Capas de Arquitectura: Presentación, Dominio y Datos

Visión general

La arquitectura de PAE separa responsabilidades en tres capas claramente definidas. Cada capa comunica con la siguiente a través de interfaces y contratos bien definidos.

Capa de Presentación

Responsabilidades

- Renderizar la UI
- Capturar interacciones del usuario
- Mostrar estado actual
- Delegar acciones al dominio
- NO contiene lógica de negocio

Machine - Presentación

Ubicación: `Machine/src/main/java/co/ada/paemachine/`

Componentes:

- **Screens**: pantallas principales
  - `LaboratoryScreen`: captura de entregas
  - `ShiftSelectionScreen`: selección de jornada
- **Views/Composables**: componentes reutilizables
- **ViewModels**: estado de UI

Flujo típico:

```
Button("Capturar") → ViewModel.startCapture() → Domain
      ↑
      └── ViewModel.stateFlow observa cambios
```

RutaPAE - Presentación

Ubicación: `RutaPAE/src/main/java/co/ada/rutapae/`

Componentes:

- **Screens**:

    - `MachineListScreen`: lista de máquinas
    - `MachineDetailScreen`: detalle de máquina
    - `MachineConfigurationScreen`: configuración
- **Components**: tarjetas, listas, indicadores
- **Workers**: tareas en segundo plano (WorkManager)

Flujo típico:

```
Button("Sincronizar") → enqueueSync() → DeliverySyncWorker → Domain
     ↑
     └── DeliverySyncUiEmitter observa progreso
```

Capa de Dominio

Responsabilidades

- Implementar casos de uso
- Aplicar reglas de negocio
- Orquestar flujos complejos
- Coordinar entre presentación y datos
- Manejar reintentos y errores

Machine - Dominio

Ubicación: `MachineDomain/src/main/java/co/ada/domain/`

Componentes clave:

StateManager

```
Orquesta la máquina de estados de entregas
├── WaitingForWeight
├── CaptureImages
├── CaptureFace
├── ComparingWeights
├── GenerateEmbedding
├── VerifyInDatabase
├── SaveDelivery
└── WaitForWeightRemoved
```

P2PManager

```
Gestiona sincronización P2P
├── connect(peer)
```

```
├── syncDeliveries()
├── getMachineConfiguration()
└── updateMachineConfiguration()
```

Services

- `SyncDeliveries`: subeentregas al backend HTTP
- `SyncBeneficiaries`: descarga beneficiarios
- `SyncMachineEnrollment`: sincroniza jornadas
- `HealthCheck`: verifica estado del sistema

Hardware

- `ScaleManager`: interfaz con balanza
- `Camera2Service`: captura de imágenes
- `BoardLedManager`: indicadores LED

RutaPAE - Dominio

Ubicación: `RutaPAEDomain/src/main/java/co/ada/rutapaedomain/`

Componentes clave:

P2PManager

```
Gestiona conexión P2P con máquinas
├── discoverableMachineIds()
├── discoveredMachineCandidates()
├── connectMachine(id)
├── connectMachineHotspot(ssid)
└── syncDeliveriesFromMachine()
```

DomainManager

```
Inicializa y coordina servicios
├── init(context)
├── close(context)
└── p2pManager acceso
```

Capa de Datos

Responsabilidades

- Acceso a persistencia local
- Operaciones CRUD
- Mapeo de entidades
- Caché y estrategias de consistencia
- NO contiene lógica de negocio

Machine - Datos

Ubicación: `MachineData/src/main/java/co/ada/data/`

Repository

```
class Repository {
    fun getDeliveryById(id: Long): Delivery?
    fun saveDelivery(delivery: Delivery)
    fun getAllBeneficiaries(): List<Beneficiary>
    fun createBeneficiary(beneficiary: Beneficiary)
    fun getMachineEnrollmentShift(id: Long): MachineEnrollmentShift?
}
```

Services

- `DeliveryService`: CRUD de entregas
- `BeneficiaryService`: CRUD de beneficiarios
- `MachineEnrollmentService`: CRUD de jornadas

Entidades

```
Delivery
├── weight
├── beneficiaryPhotoPath (ruta local)
├── alimentPhotoPath (ruta local)
├── similarity (0-1)
├── processTimeMs
└── serverDeliveryId

Beneficiary
├── embedding (512 dimensiones)
├── name
├── enrollmentId
├── remoteBeneficiaryId
└── lastRecognitionAtMs
```

```
MachineEnrollmentShift
├── campusId / campusName
├── modalityId / modalityTypeName
├── shiftId / shiftName
└── gradeId / gradeName
```

Base de datos

- **Tipo**: SQLite + VectorDB ORM
- **Índices**: índice vectorial para embeddings
- **Ubicación**: `/data/data/co.ada.paemachine/database.db`

RutaPAE - Datos

Ubicación: `RutaPAEData/src/main/java/co/ada/rutapaedata/`

Repository

```kotlin
class Repository {
    fun getMachineById(id: Long): Machine?
    fun getAllMachines(): List<Machine>
    fun saveDelivery(delivery: Delivery)
    fun getDeliveriesByMachineId(machineId: Long): List<Delivery>
}
```

Services

- `MachineService`: gestión de máquinas locales
- `DeliveryService`: gestión de entregas sincronizadas

Entidades

```
Machine
├── machineId (identificador único)
├── name
├── machineStatus (Active/Inactive/Error)
├── unsyncedDeliveries
└── localSyncedDeliveries

Delivery
├── weight
├── latitude / longitude (GPS)
├── benefitPhoto (URL remota)
├── beneficiaryPhoto (URL remota)
├── idLocalDeliveryT (ID en máquina)
```

```
├── machineId (FK)
└── synchronized (bandera)
```

Base de datos

- **Tipo**: SQLite + VectorDB ORM
- **Ubicación**: `/data/data/co.ada.rutapae/database.db`

Flujo de datos completo

Escenario: Entrega en Machine

```
1. Usuario presiona "Capturar"
   └─ Presentación (LaboratoryScreen)

2. ViewModel delega a Domain
   └─ StateManager.init()

3. StateManager ejecuta WaitingForWeight
   └─ espera input de balanza
4. Estado avanza a CaptureImages
   └─ Camera2Service captura foto
5. Estado avanza a GenerateEmbedding
   └─ ComputerVision genera embedding 512d
6. Estado avanza a VerifyInDatabase
   └─ Data busca en base de datos local
   └─ BeneficiaryService.findSimilar()
7. Estado avanza a SaveDelivery
   └─ DeliveryService.saveDelivery()
   └─ Capa de Datos persiste en SQLite
8. UI observa cambios via StateNameEmitter
   └─ muestra progreso al usuario
```

Escenario: Sincronización en RutaPAE

```
1. Usuario presiona "Sincronizar"
   └─ Presentación (MachineListScreen)

2. WorkManager enqueue("DeliverySyncWorker")
   └─ Presentación (UI)
3. Worker inicia DomainManager
   └─ se inicializa P2PManager
   │
   └─ Domain P2P Manager
4. P2PManager.syncDeliveriesFromMachine()
```

```
   └─ consulta P2P a máquina remota
   │
   └─ Domain P2P Manager
5. GET /p2p/machine/deliveries/1/1
   └─ recibe P2PDeliveryData
   │
   └─ Domain P2P Manager
6. DeliveryService.create(delivery)
   └─ persiste entrega sincronizada
   │
   └─ Datos (RutaPAEData)
7. DeliverySyncUiEmitter emite estado
   └─ UI observa y muestra progreso
   │
   └─ Presentación (RutaPAE)
```

Comunicación entre capas

De Presentación a Dominio

```
// Machine
Button("Capturar") {
    // UI llama Domain
    stateManager.init(context)
}

// RutaPAE
Button("Sincronizar") {
    // UI delega a Worker
    DeliverySyncScheduler.enqueue(context, machineId)
}
```

De Dominio a Datos

```
// Machine Domain
state.run(
    next = {
        // Domain accede a datos
        val beneficiaries = BeneficiaryService.findSimilar(embedding)
        DeliveryService.save(delivery)
    }
)

// RutaPAE Domain
suspend fun syncDeliveriesFromMachine(machineId: Long) {
    val machine = MachineService.getById(machineId)
    val deliveries = p2pGestor.connection.peer.get(...)
```

```
    DeliveryService.create(deliveries)
}
```

De Dominio a Presentación

```
// Machine: mediante Emitters
StateNameEmitter.emit("CapturingFace") // UI observa

// RutaPAE: mediante WorkManager + Emitters
DeliverySyncUiEmitter.update(state) // UI observa
```

Inversión de dependencias

Las capas siempre se comunican hacia abajo:

```
Presentación
    ↓ (usa)
Dominio
    ↓ (usa)
Datos
```

No hay dependencia inversa. La comunicación hacia arriba ocurre mediante:

- **Callbacks**: `next()`, `retry()` en `State.run()`
- **Emitters**: `StateFlow` y publicación de eventos
- **WorkManager**: delega desde Presentación a Worker

Inyección de dependencias

Machine

- **Dagger2/Hilt**
- Inyección automática en Activities/ViewModels
- Scope: Activity, Singleton

RutaPAE

- **Inyección manual**
- `DomainManager` singleton
- Servicios accesibles globalmente

MachineDomain / RutaPAEDomain

- **Inyección por constructor**
- Dependencias pasadas explícitamente

Ventajas de esta arquitectura

- **Testabilidad**: cada capa se prueba independientemente
- **Mantenibilidad**: responsabilidades claras
- **Reusabilidad**: módulos reutilizables en otros proyectos
- **Escalabilidad**: fácil agregar nuevas funcionalidades
- **Desacoplamiento**: cambios en una capa no afectan otras