Contacto(
    var id: Int = 0,
    var nombre: String,
    var telefono: String,
    var email: String? = null,
    var imagenId: String? = null
)
```

Clase: ContactosDB.kt(Base de datos)

```
class ContactosDB(context: Context) : SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {

  companion object {
    private const val DATABASE_VERSION = 1
    private const val DATABASE_NAME = "contactos.db"

    private const val TABLA_CONTACTOS = "contactos"
    private const val KEY_ID = "id"
    private const val KEY_NOMBRE = "nombre"
    private const val KEY_TELEFONO = "telefono"
    private const val KEY_EMAIL = "email"
    private const val KEY_IMAGEN_ID = "imagenid"
  }

  override fun onCreate(db: SQLiteDatabase) {
    val createTable = ("CREATE TABLE " + TABLA_CONTACTOS + "("
          + KEY_ID + " INTEGER PRIMARY KEY AUTOINCREMENT,"
          + KEY_NOMBRE + " TEXT NOT NULL,"
          + KEY_TELEFONO + " TEXT NOT NULL,"
          + KEY_EMAIL + " TEXT,"
          + KEY_IMAGEN_ID + " TEXT" + ")")
    db.execSQL(createTable)
  }

  override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
    db.execSQL("DROP TABLE IF EXISTS $TABLA_CONTACTOS")
    onCreate(db)
  }

  // Métodos CRUD:
  fun insertar(contacto: Contacto): Long {
    val db = this.writableDatabase
    val values = ContentValues()
    values.put(KEY_NOMBRE, contacto.nombre)
    values.put(KEY_TELEFONO, contacto.telefono)
    values.put(KEY_EMAIL, contacto.email)
    values.put(KEY_IMAGEN_ID, contacto.imagenId)

    val id = db.insert(TABLA_CONTACTOS, null, values)
    db.close()
    return id
  }

  // Leer todos los contactos
  fun leer(): ArrayList<Contacto> {
    val listaContactos = ArrayList<Contacto>()
    val selectQuery = "SELECT * FROM $TABLA_CONTACTOS"
```

```
    val db = this.readableDatabase
    val cursor: Cursor?

    try {
        cursor = db.rawQuery(selectQuery, null)
    } catch (e: Exception) {
        db.execSQL(selectQuery)
        return ArrayList()
    }

    var id: Int
    var nombre: String
    var telefono: String
    var email: String?
    var imagenId: String?

    if (cursor.moveToFirst()) {
        do {
            id = cursor.getInt(cursor.getColumnIndexOrThrow(KEY_ID))
            nombre = cursor.getString(cursor.getColumnIndexOrThrow(KEY_NOMBRE))
            telefono = cursor.getString(cursor.getColumnIndexOrThrow(KEY_TELEFONO))
            email = cursor.getString(cursor.getColumnIndexOrThrow(KEY_EMAIL))
            imagenId = cursor.getString(cursor.getColumnIndexOrThrow(KEY_IMAGEN_ID))

            val contacto = Contacto(id, nombre, telefono, email, imagenId)
            listaContactos.add(contacto)
        } while (cursor.moveToNext())
    }
    cursor.close()
    db.close()
    return listaContactos
}

// Lee un contacto por ID
fun leer(id: Int): Contacto? {
    val db = this.readableDatabase
    val cursor = db.query(
        TABLA_CONTACTOS,
        arrayOf(KEY_ID, KEY_NOMBRE, KEY_TELEFONO, KEY_EMAIL, KEY_IMAGEN_ID),
        "$KEY_ID=?",
        arrayOf(id.toString()),
        null, null, null, null
    )

    var contacto: Contacto? = null
    if (cursor != null && cursor.moveToFirst()) {
        contacto = Contacto(
            cursor.getInt(0),
```

```
            cursor.getString(1),
            cursor.getString(2),
            cursor.getString(3),
            cursor.getString(4)
        )
        cursor.close()
    }
    db.close()
    return contacto
}

// Actualiza un contacto por ID
fun actualizar(contacto: Contacto): Int {
    val db = this.writableDatabase
    val values = ContentValues()
    values.put(KEY_NOMBRE, contacto.nombre)
    values.put(KEY_TELEFONO, contacto.telefono)
    values.put(KEY_EMAIL, contacto.email)
    values.put(KEY_IMAGEN_ID, contacto.imagenId)

    // Actualizamos la fila donde el ID coincida
    val success = db.update(TABLA_CONTACTOS, values, "$KEY_ID=?", arrayOf(contacto.id.toString()))
    db.close()
    return success
}

// Elimina un contacto por ID
fun eliminar(id: Int): Int {
    val db = this.writableDatabase
    val success = db.delete(TABLA_CONTACTOS, "$KEY_ID=?", arrayOf(id.toString()))
    db.close()
    return success
}

// Elimina TODOS los contactos
fun eliminarTodos(): Int {
    val db = this.writableDatabase
    val success = db.delete(TABLA_CONTACTOS, null, null)
    db.close()
    return success
}
```

Clase: LoginScreen.kt (Pantalla 1)

```
@Composable
fun LoginScreen(navController: NavController) {
```

```
val context = LocalContext.current
//Usamos MODE_PRIVATE para que solo nuestra app pueda acceder a estas preferencias
val sharedPreferences = context.getSharedPreferences("preferencias_file", Context.MODE_PRIVATE)

var email by remember { mutableStateOf("") }
var password by remember { mutableStateOf("") }
var recordarDatos by remember { mutableStateOf(false) }

val PurpleButton = Color(0xFF6750A4)

LaunchedEffect(Unit) {
    val recordar = sharedPreferences.getBoolean("recordar_datos", false)
    if (recordar) {
        email = sharedPreferences.getString("email", "") ?: ""
        password = sharedPreferences.getString("password", "") ?: ""
        recordarDatos = true
    }
}

Column(
    modifier = Modifier
        .fillMaxSize()
        .padding(32.dp),
    horizontalAlignment = Alignment.CenterHorizontally,
    verticalArrangement = Arrangement.Center
) {

    Spacer(modifier = Modifier.height(32.dp))

    Text(
        text = "Agenda de contactos",
        fontSize = 26.sp,
        fontWeight = FontWeight.Bold,
        color = Color(0xFF49454F)
    )

    Spacer(modifier = Modifier.height(64.dp))

    // Campo Email
    TextField(
        value = email,
        onValueChange = { email = it },
        placeholder = { Text("usuario@gmail.com") },
        modifier = Modifier.fillMaxWidth(),
        colors = TextFieldDefaults.colors(
            unfocusedContainerColor = Color.Transparent,
            focusedContainerColor = Color.Transparent,
```

```
            focusedIndicatorColor = PurpleButton,
            unfocusedIndicatorColor = Color.Gray
        )
    )

    Spacer(modifier = Modifier.height(16.dp))

    // Campo Password
    TextField(
        value = password,
        onValueChange = { password = it },
        placeholder = { Text("....") },
        visualTransformation = PasswordVisualTransformation(),
        modifier = Modifier.fillMaxWidth(),
        colors = TextFieldDefaults.colors(
            unfocusedContainerColor = Color.Transparent,
            focusedContainerColor = Color.Transparent,
            focusedIndicatorColor = PurpleButton,
            unfocusedIndicatorColor = Color.Gray
        )
    )

    Spacer(modifier = Modifier.height(24.dp))

    // Checkbox
    Row(verticalAlignment = Alignment.CenterVertically, modifier = Modifier.fillMaxWidth()) {
        Checkbox(
            checked = recordarDatos,
            onCheckedChange = { recordarDatos = it },
            colors = CheckboxDefaults.colors(checkedColor = PurpleButton)
        )
        Text(text = "Recordar", color = Color.DarkGray)
    }

    Spacer(modifier = Modifier.height(48.dp))

    Button(
        onClick = {
            if (email.isNotEmpty() && password.isNotEmpty()) {
                val editor = sharedPreferences.edit()
                editor.putBoolean("logueado", true)
                if (recordarDatos) {
                    editor.putBoolean("recordar_datos", true)
                    editor.putString("email", email)
                    editor.putString("password", password)
                } else {
                    editor.putBoolean("recordar_datos", false)
                    editor.remove("email")
```

```
                editor.remove("password")
            }
            editor.apply()
            navController.navigate("lista") { popUpTo("login") { inclusive = true } }
        } else {
            Toast.makeText(context, "Rellena los campos", Toast.LENGTH_SHORT).show()
        }
    },
    modifier = Modifier.width(200.dp),
    colors = ButtonDefaults.buttonColors(containerColor = PurpleButton),
    shape = RoundedCornerShape(50)
) {
    Text("Iniciar sesión")
}
}
}
```

Clase: ListaContactosScreen.kt (Pantalla 2)

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun ListaContactosScreen(navController: NavController) {
    val context = LocalContext.current
    val db = remember { ContactosDB(context) }

    // Estados de datos
    var listaContactos by remember { mutableStateOf<List<Contacto>>(emptyList()) }

    // Estados de UI (Menús y Diálogos)
    var showMenu by remember { mutableStateOf(false) }

    // Diálogos de confirmación
    var showDialogExportar by remember { mutableStateOf(false) }
    var showDialogBorrarTodo by remember { mutableStateOf(false) }
    var showDialogBorrarUno by remember { mutableStateOf(false) }

    // Variable temporal para saber qué contacto borrar individualmente
    var contactoParaBorrar by remember { mutableStateOf<Contacto?>(null) }

    val FabContainerColor = Color(0xFFEADDFF)
    val FabContentColor = Color(0xFF21005D)
    val PurpleText = Color(0xFF6750A4)

    // Refrescar lista al volver (OnResume)
    val lifecycleOwner = LocalLifecycleOwner.current
```

```
DisposableEffect(lifecycleOwner) {
    val observer = LifecycleEventObserver { _, event ->
        if (event == Lifecycle.Event.ON_RESUME) {
            listaContactos = db.leer()
        }
    }
    lifecycleOwner.lifecycle.addObserver(observer)
    onDispose { lifecycleOwner.lifecycle.removeObserver(observer) }
}

Scaffold(
    containerColor = Color(0xFFF7F2FA),
    topBar = {
        TopAppBar(
            title = { Text("Agenda de contactos") },
            colors = TopAppBarDefaults.topAppBarColors(containerColor = Color(0xFFF7F2FA)),
            actions = {
                IconButton(onClick = { showMenu = true }) {
                    Icon(Icons.Default.MoreVert, contentDescription = "Opciones")
                }
                DropdownMenu(
                    expanded = showMenu,
                    onDismissRequest = { showMenu = false }
                ) {
                    DropdownMenuItem(
                        text = { Text("Exportar a fichero") },
                        onClick = {
                            showMenu = false
                            // Ahora abre el diálogo en vez de exportar directo
                            showDialogExportar = true
                        }
                    )
                    DropdownMenuItem(
                        text = { Text("Eliminar todos los contactos") },
                        onClick = {
                            showMenu = false
                            showDialogBorrarTodo = true
                        }
                    )
                    DropdownMenuItem(
                        text = { Text("Cerrar sesión") },
                        onClick = {
                            showMenu = false
                            cerrarSesion(context, navController)
                        }
                    )
                }
            }
        )
```

```
    },
    floatingActionButton = {
        FloatingActionButton(
            onClick = { navController.navigate("detalle/0") },
            containerColor = FabContainerColor,
            contentColor = FabContentColor,
            shape = RoundedCornerShape(16.dp)
        ) {
            Icon(Icons.Default.Add, contentDescription = "Añadir")
        }
    }
) { padding ->

    LazyColumn(
        modifier = Modifier.padding(padding),
        contentPadding = PaddingValues(0.dp)
    ) {
        items(listaContactos) { contacto ->
            ItemContacto(
                contacto = contacto,
                onEdit = { navController.navigate("detalle/${contacto.id}") },
                onDelete = {
                    // Guardamos el contacto a borrar y abrimos diálogo
                    contactoParaBorrar = contacto
                    showDialogBorrarUno = true
                }
            )
            HorizontalDivider(color = Color.LightGray, thickness = 0.5.dp)
        }
    }
}

// Modales de confirmación

if (showDialogExportar) {
    AlertDialog(
        onDismissRequest = { showDialogExportar = false },
        title = { Text("Confirmar", fontWeight = FontWeight.Bold) },
        text = { Text("¿Desea exportar los contactos?") },
        confirmButton = {
            TextButton(onClick = {
                exportarJSON(context, listaContactos)
                showDialogExportar = false
            }) { Text("CONFIRMAR", color = PurpleText) }
        },
        dismissButton = {
            TextButton(onClick = { showDialogExportar = false }) { Text("CANCELAR", color = PurpleText) }
        },
        containerColor = Color.White
```

```
    )
  }

  if (showDialogBorrarTodo) {
    AlertDialog(
      onDismissRequest = { showDialogBorrarTodo = false },
      title = { Text("Confirmar", fontWeight = FontWeight.Bold) },
      text = { Text("¿Desea eliminar todos?") },
      confirmButton = {
        TextButton(onClick = {
          db.eliminarTodos()
          listaContactos = db.leer()
          showDialogBorrarTodo = false
        }) { Text("CONFIRMAR", color = PurpleText) }
      },
      dismissButton = {
        TextButton(onClick = { showDialogBorrarTodo = false }) { Text("CANCELAR", color = PurpleText) }
      },
      containerColor = Color.White
    )
  }

  if (showDialogBorrarUno && contactoParaBorrar != null) {
    AlertDialog(
      onDismissRequest = { showDialogBorrarUno = false },
      title = { Text("Confirmar", fontWeight = FontWeight.Bold) },
      text = { Text("¿Desea eliminar?") },
      confirmButton = {
        TextButton(onClick = {
          contactoParaBorrar?.let {
            db.eliminar(it.id)
            listaContactos = db.leer()
            Toast.makeText(context, "Contacto eliminado", Toast.LENGTH_SHORT).show()
          }
          showDialogBorrarUno = false
          contactoParaBorrar = null
        }) { Text("CONFIRMAR", color = PurpleText) }
      },
      dismissButton = {
        TextButton(onClick = {
          showDialogBorrarUno = false
          contactoParaBorrar = null
        }) { Text("CANCELAR", color = PurpleText) }
      },
      containerColor = Color.White
    )
  }
}
```

```
@Composable
fun ItemContacto(contacto: Contacto, onEdit: () -> Unit, onDelete: () -> Unit) {
    var showContextMenu by remember { mutableStateOf(false) }

    Row(
        modifier = Modifier
            .fillMaxWidth()
            .padding(horizontal = 16.dp, vertical = 12.dp),
        verticalAlignment = Alignment.CenterVertically
    ) {
        AsyncImage(
            model = ImageRequest.Builder(LocalContext.current)
                .data(if (contacto.imagenId.isNullOrEmpty()) R.drawable.ic_placeholder_camera else contacto.imagenId)
                .crossfade(true)
                .build(),
            contentDescription = "Foto",
            contentScale = ContentScale.Crop,
            placeholder = painterResource(R.drawable.ic_placeholder_camera),
            error = painterResource(R.drawable.ic_placeholder_camera),
            modifier = Modifier
                .size(50.dp)
                .clip(CircleShape)
        )

        Spacer(modifier = Modifier.width(16.dp))

        Column(modifier = Modifier.weight(1f)) {
            Text(text = contacto.nombre, fontWeight = FontWeight.Bold, style = MaterialTheme.typography.titleMedium)
            Text(text = contacto.telefono, style = MaterialTheme.typography.bodyMedium, color = Color.Gray)
        }

        Box {
            IconButton(onClick = { showContextMenu = true }) {
                Icon(Icons.Default.MoreVert, contentDescription = "Opciones item")
            }
            DropdownMenu(
                expanded = showContextMenu,
                onDismissRequest = { showContextMenu = false }
            ) {
                DropdownMenuItem(
                    text = { Text("Editar") },
                    onClick = {
                        showContextMenu = false
                        onEdit()
                    }
                )
                DropdownMenuItem(
                    text = { Text("Eliminar") },
```

```
                onClick = {
                    showContextMenu = false
                    onDelete()
                }
            )
        }
    }
  }
}

// FUNCIONES AUXILIARES
fun exportarJSON(context: Context, lista: List<Contacto>) {
  try {
    val gson = Gson()
    val jsonString = gson.toJson(lista)
    val archivo = File(context.getExternalFilesDir(null), "contactos.json")
    archivo.writeText(jsonString)
    Toast.makeText(context, "Exportado correctamente", Toast.LENGTH_SHORT).show()
  } catch (e: Exception) {
    Toast.makeText(context, "Error: ${e.message}", Toast.LENGTH_SHORT).show()
  }
}

fun cerrarSesion(context: Context, navController: NavController) {
  val prefs = context.getSharedPreferences("preferencias_file", Context.MODE_PRIVATE)
  prefs.edit().putBoolean("logueado", false).apply()
  navController.navigate("login") {
    popUpTo("lista") { inclusive = true }
  }
}
```

Clase: DetalleContactoScreen.kt (Popup)

```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun DetalleContactoScreen(navController: NavController, id: Int) {
  val context = LocalContext.current
  val db = remember { ContactosDB(context) }

  var nombre by remember { mutableStateOf("") }
  var telefono by remember { mutableStateOf("") }
  var email by remember { mutableStateOf("") }
  var imagenUrl by remember { mutableStateOf("") }

  val titulo = if (id == 0) "Crear contacto" else "Editar contacto"
  val PurpleButton = Color(0xFF6750A4)
```

```
LaunchedEffect(id) {
    if (id != 0) {
        val contacto = db.leer(id)
        if (contacto != null) {
            nombre = contacto.nombre
            telefono = contacto.telefono
            email = contacto.email ?: ""
            imagenUrl = contacto.imagenId ?: ""
        }
    }
}

Scaffold(
    topBar = {
        TopAppBar(
            title = { Text(titulo) },
            navigationIcon = {
                IconButton(onClick = { navController.popBackStack() }) {
                    Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription = "Volver")
                }
            },
            colors = TopAppBarDefaults.topAppBarColors(containerColor = Color(0xFFF7F2FA))
        )
    }
) { padding ->
    Column(
        modifier = Modifier
            .padding(padding)
            .padding(24.dp)
            .fillMaxSize()
            .verticalScroll(rememberScrollState()),
        horizontalAlignment = Alignment.CenterHorizontally
    ) {

        // IMAGEN GRANDE (Placeholder o URL)
        if (imagenUrl.isBlank()) {
            Image(
                painter = painterResource(id = R.drawable.ic_placeholder_camera),
                contentDescription = "Foto por defecto",
                modifier = Modifier.size(150.dp),
                alpha = 0.5f
            )
        } else {
            AsyncImage(
                model = ImageRequest.Builder(LocalContext.current)
                    .data(imagenUrl)
                    .crossfade(true)
                    .build(),
```

```
            contentDescription = "Foto Contacto",
            contentScale = ContentScale.Crop,
            placeholder = painterResource(id = R.drawable.ic_placeholder_camera),
            error = painterResource(id = R.drawable.ic_placeholder_camera),
            modifier = Modifier
                .size(150.dp)
                .clip(CircleShape)
        )
    }

    Spacer(modifier = Modifier.height(32.dp))

    // TextFields Estilo Línea Inferior
    val colorsTextField = TextFieldDefaults.colors(
        unfocusedContainerColor = Color.Transparent,
        focusedContainerColor = Color.Transparent,
        focusedIndicatorColor = PurpleButton
    )

    TextField(
        value = nombre,
        onValueChange = { nombre = it },
        label = { Text("Nombre") }, // Etiqueta flotante
        modifier = Modifier.fillMaxWidth(),
        colors = colorsTextField,
        singleLine = true
    )

    Spacer(modifier = Modifier.height(16.dp))

    TextField(
        value = telefono,
        onValueChange = { telefono = it },
        label = { Text("Teléfono casa") },
        modifier = Modifier.fillMaxWidth(),
        colors = colorsTextField,
        keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Phone),
        singleLine = true
    )

    Spacer(modifier = Modifier.height(16.dp))

    TextField(
        value = email,
        onValueChange = { email = it },
        label = { Text("Email personal") },
        modifier = Modifier.fillMaxWidth(),
```

```
            colors = colorsTextField,
            keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Email),
            singleLine = true
        )

        Spacer(modifier = Modifier.height(16.dp))

        TextField(
            value = imagenUrl,
            onValueChange = { imagenUrl = it },
            label = { Text("Imagen URL") },
            modifier = Modifier.fillMaxWidth(),
            colors = colorsTextField,
            singleLine = true
        )

        Spacer(modifier = Modifier.height(48.dp))

        // BOTÓN GRABAR
        Button(
            onClick = {
                if (nombre.isBlank() || telefono.isBlank()) {
                    Toast.makeText(context, "Nombre y Teléfono son obligatorios", Toast.LENGTH_SHORT).show()
                } else {
                    val contacto = Contacto(
                        id = id,
                        nombre = nombre,
                        telefono = telefono,
                        email = email,
                        imagenId = imagenUrl
                    )
                    if (id == 0) {
                        db.insertar(contacto)
                        Toast.makeText(context, "Contacto creado", Toast.LENGTH_SHORT).show()
                    } else {
                        db.actualizar(contacto)
                        Toast.makeText(context, "Contacto actualizado", Toast.LENGTH_SHORT).show()
                    }
                    navController.popBackStack()
                }
            },
            modifier = Modifier.fillMaxWidth().height(50.dp),
            colors = ButtonDefaults.buttonColors(containerColor = PurpleButton),
            shape = RoundedCornerShape(50)
        ) {
            Text("Grabar")
        }
    }
```

```
  }
}
```