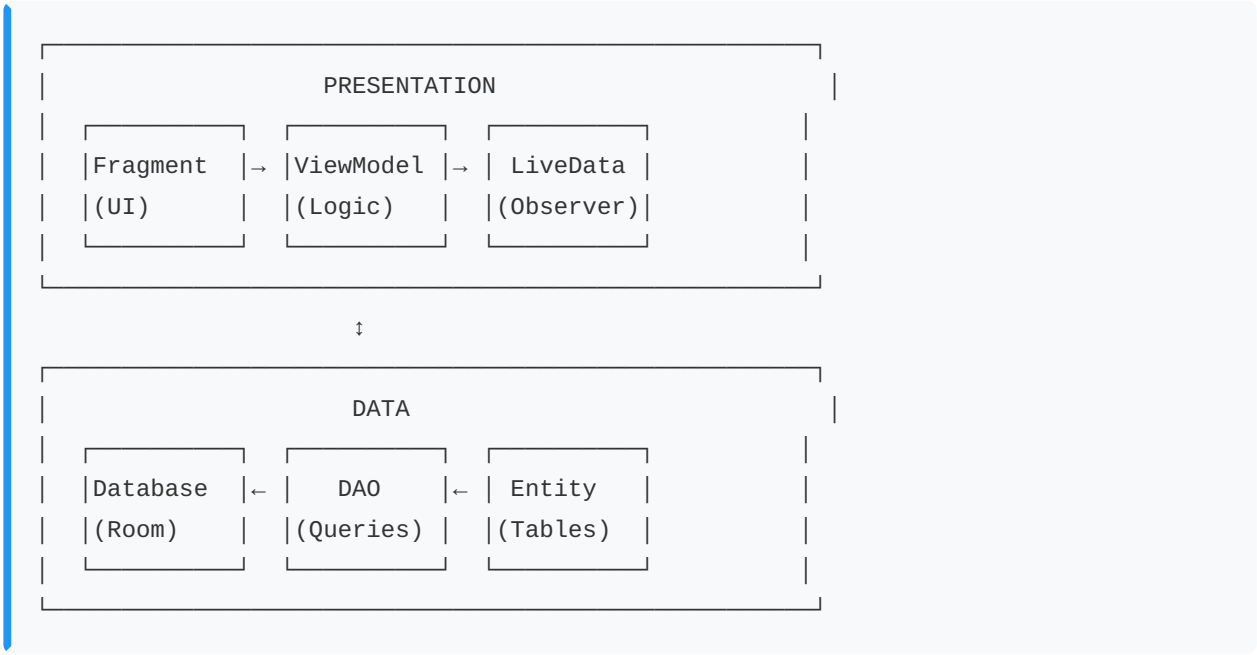
📘 Guía de Arquitectura de iTrainer

🏗️ Estructura General de la App

iTrainer sigue el patrón **MVVM (Model-View-ViewModel)** con **Android Architecture Components**:



📁 Organización de Paquetes

```
com.example.itrainer/
│
├── data/
│   ├── dao/              # Interfaces para acceso a base de datos
│   ├── database/         # Configuración de Room Database
│   ├── entities/         # Tablas de la base de datos
│   └── models/           # Modelos de datos (no persistentes)
│
├── ui/
│   ├── categories/       # Pantalla de categorías
│   ├── teams/            # Pantalla de equipos
│   ├── players/          # Pantalla de jugadores
│   ├── distribution/     # Pantalla de distribución de partidos
│   ├── history/          # Pantalla de histórico
│   ├── stats/            # Pantalla de estadísticas
│   └── rules/            # Pantalla de reglas
│
├── utils/                # Utilidades (colores, helpers)
└── MainActivity.kt       # Actividad principal con navegación
```

🗄️ Capa de Datos (DATA)

Entities (Tablas de la BD)

1. Category.kt

Define las categorías de baloncesto (Minibasket, Infantil, etc.)

```
@Entity(tableName = "categories")
data class Category(
    val id: Int,
    val name: String,              // "Minibasket", "Infantil y PreInf 2ª"
    val periodsCount: Int,         // 6 períodos para Minibasket, 4 para Infantil
    val playersPerPeriod: Int,     // 5 jugadores en cancha
    val minPeriodsPerPlayer: Int,  // Mínimo de períodos que debe jugar cada jugador
    val maxPeriodsPerPlayer: Int,  // Máximo de períodos que puede jugar
    val minPlayers: Int,           // Mínimo de jugadores en el equipo
    val maxPlayers: Int            // Máximo de jugadores en el equipo
)
```

Uso: Define las reglas específicas de cada categoría de baloncesto.

---

2. Team.kt

Equipos asociados a una categoría

```
@Entity(tableName = "teams")
data class Team(
    val id: Int,
    val name: String,        // "Real Madrid", "FC Barcelona"
    val categoryId: Int      // Referencia a Category
)
```

Relación: Un equipo pertenece a UNA categoría.

---

3. Player.kt

Jugadores de un equipo

```
@Entity(tableName = "players")
data class Player(
    val id: Int,
    val teamId: Int,    // Referencia a Team
    val name: String,   // "Juan García"
    val number: Int     // Dorsal: 7
)
```

Relación: Un jugador pertenece a UN equipo.
Cascade: Si se elimina un equipo, se eliminan sus jugadores.

---

4. GameDistribution.kt

Distribuciones de partidos guardadas

```
@Entity(tableName = "game_distributions")
data class GameDistribution(
    val id: Int,
    val teamId: Int,
    val date: Date,
    val gameDate: String,          // "20/11/2024"
    val opponent: String,           // "FC Barcelona"
    val distributionJson: String    // JSON con toda la distribución
)
```

Contenido del JSON:

```
{
  "periods": {
    "1": [{"id":1, "name":"Juan", "number":7}, ...],
    "2": [...],
    ...
  },
  "categoryId": 2,
  "substitutions": {
    "4007": {"playerId":7, "isOut":true, "isSubstitute":false}
  }
}
```

---

5. PlayerStats.kt

Estadísticas de jugadores por temporada

```
@Entity(tableName = "player_stats")
data class PlayerStats(
    val id: Int,
    val playerId: Int,
    val seasonYear: Int,              // 2024
    val totalGames: Int,              // 10 partidos
    val totalPeriods: Int,            // 30 períodos totales
    val gamesAsStarter: Int,          // 8 partidos como titular
    val gamesAsSubstitute: Int,       // 2 partidos como suplente
    val averagePeriodsPerGame: Float  // 3.0 períodos/partido
)
```

Cálculo automático: Se actualiza cada vez que guardas una distribución.

DAOs (Acceso a datos)

Cada entidad tiene su DAO que define las operaciones de base de datos:

```
@Dao
interface PlayerDao {
    @Query("SELECT * FROM players WHERE teamId = :teamId")
    suspend fun getPlayersByTeam(teamId: Int): List<Player>

    @Insert
    suspend fun insertPlayer(player: Player)

    @Update
    suspend fun updatePlayer(player: Player)

    @Delete
    suspend fun deletePlayer(player: Player)
}
```

Operaciones típicas: - `getAllX()` - Obtener todos - `getXById(id)` - Obtener por ID - `insert()` - Insertar - `update()` - Actualizar - `delete()` - Eliminar

---

ITrainerDatabase.kt

Configuración central de Room Database

```
@Database(
    entities = [Category, Team, Player, GameDistribution, PlayerStats],
    version = 6  // IMPORTANTE: Incrementar al cambiar estructura
)
abstract class ITrainerDatabase : RoomDatabase() {
    abstract fun categoryDao(): CategoryDao
    abstract fun teamDao(): TeamDao
    abstract fun playerDao(): PlayerDao
    abstract fun gameDistributionDao(): GameDistributionDao
    abstract fun playerStatsDao(): PlayerStatsDao
}
```

⚠️ **IMPORTANTE:** Cada vez que modificas una entidad (añadir/eliminar columnas): 1. Incrementa `version` 2. Desinstala la app 3. Reinstala para recrear la BD

🎨 Capa de Presentación (UI)

Patrón Fragment + ViewModel + Adapter

Cada pantalla sigue esta estructura:

```
Fragment (Vista)
    ↓
ViewModel (Lógica)
    ↓
LiveData (Observables)
    ↓
Adapter (Lista de items)
```

---

1. CategoriesFragment → Listado de categorías

Flujo:

```
Usuario abre app
    ↓
CategoriesFragment se carga
    ↓
CategoriesViewModel carga categorías desde BD
    ↓
CategoriesAdapter muestra lista con colores
    ↓
Usuario hace click en categoría
    ↓
Navega a TeamsFragment
```

Archivos clave: - `CategoriesFragment.kt` - Vista - `CategoriesViewModel.kt` - Lógica (carga categorías) - `CategoriesAdapter.kt` - Renderiza lista - `item_category.xml` - Diseño de cada tarjeta

---

2. TeamsFragment → Equipos de una categoría

Flujo:

```
Recibe categoryId por navegación
    ↓
TeamsViewModel carga equipos de esa categoría
    ↓
Muestra lista de equipos
    ↓
Usuario click en equipo → Distribución
Usuario click en "Jugadores" → PlayersFragment
```

---

3. PlayersFragment → Gestión de jugadores

Funcionalidades: - ➕ Añadir jugador (dorsal + nombre) - ✏️ Editar jugador - 🗑️ Eliminar jugador - 📊 Ver estadísticas del equipo

Validaciones: - Dorsal debe ser numérico - Nombre no puede estar vacío

---

4. DistributionFragment ⭐ (La más compleja)

Responsabilidades principales:

A) Gestión de jugadores disponibles

```
// Chips expandibles para marcar quién juega
binding.availablePlayersGroup // ChipGroup con todos los jugadores
```

B) Grid de distribución

```
┌─────────┬────┬────┬────┬────┐
│ Jugador│ P1 │ P2 │ P3 │ P4 │
├─────────┼────┼────┼────┼────┤
│ 7-Juan │ ✓  │ ✓  │    │ X  │ ← Azul=titular, X=sale
│ 8-Pedro│    │ ✓  │ ✓  │ ✓  │
│ 9-Luis │ ✓  │    │ ✓  │ ✓  │ ← Verde=sustituto
└─────────┴────┴────┴────┴────┘
```

C) Sistema de sustituciones (Infantil/PreInf)

```
// Último período puede tener:
- 5 titulares (azul) ✓
- Sustitutos adicionales (verde) ↑
- Marcas de salida (X roja) en titulares
```

Clave: `SubstitutionInfo`

```
data class SubstitutionInfo(
    val playerId: Int,
    val isOut: Boolean,        // Sale (X roja)
    val isSubstitute: Boolean  // Entra (verde)
)
```

Almacenamiento:

```
// Key = period * 1000 + playerId
// Ejemplo: Período 4, Jugador 7 → Key = 4007
val substitutions = mapOf(
    "4007" to SubstitutionInfo(7, isOut=true, isSubstitute=false)
)
```

D) Validación en tiempo real

```
validateDistribution() {
    // Verifica:
    - 5 jugadores por período
    - Mínimo/máximo de períodos por jugador
    - Reglas específicas de la categoría
}
```

E) Guardado

```
saveDistribution(gameDate, opponent) {
    1. Serializa distribución a JSON
    2. Guarda en GameDistribution
    3. Actualiza estadísticas de todos los jugadores
}
```

---

5. HistoryFragment → Histórico de partidos

Muestra: - Lista de distribuciones guardadas - Color según categoría - Click → Ver detalles - Long click → Eliminar

---

6. DistributionDetailsFragment → Visualización de partido guardado

Renderiza:

```
Grid completo con:
- Titulares en azul
- Sustitutos en verde
- Salidas con X roja
```

Carga desde JSON:

```
loadDistribution() {
    val json = distribution.distributionJson
    val model = Json.decodeFromString<DistributionModel>(json)
    // Renderiza períodos + sustituciones
}
```

---

7. PlayerStatsFragment → Estadísticas del equipo

Muestra por jugador: - Partidos totales (titular/suplente) - Períodos totales jugados - Media de períodos por partido

Ordenado por: Total de períodos (descendente)

🎨 Sistema de Colores por Categoría

CategoryColors.kt (Utilidad centralizada)

```
object CategoryColors {
    fun getCategoryColor(categoryName: String?): Int {
        return when (categoryName?.lowercase()) {
            "minibasket" -> R.color.category_minibasket      // 🟠 Naranja
            "benjamín" -> R.color.category_benjamin          // 🟢 Verde
            "infantil y preinf 2ª" -> R.color.category_infantil // 🔵 Azul
            "cadete" -> R.color.category_cadete              // 🟣 Morado
            else -> R.color.category_default                 // ⚪ Gris
        }
    }
}
```

Usado en: - Tarjetas de categorías - Tarjetas de histórico - Cualquier referencia visual a categorías

---

🧭 Navegación (Navigation Component)

nav_graph.xml

Define todas las pantallas y transiciones:

```
<navigation>
    <fragment id="categoriesFragment" />
        ↓ (selecciona categoría)
    <fragment id="teamsFragment">
        <argument name="categoryId" />
    </fragment>
        ↓ (selecciona equipo)
    <fragment id="distributionFragment">
        <argument name="teamId" />
        <argument name="categoryId" />
    </fragment>
</navigation>
```

Navegación típica:

```
findNavController().navigate(
    R.id.destinoFragment,
    bundleOf("key" to value)
)
```

🔄 Flujo Completo: Crear Distribución

```
1. Usuario selecciona Categoría
   ↓
2. Usuario selecciona Equipo
   ↓
3. DistributionFragment carga:
   - Jugadores del equipo
   - Reglas de la categoría (períodos, min/max)
   ↓
4. Usuario marca jugadores disponibles
   ↓
5. Usuario asigna jugadores a períodos (click en checkboxes)
   ↓
6. Sistema valida en tiempo real:
   - 5 jugadores/período
   - Mínimo/máximo por jugador
   - Reglas especiales de categoría
   ↓
7. Usuario hace long-press en último período (Infantil):
   - Marca salidas (X roja)
   - Click adicional añade sustitutos (verde)
   ↓
8. Usuario guarda distribución:
   a) Crea DistributionModel (JSON)
   b) Guarda en GameDistribution
   c) Actualiza PlayerStats de cada jugador:
      - Incrementa totalGames
      - Incrementa totalPeriods
      - Marca si fue titular (jugó P1) o suplente
   ↓
9. Navega a HistoryFragment (distribución guardada)
```

📊 Sistema de Estadísticas

Actualización automática al guardar

```
updatePlayerStatistics() {
    val firstPeriodPlayers = distribution[1] // Titulares

    distribution.allPlayers.forEach { player ->
        val periodsPlayed = countPeriods(player)
        val wasStarter = player in firstPeriodPlayers

        stats.update(
            totalGames++,
            totalPeriods += periodsPlayed,
            gamesAsStarter++ if wasStarter,
            gamesAsSubstitute++ if !wasStarter
        )
    }
}
```

Criterio: - **Titular**: Jugó el primer período - **Suplente**: No jugó el primer período

---

🛠️ Modificar Funcionalidades Clave

Añadir nueva categoría:

1. Ir a `CategoriesViewModel.kt`:

```
val nuevaCategoria = Category(
    name = "Nueva Categoría",
    periodsCount = 4,
    playersPerPeriod = 5,
    minPeriodsPerPlayer = 1,
    maxPeriodsPerPlayer = 4,
    minPlayers = 8,
    maxPlayers = 12
)
categoryDao.insertCategory(nuevaCategoria)
```

1. Añadir color en `colors.xml`:

```
<color name="category_nueva">#FF5722</color>
```

1. Actualizar `CategoryColors.kt`:

```
"nueva categoría" -> R.color.category_nueva
```

Modificar reglas de validación:

Ir a `DistributionViewModel.kt` → función `applySpecialRules()`:

```
when (cat.name) {
    "Tu Categoría" -> {
        // Tus reglas específicas
        if (condicion) {
            messages.add("Mensaje de error")
        }
    }
}
```

Cambiar visualización de grid:

Ir a `DistributionFragment.kt` → función `setupGrid()`:

```
// Modificar tamaño de celdas
R.dimen.grid_cell_size // en dimens.xml

// Cambiar colores
R.color.colorPrimary // Titular (azul)
R.color.substitute_green // Sustituto (verde)
```

Añadir campo a estadísticas:

1. Actualizar `PlayerStats.kt`:

```
data class PlayerStats(
    // ... campos existentes
    val nuevoCampo: Int = 0
)
```

1. Incrementar versión BD en `ITrainerDatabase.kt`:

```
version = 7  // Anterior era 6
```

1. Actualizar `updatePlayerStatistics()` en `DistributionViewModel.kt`
2. **Desinstalar y reinstalar app**

⚠️ Puntos Críticos a Recordar

1. Versión de Base de Datos

```
// SIEMPRE incrementar cuando cambies estructura de tablas
@Database(..., version = X)
```

2. Deserialización de JSON

```
// Usar ignoreUnknownKeys para compatibilidad
Json { ignoreUnknownKeys = true }
```

3. IDs únicos de sustituciones

```
// Key = period * 1000 + playerId
val key = period * 1000 + playerId
```

4. Actualización de estadísticas

Se ejecuta automáticamente en `saveDistribution()` - no olvidar añadir lógica si cambias estructura.

🎯 Resumen de Responsabilidades

Componente	Responsabilidad
Fragment	Renderiza UI, captura eventos del usuario
ViewModel	Lógica de negocio, carga datos, validaciones
Adapter	Convierte lista de datos en items visuales
DAO	Operaciones CRUD en base de datos
Entity	Define estructura de tablas
Model	Datos temporales (no se guardan en BD)

📝 Checklist para Modificaciones

✅ **Antes de tocar código:** 1. ¿Qué pantalla/funcionalidad afecta? 2. ¿Necesito cambiar la BD? → Incrementar versión 3. ¿Afecta a validaciones? → Revisar `validateDistribution()` 4. ¿Cambia la visualización? → Revisar Fragment + XML

✅ **Después de modificar:** 1. Clean + Rebuild Project 2. Si cambió BD: Desinstalar app 3. Probar flujo completo 4. Verificar que estadísticas se actualizan correctamente

---

iTrainer - Guía de Arquitectura
Versión 1.0 - Noviembre 2024
Desarrollado por Mikel

¡Esta guía te ayudará a entender y modificar iTrainer con confianza! 🚀🏀