Arquitectura General del Proyecto PAE

Visión macro

PAE utiliza una arquitectura multicapa basada en módulos independientes que comunican a través de contratos claramente definidos. El proyecto sigue patrones de clean architecture con separación de responsabilidades entre presentación, dominio y datos.

Estructura de módulos

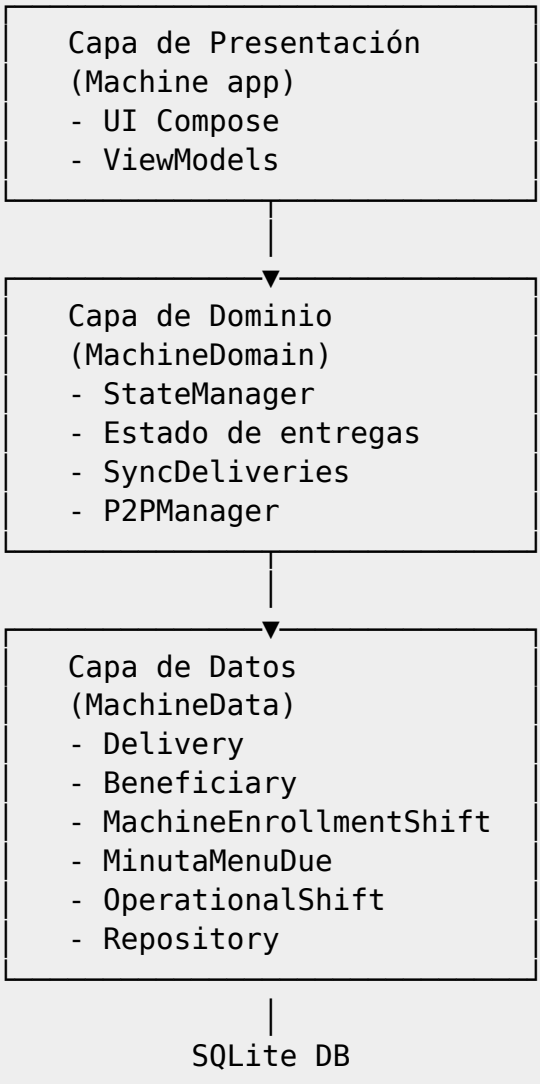
```
PAE (root)
├── Machine (app)              # Aplicación Android de máquina
├── RutaPAE (app)              # Aplicación Android de operador
│
├── MachineDomain              # Lógica de dominio de máquina
├── MachineData                # Acceso a datos de máquina
│
├── RutaPAEDomain              # Lógica de dominio de ruta
├── RutaPAEData                # Acceso a datos de ruta
│
├── Contract                   # Contrato P2P compartido
├── Core                       # Utilidades compartidas
├── DirectLink                 # Implementación P2P
│
├── ComputerVision             # Modelos de visión por computadora
├── VectorialDB                # Motor de base de datos vectorial
└── gradle/                    # Configuración de dependencias
```

Módulos principales

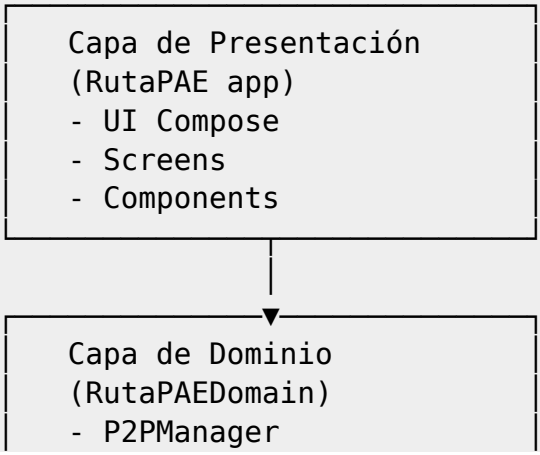
Módulo	Tipo	Responsabilidad
`Machine`	Application	UI/UX de máquina, orquestación de flujo
`MachineDomain`	Library	Lógica de entrega, hardware, sincronización
`MachineData`	Library	Persistencia local, entidades, servicios
`RutaPAE`	Application	UI/UX de operador, gestión de máquinas
`RutaPAEDomain`	Library	Gestión P2P, lógica de sincronización remota
`RutaPAEData`	Library	Persistencia local, API de máquinas
`Contract`	Library	Modelos P2P, rutas, topics compartidos
`DirectLink`	Library	Implementación Wi-Fi Direct / hotspot
`VectorialDB`	Library	ORM con índices vectoriales para embeddings
`ComputerVision`	Library	Modelos IA para reconocimiento facial
`Core`	Library	Interfaces y utilidades de log

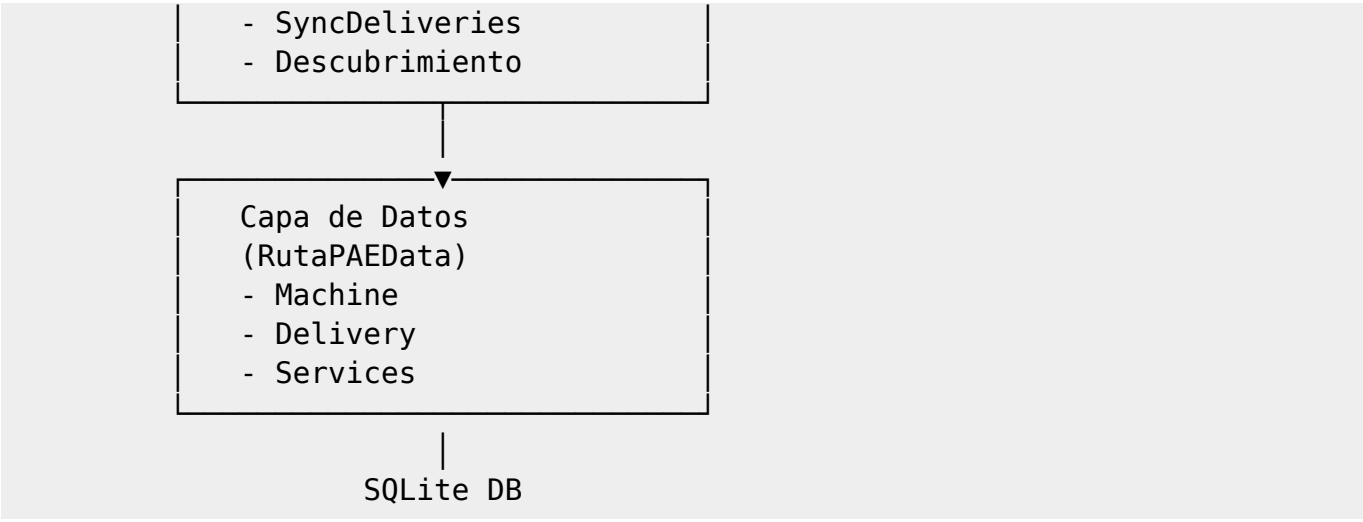
Arquitectura por capas

Machine



RutaPAE





Flujo de comunicación entre capas

Machine: Captura de entrega

1. **UI** → usuario solicita captura
2. **Data** → se resuelve la jornada operativa visible del día con fallback a `MachineEnrollmentShift` cuando falta `MinutaMenuDue` para una modalidad válida
3. **Domain** → StateManager ejecuta WaitingForWeight
4. **Data** → Beneficiary consulta base de datos
5. **Domain** → GenerateEmbedding crea vector 512d
6. **Data** → Delivery se guarda en base de datos

RutaPAE: Sincronización de entregas

1. **UI** → usuario selecciona máquina y solicita sincronización
2. **Domain** → P2PManager conecta vía P2P
3. **Domain** → syncDeliveriesFromMachine descarga entregas
4. **Data** → Delivery se almacena localmente
5. **UI** → muestra progreso y resultado

Responsabilidades por capa

Presentación (Machine / RutaPAE)

- Renderizar UI con Jetpack Compose
- Capturar interacciones del usuario
- Mostrar la card de sincronización con el mismo conteo de jornadas visibles que la lista operativa
- Mostrar estados vía ViewModels y StateFlow
- No contiene lógica de negocio

Dominio (MachineDomain / RutaPAEDomain)

- Implementar casos de uso
- Orquestar flujos complejos
- Coordinar entre Data y Presentación
- Manejar errores y reintentos

Datos (MachineData / RutaPAEData)

- Acceso a repositorios locales
- Operaciones CRUD
- Caché y persistencia
- Mapeo de entidades
- Proyecciones operativas para UI cuando la entidad técnica no coincide con la entidad visible del negocio

Comunicación inter-módulos

A través de interfaces comunes

- `Contract`: todos los módulos usan modelos P2P
- `Core`: interfaz para logging y utilidades

A través de emisores (Emitters)

- `StateNameEmitter`: cambios de estado
- `DeliverySyncUiEmitter`: progreso de sincronización
- `SyncRunningEmitter`: estado global de sincronización

A través de inyección de dependencias

- Dagger2 / Hilt (Machine)
- Inyección manual (RutaPAE Domain)

Patrones clave

State Management

- `StateManager`: máquina de estados para entregas (Machine)
- `StateFlow`: estado reactivo (Kotlin Coroutines)

Repository Pattern

- `MachineService`: acceso a Machine local
- `DeliveryService`: acceso a Delivery local
- Abstraen acceso a datos

Observer Pattern

- Emisores: para distribución de eventos
- Flow: para observar cambios

Factory Pattern

- Creación de estados en StateManager
- Creación de P2PManager en DomainManager

Dependencias entre módulos

```
Machine                 RutaPAE
  ├── MachineDomain       ├── RutaPAEDomain
  ├── MachineData         ├── RutaPAEData
  ├── Contract            ├── Contract
  └── Core                └── Core

MachineDomain
  ├── MachineData
  ├── Contract
  ├── DirectLink
  └── Core

RutaPAEDomain
  ├── RutaPAEData
  ├── Contract
  ├── DirectLink
  └── Core

MachineData / RutaPAEData
  ├── VectorialDB
  ├── Contract
  └── Core

DirectLink
  └── Contract

VectorialDB
  └── (sin dependencias PAE)
```

```
ComputerVision
  └── (sin dependencias PAE)
```

Versiones de compilación

- **API mínima**: 24 (Android 7.0)
- **API objetivo**: 36 (Android 15)
- **JVM target**: 17
- **Lenguaje**: Kotlin

Librerías principales

- **Jetpack Compose**: UI declarativa
- **Coroutines**: asincronía
- **Dagger2/Hilt**: inyección de dependencias (Machine)
- **Fuel**: cliente HTTP
- **Kotlinx Serialization**: serialización JSON
- **Camera2 API**: captura de imágenes
- **WorkManager**: tareas en segundo plano

Convenciones de nomenclatura

- **Paquetes**: `co.ada.<modulo>.xxx.xxx`
- **Servicios**: `*Service` (e.g., `MachineService`)
- **Managers**: `*Manager` (e.g., `StateManager`)
- **Emisores**: `*Emitter` (e.g., `DeliveryEmitter`)
- **Estados**: `*State` (e.g., `P2PMachineStatus`)
- **Data classes**: nombres directos (e.g., `Delivery`, `Machine`)