# PictoComm2 - Documentación Técnica

> **Aplicación Android de Comunicación Alternativa Aumentativa (AAC)** Sistema de pictogramas con control parental y persistencia local mediante Room Database

## Tabla de Contenidos



## 1. Visión General del Proyecto

### ¿Qué es PictoComm2?

**PictoComm2** es una aplicación Android diseñada para personas con dificultades de comunicación verbal. Permite construir oraciones mediante la selección de pictogramas (imágenes que representan palabras) y reproducirlas mediante voz sintética (Text-to-Speech).

### Características Principales

- **Sistema de Pictogramas**: 51 pictogramas precargados organizados en 6 categorías gramaticales
- **Construcción de Oraciones**: Interfaz táctil intuitiva para formar frases
- **Text-to-Speech**: Reproducción de oraciones en español
- **Predicción Inteligente**: Sugiere automáticamente la siguiente categoría de palabras
- **Control Parental**: Sistema de usuarios PADRE/HIJO con aprobación de pictogramas
- **Persistencia Local**: Base de datos Room para almacenar pictogramas y usuarios sin conexión
- **Sistema de Favoritos**: Acceso rápido a pictogramas frecuentes
- **Creación de Pictogramas**: Usuarios pueden crear sus propios pictogramas personalizados

### Stack Tecnológico

```
- Lenguaje: Kotlin
- UI: XML Layouts (Material Design 3)
- Base de Datos: Room Database (SQLite)
- Arquitectura: MVVM (Model-View-ViewModel)
- Gestión de Estado: StateFlow (Kotlin Coroutines)
- Async: Coroutines + Flow
```

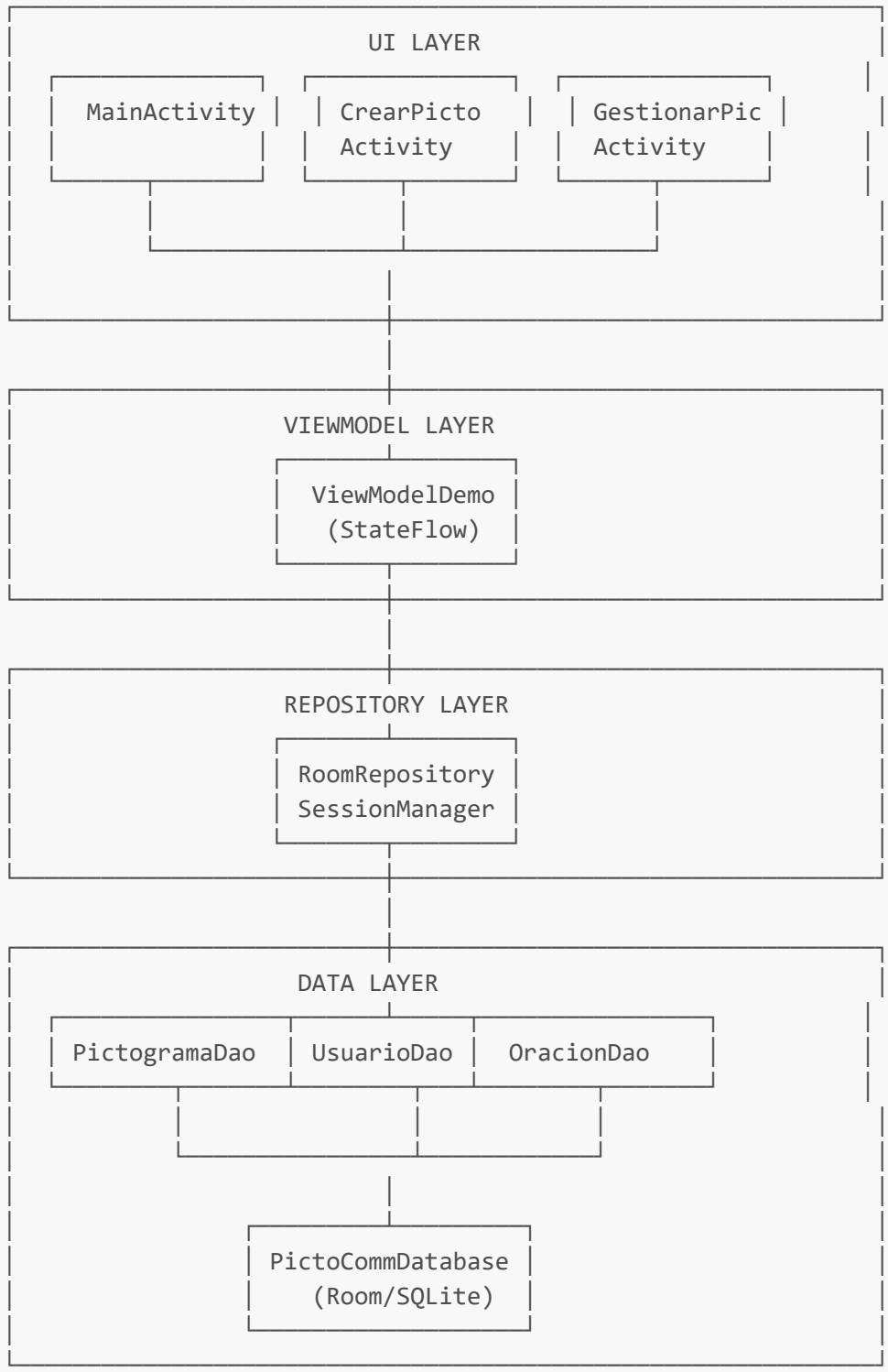
```
- Text-to-Speech: Android TTS API
- Min SDK: API 24 (Android 7.0)
- Target SDK: API 34 (Android 14)
```

---

## 2. Arquitectura del Proyecto

### 📐 Patrón MVVM + Repository

PictoComm2 implementa una arquitectura limpia y escalable basada en MVVM:



### Estructura de Carpetas

```
com.inacap.picto_comm/
├── data/
│   ├── database/
│   │   ├── entities/          # Entidades de Room
│   │   │   ├── PictogramaEntity.kt
│   │   │   ├── UsuarioEntity.kt
│   │   │   └── OracionEntity.kt
│   │   ├── dao/               # Data Access Objects
│   │   │   ├── PictogramaDao.kt
│   │   │   ├── UsuarioDao.kt
│   │   │   └── OracionDao.kt
│   │   ├── converters/        # Type Converters
│   │   │   └── Converters.kt
│   │   └── PictoCommDatabase.kt
│   ├── model/                 # Modelos de dominio
│   │   ├── PictogramaSimple.kt
│   │   ├── Usuario.kt
│   │   ├── Categoria.kt
│   │   └── TipoUsuario.kt
│   ├── repository/            # Repositorios
│   │   ├── RoomRepository.kt
│   │   └── SessionManager.kt
│   └── FuenteDatosMock.kt     # Datos de prueba
├── ui/
│   ├── screens/               # Activities
│   │   ├── SplashActivity.kt
│   │   ├── ConfiguracionInicialActivity.kt
│   │   ├── SeleccionPerfilActivity.kt
│   │   ├── PinActivity.kt
│   │   ├── CrearPictogramaActivity.kt
│   │   └── GestionarPictogramasActivity.kt
│   ├── adapters/              # RecyclerView Adapters
│   │   ├── PictogramaAdapter.kt
│   │   ├── CategoriaAdapter.kt
│   │   ├── OracionAdapter.kt
│   │   └── PictogramaPendienteAdapter.kt
│   ├── viewmodel/             # ViewModels
│   │   └── ViewModelDemo.kt
│   └── utils/                 # Utilidades UI
│       ├── IconoHelper.kt
│       └── SessionManager.kt
├── MainActivity.kt
└── PictoCommApplication.kt    # Application class
```

---

## 3. Room Database - Persistencia Local

### ¿Qué es Room Database?

**Room** es una biblioteca de persistencia que proporciona una capa de abstracción sobre SQLite. En PictoComm2, Room permite:

- ☑ Almacenar pictogramas, usuarios y oraciones localmente
- ☑ Funcionar completamente offline (sin internet)
- ☑ Acceso a datos mediante objetos Kotlin (no SQL directo)
- ☑ Validación en tiempo de compilación
- ☑ Operaciones asíncronas con Coroutines

## 📊 Esquema de Base de Datos

La base de datos `PictoCommDatabase` contiene 3 tablas principales:

```
┌──────────────────────────────────────────────────────┐
│                    PICTOGRAMAS                       │
├──────────────────┬───────────────────────────────────┤
│ id               │ Long (PK, AutoGenerate)           │
│ texto            │ String (ej: "Yo", "Quiero")       │
│ categoria        │ String (PERSONAS, ACCIONES, etc.) │
│ recursoImagen    │ String (nombre del drawable)      │
│ esFavorito       │ Boolean (marcado como favorito)   │
│ frecuenciaUso    │ Int (contador de veces usado)     │
│ aprobado         │ Boolean (aprobado por PADRE)      │
│ creadoPor        │ Long (ID del usuario creador)     │
│ tipoImagen       │ String ("ICONO" o "FOTO")         │
│ rutaImagen       │ String (ruta local de foto)       │
│ fechaCreacion    │ Long (timestamp)                  │
└──────────────────┴───────────────────────────────────┘

┌──────────────────────────────────────────────────────┐
│                     USUARIOS                         │
├──────────────────┬───────────────────────────────────┤
│ id               │ Long (PK, AutoGenerate)           │
│ nombre           │ String (nombre del usuario)       │
│ tipo             │ String ("PADRE" o "HIJO")         │
│ pin              │ String (4 dígitos, solo PADRE)    │
│ email            │ String (opcional)                 │
│ fechaCreacion    │ Long (timestamp)                  │
│ activo           │ Boolean (soft delete)             │
└──────────────────┴───────────────────────────────────┘

┌──────────────────────────────────────────────────────┐
│                    ORACIONES                         │
├──────────────────┬───────────────────────────────────┤
│ id               │ Long (PK, AutoGenerate)           │
│ pictogramaIds    │ List<Long> (IDs en orden)         │
│ textoCompleto    │ String ("Yo Quiero Comer Helado") │
│ fechaCreacion    │ Long (timestamp)                  │
│ vecesUsada       │ Int (contador)                    │
└──────────────────┴───────────────────────────────────┘
```

## 🔧 Componentes de Room

### 3.1. Entities (Entidades)

Las entidades son clases Kotlin anotadas con `@Entity` que representan tablas:

**Ejemplo: PictogramaEntity.kt**

```
@Entity(tableName = "pictogramas")
data class PictogramaEntity(
    @PrimaryKey(autoGenerate = true)
    val id: Long = 0,

    val texto: String,                    // "Yo", "Quiero", "Helado"
    val categoria: String,                // "PERSONAS", "ACCIONES", "COSAS"
    val recursoImagen: String,            // "ic_person_yo"
    val esFavorito: Boolean = false,
    val frecuenciaUso: Int = 0,

    // Control parental
    val aprobado: Boolean = true,         // Por defecto aprobado
    val creadoPor: Long = 0,              // 0 = sistema, >0 = usuario

    // Fotos personalizadas
    val tipoImagen: String = "ICONO",     // "ICONO" o "FOTO"
    val rutaImagen: String = "",

    val fechaCreacion: Long = System.currentTimeMillis()
) {
    // Conversión a modelo de dominio
    fun toModel(): PictogramaSimple {
        return PictogramaSimple(
            id = id.toString(),
            texto = texto,
            categoria = Categoria.valueOf(categoria),
            recursoImagen = recursoImagen,
            esFavorito = esFavorito,
            aprobado = aprobado,
            creadoPor = creadoPor.toString(),
            tipoImagen = TipoImagen.valueOf(tipoImagen),
            urlImagen = rutaImagen,
            fechaCreacion = fechaCreacion
        )
    }
}
```

**Características importantes**:

- `@PrimaryKey(autoGenerate = true)`: ID autoincremental
- `tableName = "pictogramas"`: Nombre de la tabla en SQLite
- Método `toModel()`: Convierte Entity a modelo de dominio (desacopla BD de UI)

### 3.2. DAOs (Data Access Objects)

Los DAOs definen las operaciones sobre la base de datos mediante interfaces:

**Ejemplo: PictogramaDao.kt**

```
@Dao
interface PictogramaDao {

    // CREATE - Insertar pictogramas
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertar(pictograma: PictogramaEntity): Long

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertarTodos(pictogramas: List<PictogramaEntity>)

    // READ - Consultas con Flow (observable)
    @Query("SELECT * FROM pictogramas WHERE aprobado = 1 ORDER BY texto ASC")
    fun obtenerTodosAprobados(): Flow<List<PictogramaEntity>>

    @Query("SELECT * FROM pictogramas WHERE categoria = :categoria AND aprobado =
1")
    fun obtenerPorCategoria(categoria: String): Flow<List<PictogramaEntity>>

    @Query("SELECT * FROM pictogramas WHERE esFavorito = 1 AND aprobado = 1")
    fun obtenerFavoritos(): Flow<List<PictogramaEntity>>

    // Control parental
    @Query("SELECT * FROM pictogramas WHERE aprobado = 0 ORDER BY fechaCreacion
DESC")
    fun obtenerPendientesAprobacion(): Flow<List<PictogramaEntity>>

    @Query("UPDATE pictogramas SET aprobado = 1 WHERE id = :id")
    suspend fun aprobar(id: Long)

    @Query("DELETE FROM pictogramas WHERE id = :id")
    suspend fun rechazar(id: Long)

    // UPDATE - Actualizar campos
    @Query("UPDATE pictogramas SET esFavorito = :esFavorito WHERE id = :id")
    suspend fun actualizarFavorito(id: Long, esFavorito: Boolean)

    @Query("UPDATE pictogramas SET frecuenciaUso = frecuenciaUso + 1 WHERE id =
:id")
    suspend fun incrementarFrecuencia(id: Long)

    // DELETE
    @Delete
    suspend fun eliminar(pictograma: PictogramaEntity)
}
```

**Características importantes**:

- `@Query`: SQL validado en tiempo de compilación
- `suspend fun`: Operaciones asíncronas (no bloquean UI)
- `Flow<List<T>>`: Observable reactivo (emite cambios automáticamente)
- `OnConflictStrategy.REPLACE`: Actualiza si ya existe

### 3.3. Database (Singleton)

Clase abstracta que define la base de datos:

**Archivo: PictoCommDatabase.kt**

```
@Database(
    entities = [
        PictogramaEntity::class,
        UsuarioEntity::class,
        OracionEntity::class
    ],
    version = 1,
    exportSchema = false
)
@TypeConverters(Converters::class)
abstract class PictoCommDatabase : RoomDatabase() {

    // DAOs abstractos
    abstract fun pictogramaDao(): PictogramaDao
    abstract fun usuarioDao(): UsuarioDao
    abstract fun oracionDao(): OracionDao

    companion object {
        @Volatile
        private var INSTANCE: PictoCommDatabase? = null

        // Singleton thread-safe
        fun getDatabase(context: Context): PictoCommDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    PictoCommDatabase::class.java,
                    "pictocomm_database"
                )
                .addCallback(object : RoomDatabase.Callback() {
                    override fun onCreate(db: SupportSQLiteDatabase) {
                        super.onCreate(db)
                        // Aquí se pueden cargar datos iniciales
                    }
                })
                .build()

                INSTANCE = instance
                instance
```

```
            }
        }
    }
}
```

**Características importantes**:

- `@Database`: Define entidades y versión
- `@TypeConverters`: Convierte tipos complejos (List → String)
- **Singleton Pattern**: Una sola instancia en toda la app
- `@Volatile`: Garantiza visibilidad entre threads

#### 3.4. Type Converters

Convierten tipos complejos para almacenar en SQLite:

**Archivo: Converters.kt**

```
class Converters {

    @TypeConverter
    fun fromLongList(value: List<Long>): String {
        return value.joinToString(",")
    }

    @TypeConverter
    fun toLongList(value: String): List<Long> {
        return if (value.isEmpty()) emptyList()
        else value.split(",").map { it.toLong() }
    }
}
```

**Uso**: Permite almacenar `List<Long>` (pictogramaIds) como String ("1,2,3,4")

---

## 🔄 Flujo de Datos con Room

```
┌─────────────────────────────────────────────────────────┐
│                    FLUJO DE LECTURA                     │
└─────────────────────────────────────────────────────────┘

1. UI (Activity)
   ↓ Solicita datos
2. ViewModel
   ↓ Llama al Repository
3. RoomRepository
   ↓ Llama al DAO
4. PictogramaDao.obtenerTodosAprobados()
   ↓ Ejecuta Query SQL
```

```
5. Room Database (SQLite)
   ↓ Retorna Flow<List<PictogramaEntity>>
6. Repository
   ↓ Convierte Entity → Model (toModel())
7. ViewModel
   ↓ Actualiza StateFlow
8. UI
   ↓ Observa StateFlow y actualiza RecyclerView

┌─────────────────────────────────────────────────────────┐
│                   FLUJO DE ESCRITURA                    │
└─────────────────────────────────────────────────────────┘

1. UI (Activity)
   ↓ Usuario crea pictograma
2. ViewModel (opcional)
   ↓
3. RoomRepository.crearPictograma(pictograma)
   ↓ Convierte Model → Entity (fromModel())
4. PictogramaDao.insertar(entity)
   ↓ Ejecuta INSERT SQL
5. Room Database (SQLite)
   ↓ Retorna ID generado
6. Flow observado emite cambio automáticamente
   ↓
7. UI se actualiza reactivamente
```

---

# 4. Sistema de Usuarios y Sesiones

## 👥 Tipos de Usuario

PictoComm2 maneja 2 tipos de usuarios con permisos diferenciados:

```kotlin
enum class TipoUsuario {
    PADRE,  // Administrador - Control total
    HIJO    // Usuario limitado - Requiere aprobación
}
```

**Comparación de Permisos**:

| Funcionalidad | PADRE (Admin) | HIJO (Limitado) |
|---|---|---|
| Ver pictogramas aprobados | ☑ | ☑ |
| Ver pictogramas NO aprobados | ☑ | ❌ |
| Crear pictogramas con aprobación automática | ☑ | ❌ |
| Crear pictogramas pendientes de aprobación | N/A | ☑ |

| Funcionalidad | PADRE (Admin) | HIJO (Limitado) |
|---|---|---|
| Aprobar/Rechazar pictogramas | ☑ | ✗ |
| Gestionar usuarios | ☑ | ✗ |
| Cambiar PIN | ☑ | ✗ |
| Acceder a pantalla de gestión | ☑ | ✗ |

### Sistema de PIN

**Características**:

- Solo el usuario PADRE tiene PIN (4 dígitos)
- Almacenado en SharedPreferences (en texto plano - mejora pendiente)
- Verificado antes de acceder a MainActivity si el usuario es PADRE

**Flujo de Autenticación**:

```
Usuario selecciona perfil
        ↓
¿Es tipo PADRE?
   ↙      ↘
  SÍ      NO
  ↓       ↓
PinActivity → MainActivity
  ↓
Solicita PIN (4 dígitos)
  ↓
¿Correcto?
  ↙    ↘
 SÍ     NO
 ↓      ↓
MainActivity  Error
```

### SessionManager

Gestiona la sesión activa mediante **SharedPreferences**:

**Archivo: SessionManager.kt**

```kotlin
class SessionManager(context: Context) {
    private val prefs: SharedPreferences = context.getSharedPreferences(
        "pictocomm_session",
        Context.MODE_PRIVATE
    )

    // Guardar sesión
    fun guardarSesion(usuario: Usuario) {
        prefs.edit().apply {
```

```
                putString("user_id", usuario.id)
                putString("user_name", usuario.nombre)
                putString("user_type", usuario.tipo.name)
                putBoolean("is_logged_in", true)
                apply()
            }
        }

        // Obtener usuario activo
        fun obtenerUsuarioId(): Long {
            val id = prefs.getString("user_id", "0") ?: "0"
            return id.toLongOrNull() ?: 0L
        }

        fun obtenerNombreUsuario(): String {
            return prefs.getString("user_name", "Usuario") ?: "Usuario"
        }

        fun obtenerTipoUsuario(): String {
            return prefs.getString("user_type", TipoUsuario.HIJO.name)
                ?: TipoUsuario.HIJO.name
        }

        // Verificar sesión activa
        fun hayUsuarioActivo(): Boolean {
            return prefs.getBoolean("is_logged_in", false)
        }

        // Cerrar sesión
        fun cerrarSesion() {
            prefs.edit().clear().apply()
        }
    }
```

**Uso en Application Class**:

```
    class PictoCommApplication : Application() {

        // Singleton de SessionManager
        val sessionManager: SessionManager by lazy {
            SessionManager(applicationContext)
        }

        // Singleton de Repository
        val repository: RoomRepository by lazy {
            val database = PictoCommDatabase.getDatabase(applicationContext)
            RoomRepository(
                database.pictogramaDao(),
                database.oracionDao(),
                database.usuarioDao()
            )
```

```
    }
}
```

---

# 5. Sistema de Pictogramas

## Categorías Gramaticales

Los pictogramas están organizados en 6 categorías con colores distintivos:

```
enum class Categoria(
    val nombreMostrar: String,
    val color: Long  // Color en hexadecimal
) {
    PERSONAS("Personas", 0xFF4CAF50),     // Verde
    ACCIONES("Acciones", 0xFF2196F3),     // Azul
    COSAS("Cosas", 0xFFFFC107),           // Amarillo
    CUALIDADES("Cualidades", 0xFFFF5722),// Naranja
    LUGARES("Lugares", 0xFF9C27B0),       // Morado
    TIEMPO("Tiempo", 0xFF00BCD4)          // Cian
}
```

## Pictogramas Precargados

La app incluye **51 pictogramas del sistema**:

| Categoría | Cantidad | Ejemplos |
|---|---|---|
| PERSONAS | 8 | Yo, Tú, Mamá, Papá, Nosotros, Profesor, Amigo |
| ACCIONES | 12 | Quiero, Tengo, Necesito, Comer, Beber, Jugar, Dormir, Ir al baño |
| COSAS | 10 | Helado, Agua, Comida, Juguete, Libro, Pelota, TV, Música, Tablet |
| CUALIDADES | 9 | Hambre, Sed, Sueño, Feliz, Triste, Enojado, Cansado, Grande, Pequeño |
| LUGARES | 7 | Casa, Escuela, Parque, Hospital, Baño, Cocina, Habitación |
| TIEMPO | 5 | Ahora, Después, Mañana, Hoy, Ayer |

**Fuente**: `FuenteDatosMock.kt` - Se cargan automáticamente en la BD en la primera ejecución.

## Operaciones CRUD

**Ejemplo de uso en Repository**:

```
// CREATE - Crear pictograma
val nuevoPictograma = PictogramaSimple(
    texto = "Chocolate",
    categoria = Categoria.COSAS,
```

```
    recursoImagen = "ic_thing_icecream",
    aprobado = false,  // Pendiente si lo crea HIJO
    creadoPor = usuarioId.toString()
)
val id = repository.crearPictograma(nuevoPictograma)

// READ - Obtener pictogramas aprobados
lifecycleScope.launch {
    repository.obtenerTodosPictogramas().collect { pictogramas ->
        // Se actualiza automáticamente cuando cambian
        adapter.submitList(pictogramas)
    }
}

// UPDATE - Marcar como favorito
repository.alternarFavorito(pictogramaId, esFavorito = true)

// DELETE - Rechazar pictograma
repository.rechazarPictograma(pictogramaId)
```

---

## 6. Sistema de Aprobación de Pictogramas

### ☑ Flujo Completo de Aprobación

```
┌─────────────────────────────────────────────────────────┐
│              HIJO crea nuevo pictograma                 │
└─────────────────────────────────────────────────────────┘
                          ↓
         CrearPictogramaActivity
                          ↓
         ┌──────────────────────────────┐
         │ PictogramaSimple             │
         │ - texto: "Chocolate"         │
         │ - categoria: COSAS           │
         │ - aprobado: false ❌         │ ← Pendiente
         │ - creadoPor: 2 (HIJO)        │
         └──────────────────────────────┘
                          ↓
         repository.crearPictograma()
                          ↓
         Room Database (INSERT)
                          ↓
    "Tu pictograma está pendiente de aprobación"
                          ↓
         Vuelve a MainActivity
                          ↓
    HIJO NO ve el pictograma (aprobado = false)

┌─────────────────────────────────────────────────────────┐
│          PADRE accede a Gestionar Pictogramas           │
```

```
└──────────────────────────────────────────────────────────────┘
                          ↓
      GestionarPictogramasActivity
                          ↓
    repository.obtenerPictogramasPendientes()
                          ↓
    Flow<List<PictogramaSimple>> donde aprobado = false
                          ↓
         RecyclerView muestra:
        ┌──────────────────────────────────┐
        │ 🖼️  Chocolate                    │
        │ Categoría: Cosas                 │
        │ Creado por: Hijo1                │
        │ [Rechazar] [Aprobar]             │
        └──────────────────────────────────┘
                          ↓
         PADRE presiona "Aprobar"
                          ↓
    repository.aprobarPictograma(id)
                          ↓
    UPDATE pictogramas SET aprobado = 1 WHERE id = X
                          ↓
         Room Database actualiza
                          ↓
    Flow emite cambio automáticamente
                          ↓
    Lista de pendientes se actualiza (se quita)
                          ↓
    "Pictograma aprobado" (Toast)

┌──────────────────────────────────────────────────────────────┐
│          Usuario vuelve a MainActivity                       │
└──────────────────────────────────────────────────────────────┘
                          ↓
         onResume() se ejecuta
                          ↓
    cargarPictogramasDesdeBaseDatos()
                          ↓
    repository.obtenerPictogramasAprobados()
                          ↓
    SELECT * FROM pictogramas WHERE aprobado = 1
                          ↓
    Incluye "Chocolate" (ahora aprobado = true ☑)
                          ↓
    ViewModelDemo.cargarPictogramasDesdeBaseDatos()
                          ↓
    entity.toModel() convierte correctamente
                          ↓
    StateFlow se actualiza
                          ↓
    UI muestra "Chocolate" en la lista ☑
```

## Métodos Clave

**En RoomRepository.kt**:

```
// Obtener pictogramas pendientes (solo PADRE)
fun obtenerPictogramasPendientes(): Flow<List<PictogramaSimple>> {
    return pictogramaDao.obtenerPendientesAprobacion().map { lista ->
        lista.map { it.toModel() }
    }
}

// Aprobar pictograma
suspend fun aprobarPictograma(pictogramaId: String) {
    val id = pictogramaId.toLongOrNull() ?: return
    pictogramaDao.aprobar(id)  // UPDATE aprobado = 1
}

// Rechazar pictograma (elimina permanentemente)
suspend fun rechazarPictograma(pictogramaId: String) {
    val id = pictogramaId.toLongOrNull() ?: return
    pictogramaDao.rechazar(id)  // DELETE
}

// Contar pendientes
suspend fun contarPictogramasPendientes(): Int {
    return pictogramaDao.contarPendientes()
}
```

**En CrearPictogramaActivity.kt**:

```
private fun crearPictograma() {
    // Validaciones...

    val tipoUsuario = sessionManager.obtenerTipoUsuario()
    val usuarioId = sessionManager.obtenerUsuarioId()

    // PADRE: aprobado automáticamente
    // HIJO: queda pendiente
    val aprobadoAutomaticamente = tipoUsuario == TipoUsuario.PADRE.name

    val nuevoPictograma = PictogramaSimple(
        texto = texto,
        categoria = categoriaSeleccionada!!,
        recursoImagen = iconoSeleccionado!!,
        aprobado = aprobadoAutomaticamente,  // ← CLAVE
        creadoPor = usuarioId.toString()
    )

    lifecycleScope.launch {
        repository.crearPictograma(nuevoPictograma)
```

```
        if (aprobadoAutomaticamente) {
            Toast.makeText(this@CrearPictogramaActivity,
                "Pictograma creado", Toast.LENGTH_SHORT).show()
        } else {
            Toast.makeText(this@CrearPictogramaActivity,
                "Tu pictograma está pendiente de aprobación",
                Toast.LENGTH_LONG).show()
        }
        finish()
    }
}
```

---

## 7. Flujo de Datos Completo

### 🔄 Desde la Creación hasta la Visualización

```
┌─────────────────────────────────────────────────────────┐
│ PASO 1: Usuario crea pictograma                         │
└─────────────────────────────────────────────────────────┘

CrearPictogramaActivity
  ↓ onClick(btnCrear)
  ↓ Valida campos
  ↓ Crea PictogramaSimple (aprobado según tipo de usuario)
  ↓
RoomRepository.crearPictograma(pictograma)
  ↓ Convierte: PictogramaSimple → PictogramaEntity
  ↓
PictogramaDao.insertar(entity)
  ↓ Ejecuta: INSERT INTO pictogramas VALUES (...)
  ↓
SQLite Database
  ☑ Pictograma guardado
  ↓ Flow emite cambio

┌─────────────────────────────────────────────────────────┐
│ PASO 2: PADRE gestiona pictogramas pendientes           │
└─────────────────────────────────────────────────────────┘

GestionarPictogramasActivity.onCreate()
  ↓
RoomRepository.obtenerPictogramasPendientes()
  ↓ Retorna: Flow<List<PictogramaSimple>>
  ↓
PictogramaDao.obtenerPendientesAprobacion()
  ↓ Ejecuta: SELECT * FROM pictogramas WHERE aprobado = 0
  ↓
SQLite Database
  ↓ Retorna: Flow<List<PictogramaEntity>>
  ↓ Convierte: Entity → Model (toModel())
  ↓
```

```
GestionarPictogramasActivity.collect { pictogramas ->
  adapter.submitList(pictogramas)
}
  ↓
RecyclerView muestra lista
  ↓ Usuario presiona "Aprobar"
  ↓
RoomRepository.aprobarPictograma(id)
  ↓
PictogramaDao.aprobar(id)
  ↓ Ejecuta: UPDATE pictogramas SET aprobado = 1 WHERE id = X
  ↓
SQLite Database
  ☑ Campo aprobado actualizado a true
  ↓ Flow emite cambio automáticamente
  ↓
collect { } recibe lista actualizada
  ↓ Pictograma desaparece de pendientes

┌──────────────────────────────────────────────────────────────┐
│ PASO 3: MainActivity recarga pictogramas                     │
└──────────────────────────────────────────────────────────────┘
MainActivity.onResume()
  ↓
cargarPictogramasDesdeBaseDatos()
  ↓
repository.obtenerPictogramasAprobados()
  ↓
PictogramaDao.obtenerTodosAprobadosList()
  ↓ Ejecuta: SELECT * FROM pictogramas WHERE aprobado = 1
  ↓
SQLite Database
  ↓ Retorna: List<PictogramaEntity> (incluye recién aprobado)
  ↓
viewModel.cargarPictogramasDesdeBaseDatos(pictogramas)
  ↓ Convierte: Entity → Model usando entity.toModel()
  ↓ Actualiza: StateFlow<EstadoInterfazDemo>
  ↓
lifecycleScope.launch {
    viewModel.estadoInterfaz.collect { estado ->
        pictogramaAdapter.submitList(estado.pictogramasDisponibles)
    }
}
  ↓
RecyclerView actualiza automáticamente
  ☑ Pictograma aprobado ahora visible
```

## 🔀 ViewModel y Estado Reactivo

**ViewModelDemo.kt** gestiona el estado completo de la UI:

```
data class EstadoInterfazDemo(
    val oracionActual: List<PictogramaSimple> = emptyList(),
    val pictogramasDisponibles: List<PictogramaSimple> = emptyList(),
    val todosPictogramas: List<PictogramaSimple> = emptyList(),
    val categoriaSeleccionada: Categoria? = null
) {
    val textoOracion: String
        get() = oracionActual.joinToString(" ") { it.texto }

    val puedeReproducir: Boolean
        get() = oracionActual.isNotEmpty()

    val puedeGuardar: Boolean
        get() = oracionActual.size >= 2
}

class ViewModelDemo : ViewModel() {

    private val _estadoInterfaz = MutableStateFlow(EstadoInterfazDemo())
    val estadoInterfaz: StateFlow<EstadoInterfazDemo> =
_estadoInterfaz.asStateFlow()

    // Cargar pictogramas desde BD
    fun cargarPictogramasDesdeBaseDatos(pictogramasEntity: List<PictogramaEntity>)
{
        val pictogramasSimple = pictogramasEntity.map { it.toModel() }

        _estadoInterfaz.update {
            it.copy(
                todosPictogramas = pictogramasSimple,
                pictogramasDisponibles = pictogramasSimple.take(20)
            )
        }
    }

    // Filtrar por categoría (usa datos de memoria, no Mock)
    fun seleccionarCategoria(categoria: Categoria?) {
        val todosPictogramas = _estadoInterfaz.value.todosPictogramas

        val pictogramas = if (todosPictogramas.isNotEmpty()) {
            if (categoria != null) {
                todosPictogramas.filter { it.categoria == categoria }
            } else {
                todosPictogramas.take(20)
            }
        } else {
            // Fallback a Mock si no hay datos en BD
            if (categoria != null) {
                FuenteDatosMock.obtenerPictogramasPorCategoria(categoria)
            } else {
                FuenteDatosMock.obtenerPictogramasMasUsados(20)
            }
        }
```

```
            _estadoInterfaz.update {
                it.copy(
                    categoriaSeleccionada = categoria,
                    pictogramasDisponibles = pictogramas
                )
            }
        }
    }
```

**Clave**: Todas las operaciones de filtrado ahora usan `todosPictogramas` (cargados desde BD) en lugar de `FuenteDatosMock`, garantizando que los pictogramas aprobados aparezcan correctamente.

---

# 8. Componentes Principales

## Activities

### 8.1. SplashActivity

- **Pantalla inicial** (1.5 segundos)
- Verifica si existen usuarios en BD
- Redirige a:
  - `ConfiguracionInicialActivity` si no hay usuarios
  - `SeleccionPerfilActivity` si hay usuarios

### 8.2. ConfiguracionInicialActivity

- **Primera ejecución** de la app
- Crea usuario PADRE (con PIN) y usuario HIJO
- Carga pictogramas del sistema desde `FuenteDatosMock`
- Guarda sesión del PADRE automáticamente

### 8.3. SeleccionPerfilActivity

- Muestra lista de usuarios activos
- Al seleccionar HIJO → `MainActivity`
- Al seleccionar PADRE → `PinActivity`

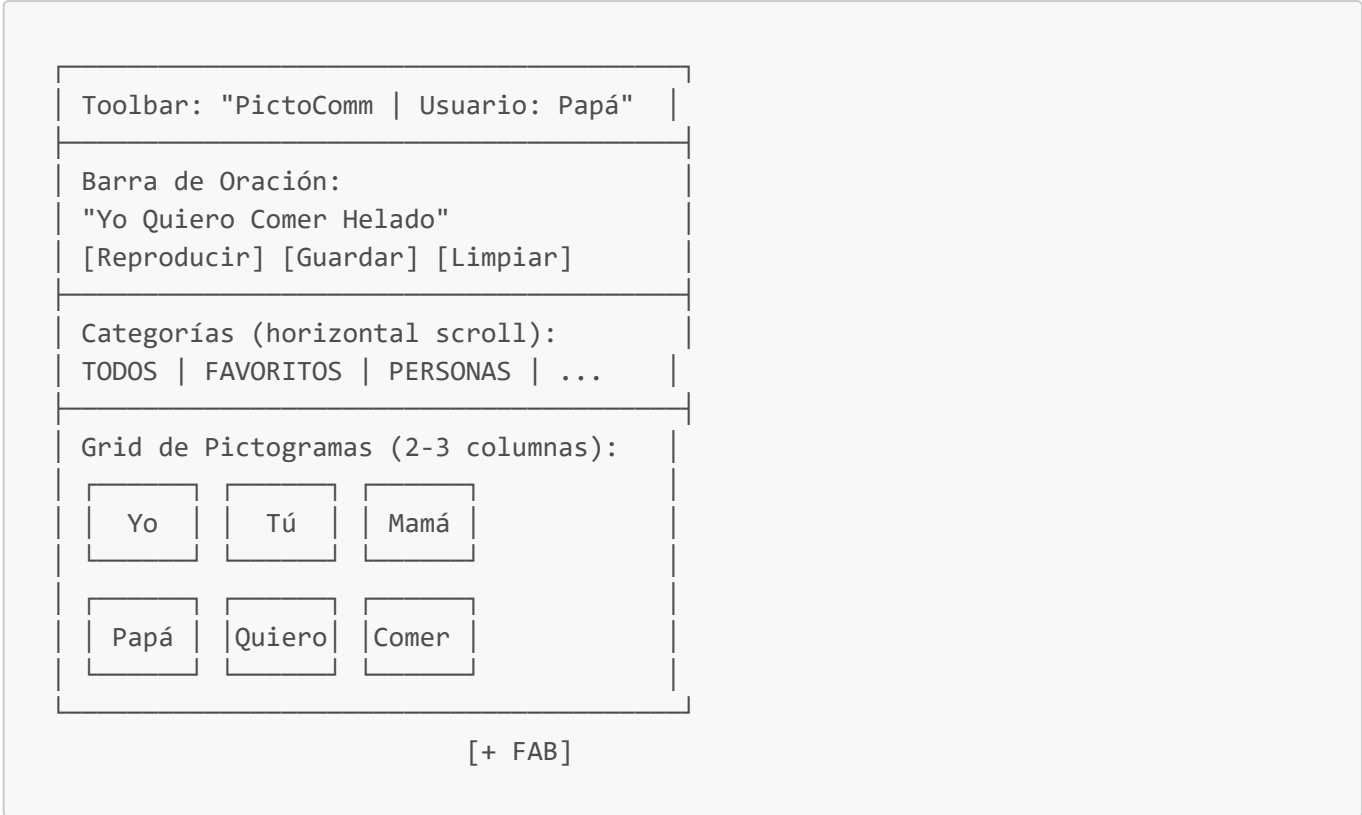
### 8.4. PinActivity

- Solicita PIN de 4 dígitos
- Valida contra BD: `UsuarioDao.verificarPin()`
- Si correcto → `MainActivity`

### 8.5. MainActivity ★

**Actividad principal de la aplicación**

**Componentes UI**:




**Menú Toolbar (solo PADRE)**:

- 🔍 Gestionar pictogramas
- 📜 Historial (próximamente)
- ℹ Información
- 👤 Cambiar usuario

**Interacciones**:

- **Tap en pictograma** → Añade a oración
- **Long-press en pictograma** → Marca/desmarca favorito
- **Tap en categoría** → Filtra pictogramas
- **Botón Reproducir** → Text-to-Speech lee la oración
- **Botón Guardar** → Guarda en BD (requiere ≥2 pictogramas)
- **FAB (+)** → Abre `CrearPictogramaActivity`

### 8.6. CrearPictogramaActivity

**Formulario de creación de pictogramas**

**Campos**:

- Campo de texto (palabra del pictograma)
- Selector de categoría (diálogo)
- Vista previa del icono seleccionado
- Botón "Seleccionar icono" (6 iconos predefinidos)
- Botón "Usar cámara" (mensaje informativo)
- Botón "Agregar pictograma"

**Lógica de aprobación**:

```
val aprobadoAutomaticamente = tipoUsuario == TipoUsuario.PADRE.name

val nuevoPictograma = PictogramaSimple(
    texto = texto,
    categoria = categoriaSeleccionada!!,
    recursoImagen = iconoSeleccionado!!,
    aprobado = aprobadoAutomaticamente,  // true para PADRE, false para HIJO
    creadoPor = usuarioId.toString()
)
```

#### 8.7. GestionarPictogramasActivity

**Gestión de pictogramas pendientes (solo PADRE)**

**Funcionalidad**:

- Lista de pictogramas con `aprobado = false`
- Muestra: texto, categoría, creador, fecha
- Botones por ítem:
  - **Aprobar** (verde) → `UPDATE aprobado = 1`
  - **Rechazar** (rojo) → `DELETE` (con confirmación)
- Empty state si no hay pendientes

### Adapters (RecyclerView)

#### 8.1. PictogramaAdapter

**Grid de pictogramas**

- DiffUtil para actualizaciones eficientes
- Click → añade a oración
- Long-click → marca favorito
- Color de fondo según categoría
- Icono de estrella si es favorito

#### 8.2. PictogramaPendienteAdapter

**Lista de pictogramas pendientes**

- Muestra información completa del pictograma
- Formatea fecha de creación
- Obtiene nombre del creador desde BD
- Botones de aprobar/rechazar

#### 8.3. CategoriaAdapter

**Barra horizontal de categorías**

- Selector visual (subrayado)
- Incluye "TODOS", "FAVORITOS" + 6 categorías
- Actualización del ítem seleccionado

#### 8.4. OracionAdapter

**Barra de oración construida**

- Lista horizontal compacta
- Números de orden (1, 2, 3...)
- Click en ítem → elimina de la oración

---

## 9. Características Avanzadas

### 🧠 Sistema Predictivo Inteligente

El ViewModel implementa un sistema de predicción que sugiere automáticamente la siguiente categoría:

**Niveles de predicción** (por prioridad):

1. **Específica por palabra**:

```
"Voy" → LUGARES
"Comer", "Beber" → COSAS
"Jugar" → COSAS
"Dormir" → LUGARES
```

2. **Contextual** (últimos 2 pictogramas):

```
PERSONA + ACCION → depende de la acción
"Yo" + "Quiero" → COSAS
"Yo" + "Voy" → LUGARES
ACCION + COSA → TIEMPO
```

3. **Básica** (solo último pictograma):

```
PERSONAS → ACCIONES
ACCIONES → COSAS
COSAS → TIEMPO
LUGARES → TIEMPO
```

**Ejemplo de flujo**:

```
Usuario toca: "Yo" (PERSONAS)
  → Sistema sugiere: ACCIONES

Usuario toca: "Quiero" (ACCIONES)
  → Sistema sugiere: COSAS (por palabra específica)

Usuario toca: "Helado" (COSAS)
  → Sistema sugiere: TIEMPO (contextual)

Oración completa: "Yo Quiero Helado Ahora"
```

### Text-to-Speech

**Configuración en MainActivity**:

```
private var tts: TextToSpeech? = null

private fun configurarTextToSpeech() {
    tts = TextToSpeech(this) { status ->
        if (status == TextToSpeech.SUCCESS) {
            tts?.setLanguage(Locale.forLanguageTag("es-ES"))
        }
    }

    btnReproducir.setOnClickListener {
        val texto = viewModel.estadoInterfaz.value.textoOracion
        tts?.speak(texto, TextToSpeech.QUEUE_FLUSH, null, null)
    }
}

override fun onDestroy() {
    tts?.shutdown()  // Liberar recursos
    super.onDestroy()
}
```

**Características**:

- Idioma: Español (es-ES)
- Reproducción: Cola simple (QUEUE_FLUSH)
- Liberación automática de recursos

### Sistema de Favoritos

**Marcar/desmarcar favorito**:

```
// UI: Long-press en pictograma
onLongClickPictograma = { pictograma ->
    viewModel.alternarFavorito(pictograma)
```

```
        Toast.makeText(this,
            if (pictograma.esFavorito) "Agregado a favoritos"
            else "Eliminado de favoritos",
            Toast.LENGTH_SHORT
        ).show()
    }

    // ViewModel: actualiza estado en memoria
    fun alternarFavorito(pictograma: PictogramaSimple) {
        val pictogramasActualizados = _estadoInterfaz.value.todosPictogramas.map {
            if (it.id == pictograma.id) {
                it.copy(esFavorito = !it.esFavorito)
            } else {
                it
            }
        }

        _estadoInterfaz.update {
            it.copy(todosPictogramas = pictogramasActualizados)
        }
    }

    // Repository: persiste en BD
    suspend fun alternarFavorito(pictogramaId: String, esFavorito: Boolean) {
        val id = pictogramaId.toLongOrNull() ?: return
        pictogramaDao.actualizarFavorito(id, esFavorito)
    }
```

**Acceso a favoritos**:

- Tap en botón "FAVORITOS" en barra de categorías
- Filtra: `pictogramas.filter { it.esFavorito }`

---

## 10. Consideraciones Técnicas

### Rendimiento

**Optimizaciones implementadas**:

1. **DiffUtil en Adapters**: Solo actualiza ítems cambiados, no toda la lista
2. **Flow con reactividad**: La UI se actualiza solo cuando cambian los datos
3. **Coroutines**: Operaciones de BD en background threads
4. **ViewHolder Pattern**: Reciclaje eficiente de vistas en RecyclerView
5. **Singleton de Database**: Una única instancia compartida
6. **StateFlow**: Emisión solo cuando el estado cambia realmente

**Ejemplo de DiffUtil**:

```
    private class DiffCallback : DiffUtil.ItemCallback<PictogramaSimple>() {
        override fun areItemsTheSame(
```

```
            oldItem: PictogramaSimple,
            newItem: PictogramaSimple
        ): Boolean {
            return oldItem.id == newItem.id  // Compara IDs
        }

        override fun areContentsTheSame(
            oldItem: PictogramaSimple,
            newItem: PictogramaSimple
        ): Boolean {
            return oldItem == newItem  // Compara contenido completo
        }
    }
```

## 🔒 Seguridad

**Puntos críticos identificados**:

1. ⚠ **PIN en texto plano**: Actualmente el PIN se guarda sin encriptar en SharedPreferences
    - **Mejora sugerida**: Usar `EncryptedSharedPreferences` o hash con BCrypt
2. ⚠ **Sin timeout de sesión**: La sesión permanece activa indefinidamente
    - **Mejora sugerida**: Implementar timeout automático para PADRE
3. ⚠ **Sin validación de inyección SQL**: Room protege automáticamente
    - ☑ Room usa prepared statements (seguro por defecto)

**Recomendaciones**:

```
// Ejemplo de encriptación de PIN (mejora futura)
import androidx.security.crypto.EncryptedSharedPreferences
import androidx.security.crypto.MasterKey

val masterKey = MasterKey.Builder(context)
    .setKeyScheme(MasterKey.KeyScheme.AES256_GCM)
    .build()

val encryptedPrefs = EncryptedSharedPreferences.create(
    context,
    "secure_prefs",
    masterKey,
    EncryptedSharedPreferences.PrefKeyEncryptionScheme.AES256_SIV,
    EncryptedSharedPreferences.PrefValueEncryptionScheme.AES256_GCM
)
```

## 🧪 Testing

**Estructura de testing recomendada**:

```
test/
├── database/
│   ├── PictogramaDaoTest.kt        # Test de DAOs
│   └── DatabaseMigrationTest.kt    # Test de migraciones
├── repository/
│   └── RoomRepositoryTest.kt       # Test de Repository
└── viewmodel/
    └── ViewModelDemoTest.kt         # Test de ViewModel

androidTest/
├── ui/
│   ├── MainActivityTest.kt          # Test de UI
│   └── CrearPictogramaTest.kt       # Test de flujo completo
└── database/
    └── DatabaseIntegrationTest.kt   # Test de integración
```

**Ejemplo de test de DAO**:

```
@RunWith(AndroidJUnit4::class)
class PictogramaDaoTest {

    private lateinit var database: PictoCommDatabase
    private lateinit var dao: PictogramaDao

    @Before
    fun setup() {
        val context = ApplicationProvider.getApplicationContext<Context>()
        database = Room.inMemoryDatabaseBuilder(context,
PictoCommDatabase::class.java)
            .allowMainThreadQueries()
            .build()
        dao = database.pictogramaDao()
    }

    @After
    fun tearDown() {
        database.close()
    }

    @Test
    fun insertarYObtenerPictograma() = runBlocking {
        val pictograma = PictogramaEntity(
            texto = "Test",
            categoria = "COSAS",
            recursoImagen = "ic_test",
            aprobado = true
        )

        val id = dao.insertar(pictograma)
        val obtenido = dao.obtenerPorId(id)
```

```
        assertEquals("Test", obtenido?.texto)
        assertTrue(obtenido?.aprobado == true)
    }

    @Test
    fun aprobarPictogramaPendiente() = runBlocking {
        // Crear pictograma no aprobado
        val pictograma = PictogramaEntity(
            texto = "Pendiente",
            categoria = "COSAS",
            recursoImagen = "ic_test",
            aprobado = false
        )
        val id = dao.insertar(pictograma)

        // Aprobar
        dao.aprobar(id)

        // Verificar
        val obtenido = dao.obtenerPorId(id)
        assertTrue(obtenido?.aprobado == true)
    }
}
```

## 📊 Migraciones de Base de Datos

**Si se agregan/modifican campos en el futuro**:

```
val MIGRATION_1_2 = object : Migration(1, 2) {
    override fun migrate(database: SupportSQLiteDatabase) {
        // Ejemplo: Agregar campo 'color' a pictogramas
        database.execSQL(
            "ALTER TABLE pictogramas ADD COLUMN color TEXT NOT NULL DEFAULT
'#000000'"
        )
    }
}

Room.databaseBuilder(context, PictoCommDatabase::class.java, "pictocomm_database")
    .addMigrations(MIGRATION_1_2)
    .build()
```

## 🔄 Sincronización Futura con Firebase

**Preparación para sincronización cloud**:

El modelo `PictogramaSimple` incluye métodos para Firebase:

```
    fun toMap(): Map<String, Any> {
        return mapOf(
            "id" to id,
            "texto" to texto,
            "categoria" to categoria.name,
            "aprobado" to aprobado,
            // ... otros campos
        )
    }

    companion object {
        fun fromMap(map: Map<String, Any>): PictogramaSimple {
            return PictogramaSimple(
                id = map["id"] as? String ?: "",
                texto = map["texto"] as? String ?: "",
                // ... otros campos
            )
        }
    }
```

**Estrategia de sincronización recomendada**:

1. Room Database como fuente de verdad local
2. Firebase Firestore para sincronización cloud
3. WorkManager para sincronización en background
4. Conflict resolution basado en timestamps

---

## Conclusión

**PictoComm2** es una aplicación Android robusta que combina:

☑ **Arquitectura limpia** (MVVM + Repository Pattern) ☑ **Persistencia local eficiente** (Room Database) ☑ **UI reactiva** (StateFlow + Coroutines) ☑ **Control parental funcional** (Sistema de aprobación) ☑ **Código bien organizado** (Separación de capas) ☑ **Escalabilidad** (Preparado para crecimiento)

### Próximas Mejoras Sugeridas

1. **Seguridad**: Encriptar PIN con `EncryptedSharedPreferences`
2. **Fotos personalizadas**: Implementar cámara y galería
3. **Sincronización cloud**: Integrar Firebase Firestore
4. **Estadísticas**: Dashboard de uso de pictogramas
5. **Exportación**: Backup/restore de datos
6. **Múltiples idiomas**: i18n para Text-to-Speech
7. **Testing**: Aumentar cobertura de tests
8. **Accesibilidad**: Mejorar TalkBack y contraste

---

**Autor**: Sistema PictoComm **Versión**: 1.0 **Fecha**: Diciembre 2025 **Tecnología**: Kotlin + Room Database + MVVM