

# RUMFIT

## APP FITNESS PARA ANDROID

RUMEN ORACHEV ORACHEV 2ºDAM





# ÍNDICE

1. TECNOLOGÍAS UTILIZADAS
2. ARQUITECTURA DEL PROYECTO
3. BASE DE DATOS (ROOM DATABASE)
4. ENTIDADES (MODELS)
5. DAO'S: ACCESO A LOS DATOS
6. REGISTRO DE USUARIO
7. INICIO DE SESIÓN (LOGIN)
8. PERFIL DEL USUARIO
9. CREAR RUTINA
10. LISTA DE RUTINAS (RECYCLERVIEW)
11. SELECCIONAR RUTINA DEL DÍA
12. AÑADIR EJERCICIO
13. EDITAR Y ELIMINAR EJERCICIO
14. REGISTRO DE PROGRESO
15. GRÁFICA (MPANDROIDCHART)
16. CONCLUSIÓN

## TECNOLOGÍAS UTILIZADAS

- KOTLIN
- ANDROID STUDIO
- SQLITE
- RECYCLERVIEW
- MPANDROIDCHART
- MATERIAL DESIGN / XML

¡Bienvenido a RumFit!

ENTRENAR AHORA

MIS RUTINAS

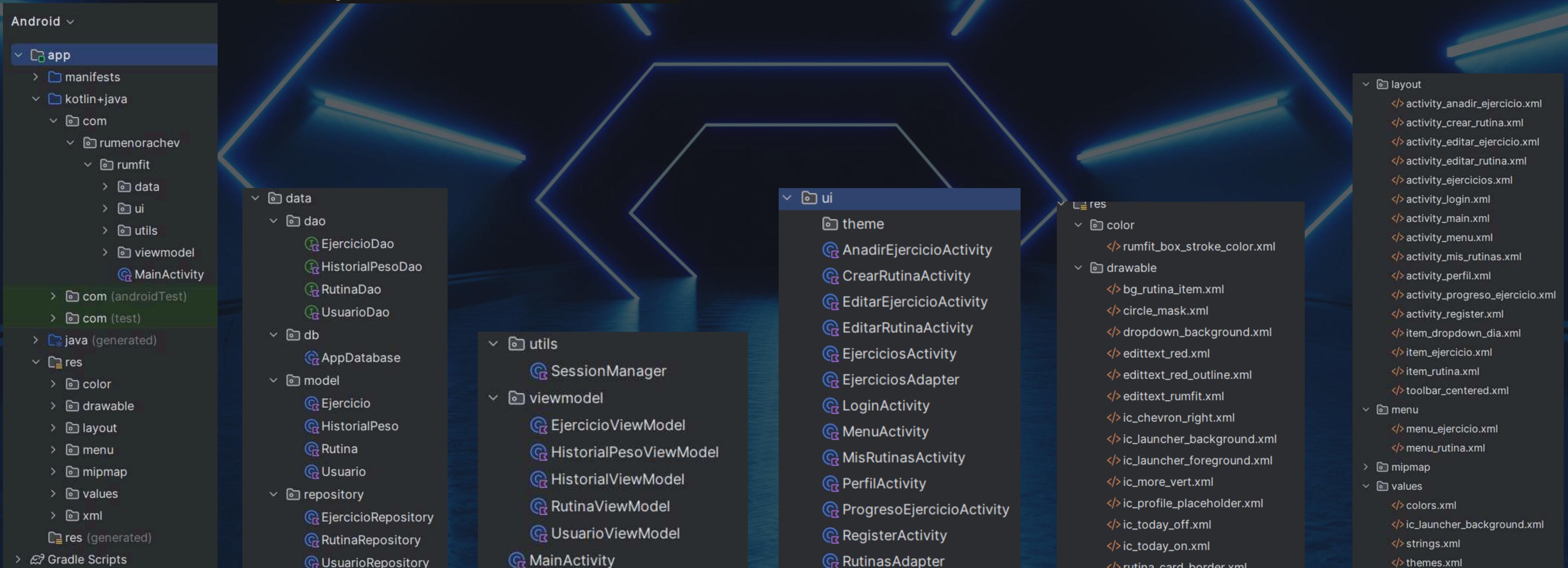
MI PERFIL

# ARQUITECTURA DEL PROYECTO

RUMFIT ESTÁ ORGANIZADA EN MÓDULOS CLAROS:

data/model	→ Entidades Room
data/dao	→ DAOs con consultas SQL
data/db	→ AppDatabase (RoomDatabase)
ui/activities	→ Pantallas de la app
ui/adapters	→ RecyclerView adapters
res/layout	→ UI XML

- SEPARACIÓN LIMPIA ENTRE UI, DATOS Y LÓGICA
- CÓDIGO MANTENIBLE
- ARQUITECTURA MODERNA ANDROID



# BASE DE DATOS: APPDATABASE (ROOM)

- ARCHIVO:  
**DATA/DB/APPDATABASE.KT**
- CARACTERÍSTICAS:
  - **BASE DE DATOS ROOM**
  - **4 ENTIDADES**
  - **4 DAO'S**
  - **ACCESO SINGLETON**
  - **MIGRACIONES**
  - **CONTROLADAS**

```
@Database(  
    entities = [  
        Rutina::class,  
        Usuario::class,  
        Ejercicio::class,  
        HistorialPeso::class // - NUEVA TABLA  
    ],  
    version = 6, // - SUBIMOS VERSIÓN  
    exportSchema = false  
)  
  
abstract class AppDatabase : RoomDatabase() {  
  
    1 Usage 1 Implementation  
    abstract fun rutinaDao(): RutinaDao  
    1 Usage 1 Implementation  
    abstract fun usuarioDao(): UsuarioDao  
    1 Usage 1 Implementation  
    abstract fun ejercicioDao(): EjercicioDao  
    2 Usages 1 Implementation  
    abstract fun historialPesoDao(): HistorialPesoDao // - NUEVO DAO
```

# ENTIDADES (MODELS)

- CADA TABLA ES UNA `@ENTITY`.
- EJEMPLO:  
`DATA/MODEL/EJERCICIO.KT`
- ESTO GENERA AUTOMÁATICAMENTE UNA TABLA EN ROOM.

```
// Un ejercicio que pertenece a una rutina
@Entity(tableName = "ejercicios")
data class Ejercicio(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val rutinaId: Int, // FK a la rutina
    val nombre: String, // p.ej. "Press banca"
    val series: Int, // n° de series
    val repeticiones: String, // n° de repeticiones
    val ultimoPeso: Float? = null, // Último peso usado (kg), puede ser null
    val notas: String? = null, // notas opcionales
    val orden: Int = 0 // nuevo
)
```

# DAOS: ACCESO A LOS DATOS

- LOS DAOS CONTIENEN LAS CONSULTAS SQL.
- ARCHIVO:  
**DATA/DAO/EJERCICIODAO.KT**
- VENTAJAS:
  - CÓDIGO LIMPIO
  - SQL TIPADO
  - SOPORTA COROUTINES

```
@Dao
interface EjercicioDao {

    1 Usage 1 Implementation
    @Query( value = "SELECT * FROM ejercicios WHERE rutinaId = :rutinaId ORDER BY orden ASC")
    fun getEjerciciosDeRutina(rutinaId: Int): Flow<List<Ejercicio>>

    1 Usage 1 Implementation
    @Query( value = "SELECT * FROM ejercicios WHERE id = :id LIMIT 1")
    fun getEjercicioById(id: Int): Flow<Ejercicio?>

    1 Usage 1 Implementation
    @Query( value = "SELECT COUNT(*) FROM ejercicios WHERE rutinaId = :rutinaId")
    suspend fun contarEjerciciosDeRutina(rutinaId: Int): Int

    1 Implementation
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(ejercicio: Ejercicio)

    1 Implementation
    @Update
    suspend fun update(ejercicio: Ejercicio)

    1 Implementation
    @Delete
    suspend fun delete(ejercicio: Ejercicio)
}
```

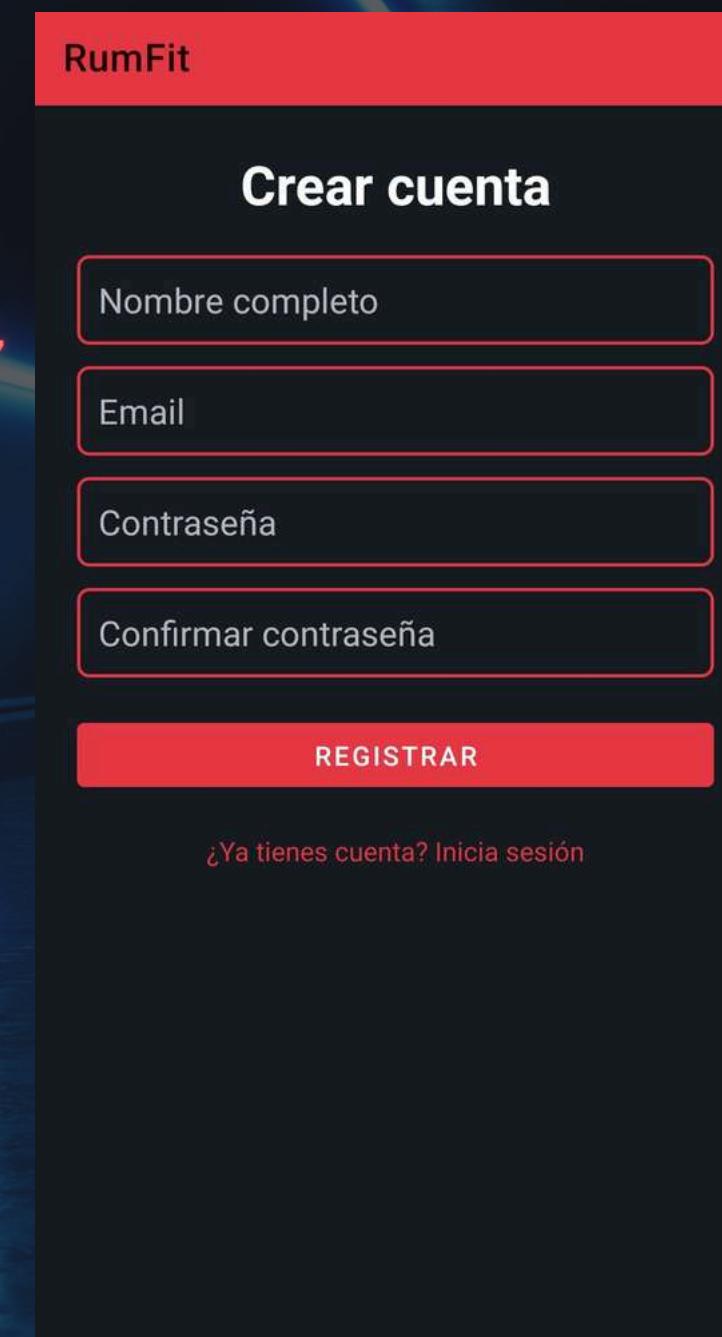
# REGISTRO DE USUARIO

## ARCHIVOS IMPLICADOS:

- UI/ACTIVITIES/REGISTERACTIVITY.KT
- USUARIODAO.KT
- ACTIVITY\_REGISTER.XML

## QUÉ HACE:

- CREA UN NUEVO USUARIO
- VALIDA DATOS (EMAIL, NOMBRE, CONTRASEÑA)
- INSERTA MEDIANTE ROOM



## CÓDIGO CLAVE:

```
// --- Comprobar si el email ya existe ---
usuarioViewModel.obtenerPorEmail(email) { existente ->
    if (existente != null) {
        Toast.makeText( context = this, text = "Este email ya está registrado", duration = Toast.LENGTH_SHORT).show()
        return@obtenerPorEmail
    }

    // Crear usuario nuevo
    val usuario = Usuario(
        nombre = nombre,
        email = email,
        password = password
    )

    usuarioViewModel.registrar(usuario)

    Toast.makeText( context = this, text = "Usuario registrado correctamente", duration = Toast.LENGTH_SHORT).show()

    startActivity(Intent( packageContext = this, cls = LoginActivity::class.java))
    finish()
}
```

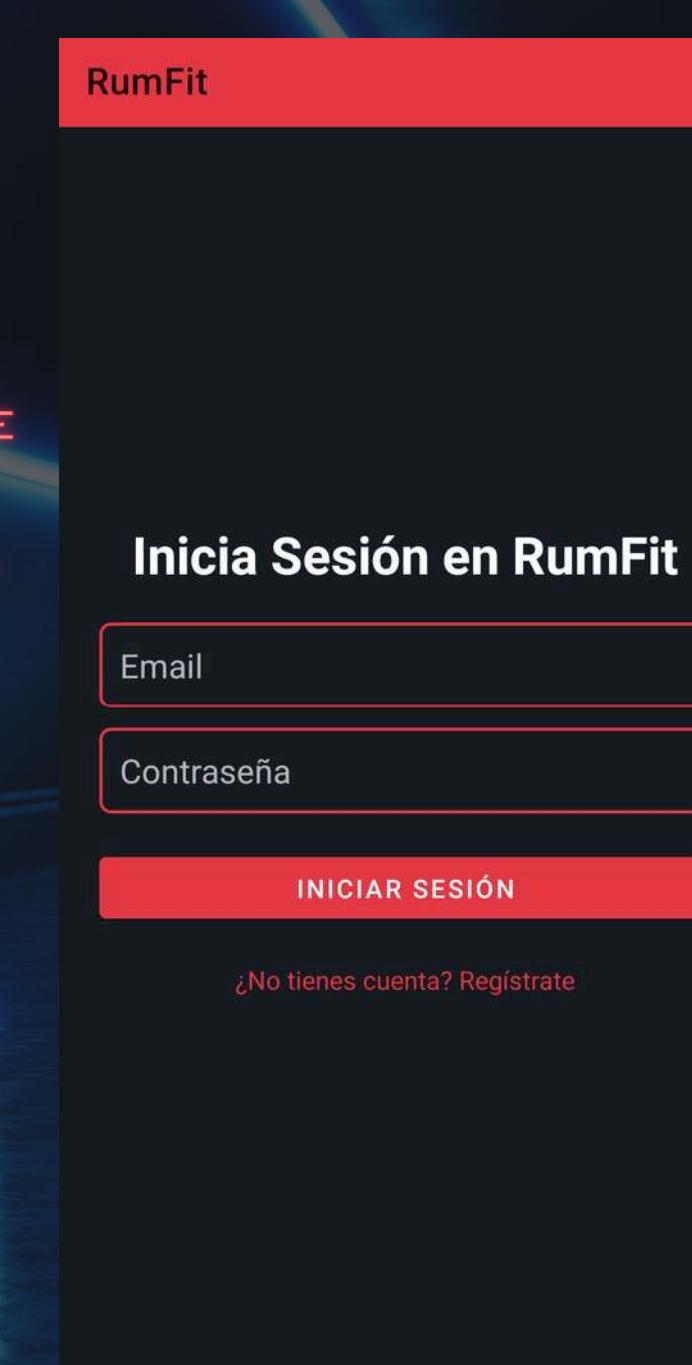
# INICIO DE SESIÓN (LOGIN)

## ARCHIVOS IMPLICADOS:

- UI/ACTIVITIES/LOGINACTIVITY.KT
- DATA/DAO/USUARIODAO.KT
- ACTIVITY\_LOGIN.XML

## QUÉ HACE:

- VALIDA EMAIL Y CONTRASEÑA
- BUSCA EL USUARIO EN LA BASE DE DATOS CON ROOM
- SI COINCIDE → ENTRA A LA APP
- SI NO → ERROR



## CÓDIGO CLAVE:

```
private fun iniciarSesion() {  
  
    val email = binding.inputEmailLogin.text.toString().trim()  
    val password = binding.inputPasswordLogin.text.toString().trim()  
  
    if (email.isEmpty() || password.isEmpty()) {  
        Toast.makeText(context = this, text = "Completa todos los campos", duration = Toast.LENGTH_SHORT).show()  
        return  
    }  
  
    usuarioViewModel.obtenerPorEmail(email) { usuario ->  
  
        if (usuario == null) {  
            Toast.makeText(context = this, text = "El usuario no existe", duration = Toast.LENGTH_SHORT).show()  
            return@obtenerPorEmail  
        }  
  
        if (usuario.password != password) {  
            Toast.makeText(context = this, text = "Contraseña incorrecta", duration = Toast.LENGTH_SHORT).show()  
            return@obtenerPorEmail  
        }  
  
        Toast.makeText(context = this, text = "Inicio de sesión correcto", duration = Toast.LENGTH_SHORT).show()  
    }  
}
```

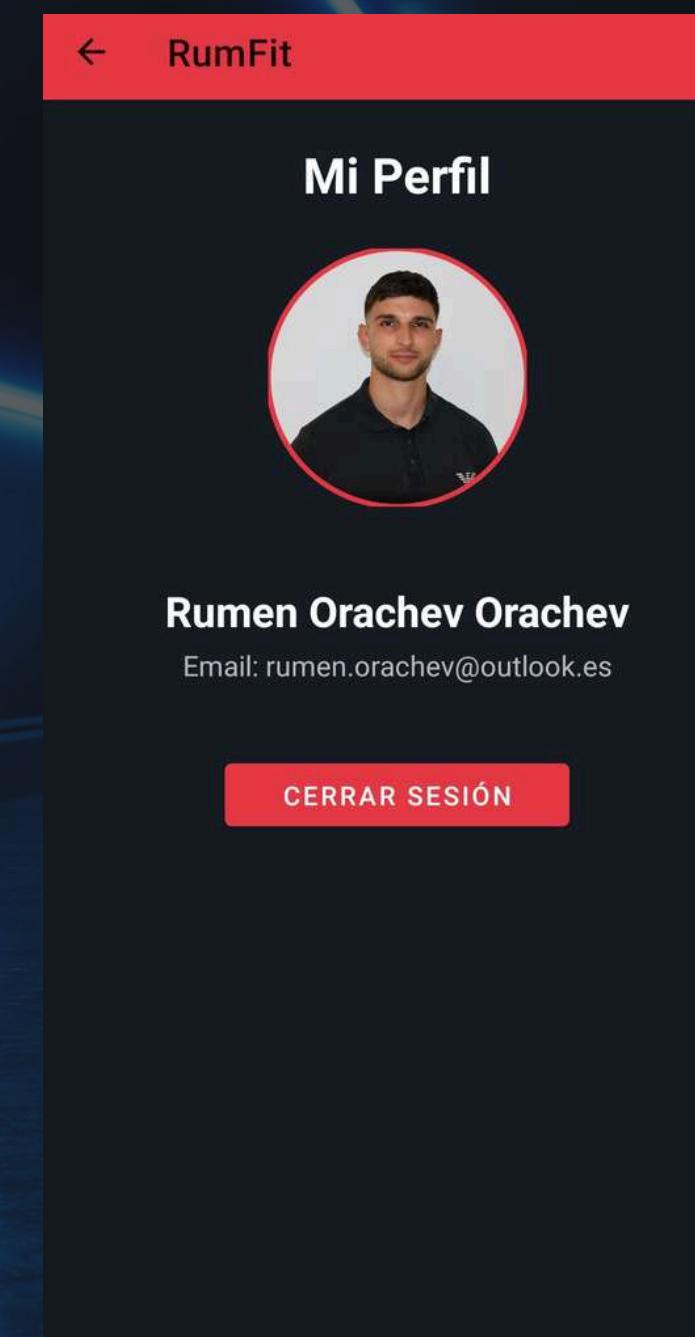
# PERFIL DEL USUARIO

## ARCHIVOS IMPLICADOS:

- UI/ACTIVITIES/PERFILACTIVITY.KT
- USUARIODAO.KT
- ACTIVITY\_PERFIL.XML

## QUÉ HACE:

- MUESTRA NOMBRE Y EMAIL DEL USUARIO
- PERMITE CAMBIAR FOTO DE PERFIL
- CARGA DATOS DESDE ROOM
- GUARDA IMAGEN CON URI



## CÓDIGO CLAVE:

```
Usage
private fun guardarFoto(uri: Uri) {
    val stream = contentResolver.openInputStream(uri)
    val bitmap = BitmapFactory.decodeStream( is = stream)

    val baos = ByteArrayOutputStream()
    bitmap.compress( format = Bitmap.CompressFormat.JPEG, quality = 70, stream = baos)

    val base64 =
        Base64.encodeToString( input = baos.toByteArray(), flags = Base64.DEFAULT)

    val usuario = usuarioActual ?: return

    val usuarioActualizado = usuario.copy(
        fotoPerfil = base64
    )

    usuarioViewModel.actualizarUsuario(usuarioActualizado)
    usuarioActual = usuarioActualizado

    Toast.makeText( context = this, text = "Foto actualizada", duration = Toast.LENGTH_SHORT).show()
}
```

# CREAR RUTINA

## ARCHIVOS IMPLICADOS:

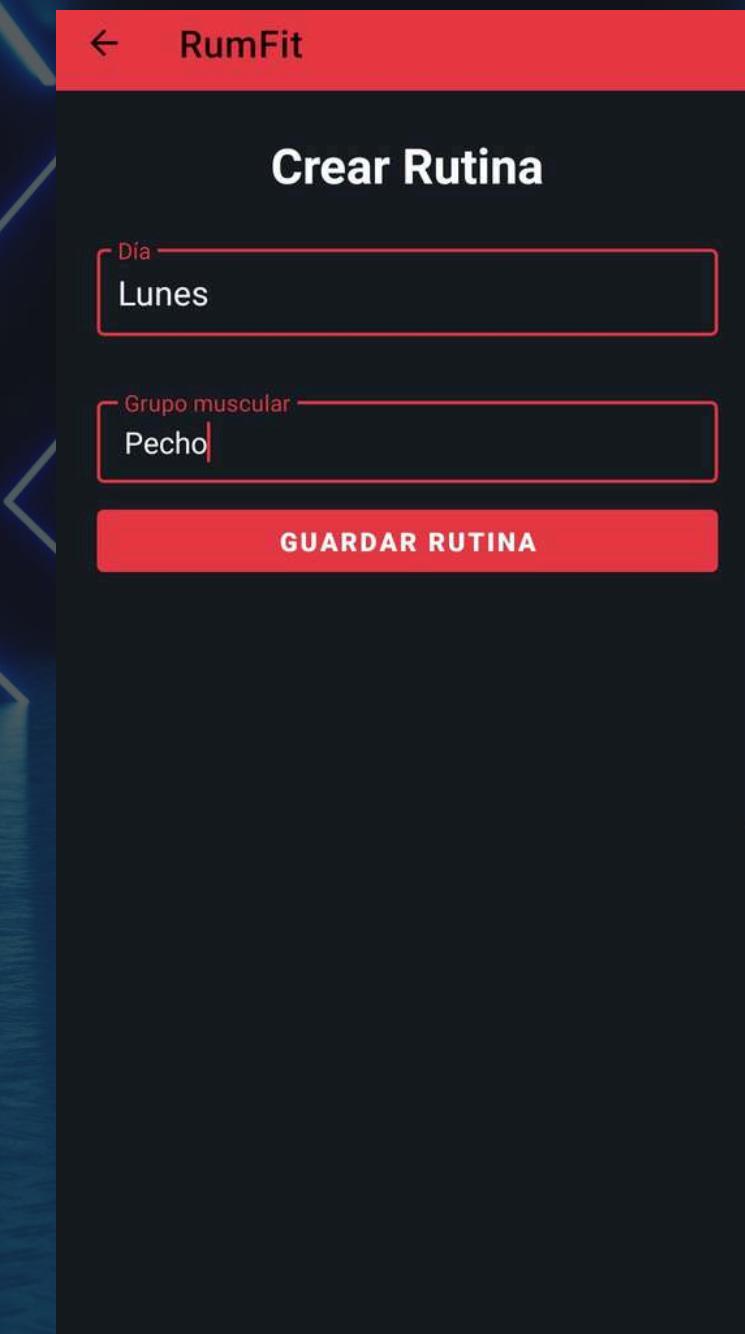
- UI/ACTIVITIES/CREARRUTINAACIVITY.KT
- DATA/DAO/RUTINADAO.KT
- RES/LAYOUT/ACTIVITY\_CREAR\_RUTINA.XML

## OPERACIÓN CRUD:

```
@Insert(onConflict = OnConflictStrategy.REPLACE)  
suspend fun insert(ejercicio: Ejercicio)
```

## INSERCIÓN:

```
suspend fun insert(rutina: Rutina) =  
    rutinaDao.insertRutina(rutina)
```



# LISTA DE RUTINAS (RECYCLERVIEW)

## ARCHIVOS IMPLICADOS:

- MISRUTINASACTIVITY.KT
- RUTINASADAPTER.KT
- ITEM\_RUTINA.XML



## CÓDIGO DEL ADAPTER:

```
override fun onBindViewHolder(holder: RutinaViewHolder, position: Int) {
    val rutina = lista[position]

    holder.binding.textNombreRutina.text = rutina.nombre
    holder.binding.textDescripcionRutina.text = rutina.descripcion

    val abrirDetalle = { view: android.view.View ->
        val intent = Intent( packageContext = view.context, cls = EjerciciosActivity::class.java)
        intent.putExtra( name = "id_rutina", value = rutina.id)
        intent.putExtra( name = "origen", value = "mis_rutinas")
        intent.addFlags( flags = Intent.FLAG_ACTIVITY_CLEAR_TOP)
        view.context.startActivity( p0 = intent)
    }
}
```

## CÓDIGO DEL ACTIVITY:

```
private val ordenDias = listOf(
    "Lunes", "Martes", "Miércoles",
    "Jueves", "Viernes", "Sábado", "Domingo"
)
```

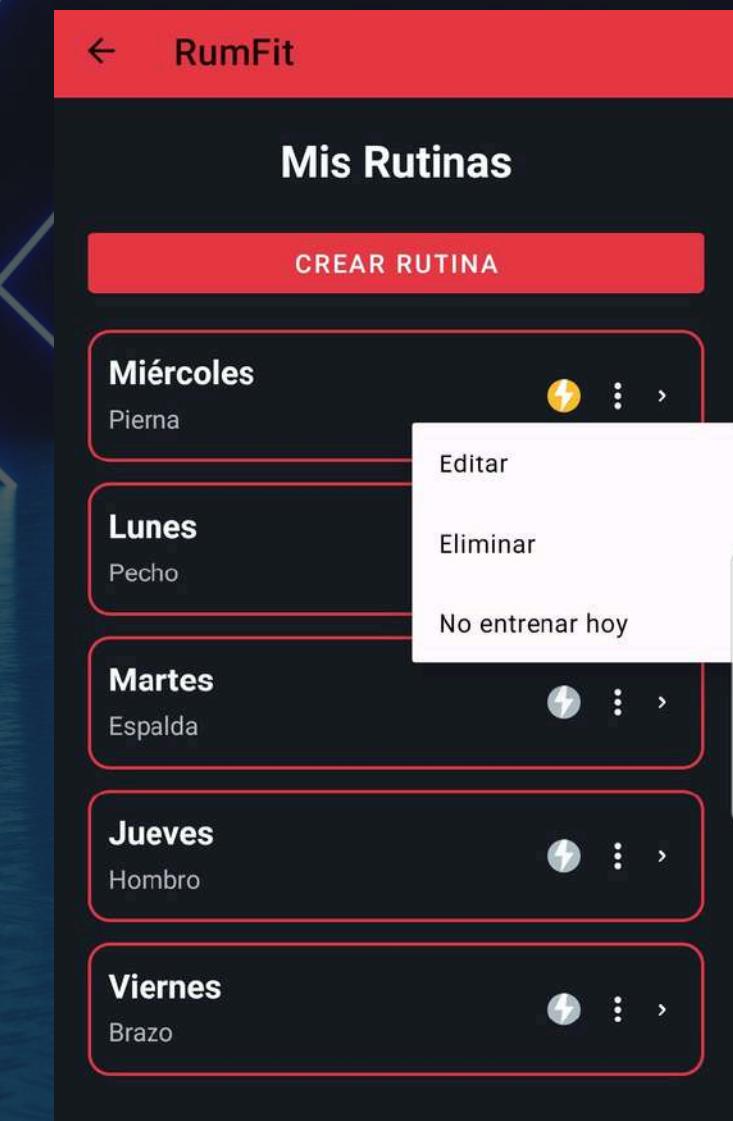
# SELECCIONAR RUTINA DEL DÍA

## ARCHIVOS IMPLICADOS:

- RUTINASACTIVITY.KT
- RUTINASADAPTER.KT
- RUTINADA0.KT

## OPERACIÓN CRUD:

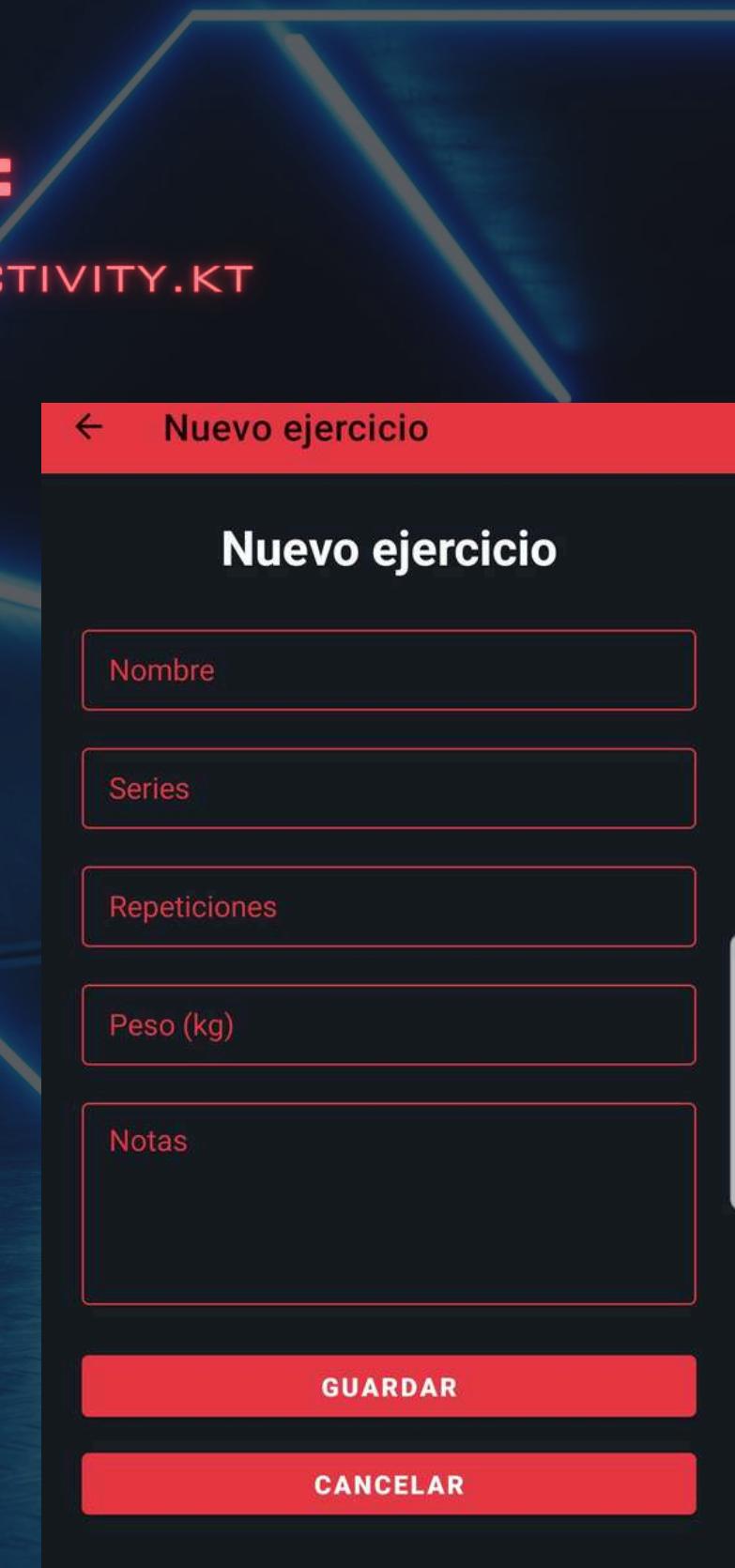
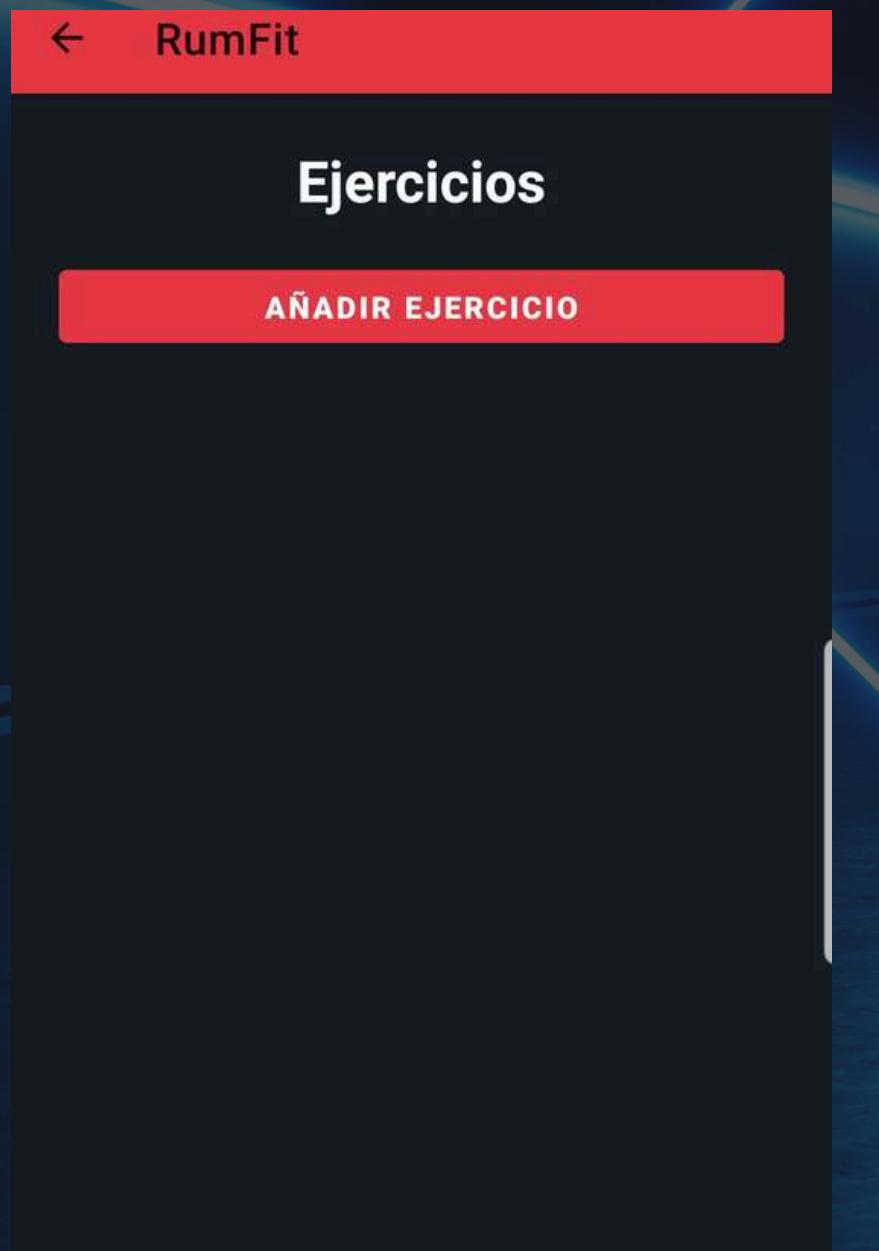
```
@Query( value = "SELECT * FROM rutinas WHERE emailUsuario = :email ORDER BY favorita DESC, id DESC")  
fun getRutinasByUsuario(email: String): Flow<List<Rutina>>  
  
1 Usage 1 Implementation  
@Query( value = "SELECT * FROM rutinas WHERE id = :id LIMIT 1")  
fun getRutinaById(id: Int): Flow<Rutina?>
```



# AÑADIR EJERCICIO

## ARCHIVOS IMPLICADOS:

- ANADIREJERCICIOACTIVITY.KT
- EJERCICIODOAO.KT



## CÓDIGO DEL ACTIVITY:

```
ejercicioViewModel.contarEjercicios(rutinaId) { cantidad ->

    val ejercicio = Ejercicio(
        rutinaId = rutinaId,
        nombre = nombre,
        series = series,
        repeticiones = repeticiones, // ★ TEXTO
        ultimoPeso = peso,
        notas = notas,
        orden = cantidad
    )

    ejercicioViewModel.insert(ejercicio)

    Toast.makeText( context = this, text = "Ejercicio guardado", duration = Toast.LENGTH_SHORT).show()
    finish()
}
```

# EDITAR Y ELIMINAR EJERCICIO

## ARCHIVOS

### IMPlicados:

- EDITAREJERCICIOACTIVITY.KT
- EJERCICIODOAO.KT
- EJERCICIOSADAPTER.KT



### EDITAR:

```
suspend fun update(ejercicio: Ejercicio) =  
    dao.update(ejercicio)
```

### ELIMINAR:

```
suspend fun delete(ejercicio: Ejercicio) =  
    dao.delete(ejercicio)
```

# REGISTRO DE PROGRESO

## ARCHIVOS

### IMPLICADOS:

- DATA/DAO/HISTORIALPESODAO.KT
- DATA/MODEL/HISTORIALPESO.KT
- EDITAREJERCICIOACTIVITY.KT

← Editar ejercicio

### Editar ejercicio

Nombre: Presa Inclinado Mancuernas

Serie: 6

Repeticiones: 15-12-10-8-6-15

Peso (kg): 42.0

Notas: Cada mancuerna

**GUARDAR CAMBIOS**

**CANCELAR**

### PARA REGISTRAR EL PESO:

```
@Insert  
suspend fun insertRegistro(registro: HistorialPeso)
```

### AL EDITAR EL PESO:

```
fun insertarRegistro(ejercicioId: Int, peso: Float) {  
    val registro = HistorialPeso(  
        ejercicioId = ejercicioId,  
        peso = peso,  
        fecha = System.currentTimeMillis()  
    )  
  
    viewModelScope.launch {  
        dao.insertRegistro(registro) // ← AQUÍ el nombre correcto  
    }  
}
```

# GRÁFICA (MPANDROIDCHART)

## ARCHIVOS IMPLICADOS:

- PROGRESOACTIVITY.KT



## CÓDIGO:

```
val entries = lista.mapIndexed { index, item ->
    Entry( x = index.toFloat(), y = item.peso)
}

// COLOR ROJO DE RUMFIT
val rojo = resources.getColor( id = R.color.rumfit_accent, theme)
val texto = resources.getColor( id = R.color.rumfit_text_primary, theme)
val fondo = resources.getColor( id = R.color.rumfit_background, theme)

val dataSet = LineDataSet( yVals = entries, label = "Peso (kg)").apply {
    lineWidth = 3f
    circleRadius = 5f
    color = rojo
    setCircleColor(rojo)
    setDrawValues(false)
    mode = LineDataSet.Mode.CUBIC_BEZIER
}

val lineData = LineData( ...dataSets = dataSet)
```

# CONCLUSIÓN

RUMFIT ES UNA APLICACIÓN  
COMPLETA Y FUNCIONAL  
QUE INTEGRA:

- REGISTRO E INICIO DE SESIÓN CON PERSISTENCIA EN ROOM
- PERFIL DEL USUARIO CON FOTO Y DATOS PERSONALES
- GESTIÓN COMPLETA DE RUTINAS (CREAR, EDITAR, ELIMINAR, MARCAR COMO ACTIVA)
- VALIDACIÓN INTELIGENTE DE DÍAS OCUPADOS
- LISTA DINÁMICA DE EJERCICIOS POR RUTINA (CRUD COMPLETO)
- NOTAS, PESO, SERIES Y REPETICIONES PERSONALIZABLES
- REGISTRO AUTOMÁTICO DEL PROGRESO AL MODIFICAR EL PESO
- HISTORIAL DE CADA EJERCICIO ALMACENADO EN ROOM
- GRÁFICA DINÁMICA CON MPANDROIDCHART PARA VISUALIZAR LA EVOLUCIÓN
- DISEÑO LIMPIO Y COHERENTE EN XML
- NAVEGACIÓN CLARA Y ESTRUCTURADA ENTRE PANTALLAS
- ARQUITECTURA PROFESIONAL BASADA EN ROOM + DAOS + ENTIDADES

RESULTADO:

UNA APP SÓLIDA, COMPLETA, ESCALABLE Y EN EVOLUCIÓN.



**¡GRACIAS POR VUESTRA ATENCIÓN!**

Progreso del peso



**RumFit**

APP FITNESS PARA ANDROID

**DESCARGA DISPONIBLE EN:**

[WWW.RUMENORACHEV.ES](http://WWW.RUMENORACHEV.ES)

[WWW.GITHUB.COM/RUMEENORACHEV/RUMFIT-PUBLIC](http://WWW.GITHUB.COM/RUMEENORACHEV/RUMFIT-PUBLIC)



Progreso del peso

100

00

01 dic.

Peso (kg)

17 dic.