Guía de referencia para desarrolladores

Acceso rápido a componentes clave

Machine: Flujo de captura de entrega

Archivo de entrada:
`Machine/src/main/java/co/ada/paemachine/screens/LaboratoryScreen.kt`

```
// Usuario presiona botón
Button("Capturar") {
    viewModel.startCapture()
}

// ViewModel delega
fun startCapture() {
    stateManager.init(context) // Inicia máquina de estados
}
```

Máquina de estados: `MachineDomain/src/main/java/co/ada/domain/state/`

```
StateManager.kt
├── init() : Boolean
├── cycle() : StateResult
├── getCurrentState(): String
└── states.toMutableList()
    ├── WaitingForWeightState.kt
    ├── CaptureImagesState.kt
    ├── CaptureFaceState.kt
    ├── ComparingWeightsState.kt
    ├── GenerateEmbeddingState.kt
    ├── VerifyInDatabaseState.kt
    ├── SaveDeliveryState.kt
    └── WaitForWeightRemovedState.kt
```

Cambios de estado emitidos: `MachineDomain/src/main/java/co/ada/domain/emitters/`

```
StateNameEmitter.emit(currentState) // "CapturingFace"
StateMessageEmitter.emit(message)   // "Por favor espere..."
```

UI observa en Composable:

```
LaunchedEffect {
    StateNameEmitter.collect { stateName ->
        updateUI(stateName)
    }
```

```
}
```

RutaPAE: Sincronización de máquinas

Punto de entrada:
RutaPAE/src/main/java/co/ada/rutapae/screens/MachineListScreen.kt

```
Button("Sincronizar") {
    onSyncClicked() // enqueue worker
}

private fun onSyncClicked() {
    DeliverySyncScheduler.enqueue(
        context,
        machineId = 1L,
        machineDatabase = machineForeignId
    )
}
```

Worker: RutaPAE/src/main/java/co/ada/rutapae/workers/DeliverySyncWorker.kt

```
class DeliverySyncWorker : CoroutineWorker() {
    override suspend fun doWork(): Result {
        // Inicializa dominio
        val initialized = DomainManager.init(context)

        // Accede a P2PManager
        val manager = DomainManager.p2pManager ?: return Result.retry()

        // Sincroniza entregas
        val result = manager.syncDeliveriesFromMachine(
            machineId,
            machineDatabaseId
        )

        return if (result.success) Result.success() else Result.retry()
    }
}
```

Dominio: RutaPAEDomain/src/main/java/co/ada/rutapaedomain/

```
DomainManager.kt (singleton)
├── init(context): Boolean
├── close(context): Boolean
└── p2pManager: P2PManager

P2PManager.kt
```

```
├── discoverableMachineIds(): Set<String>
├── discoveredMachineCandidates(): Set<P2PPeer>
├── connectMachine(machineId): P2PGestor?
├── connectMachineHotspot(ssid): P2PGestor?
└── syncDeliveriesFromMachine(): SyncDeliveriesResult
```

Persistencia: RutaPAEData/src/main/java/co/ada/rutapaedata/

```
DeliveryService.createDeliveries(deliveries: List<Delivery>) {
    // Repository.deliveries.create(delivery)
    // INSERT INTO Delivery(...)
}

MachineService.updateMachine(machine: Machine) {
    // Repository.machines.update(machine)
}
```

Servicios de datos

MachineData

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

```
object DeliveryService {
    fun getAll(): List<Delivery>
    fun create(delivery: Delivery): Delivery
    fun update(delivery: Delivery): Boolean
    fun delete(id: Long): Boolean
}

object BeneficiaryService {
    fun getAll(): List<Beneficiary>
    fun create(beneficiary: Beneficiary): Beneficiary
    fun findSimilar(embedding: FloatArray, similarity: Float = 0.7f):
List<Beneficiary>
    fun update(beneficiary: Beneficiary): Boolean
}

object MachineEnrollmentShiftService {
    fun getById(id: Long): MachineEnrollmentShift?
    fun create(shift: MachineEnrollmentShift): MachineEnrollmentShift
}
```

RutaPAEData

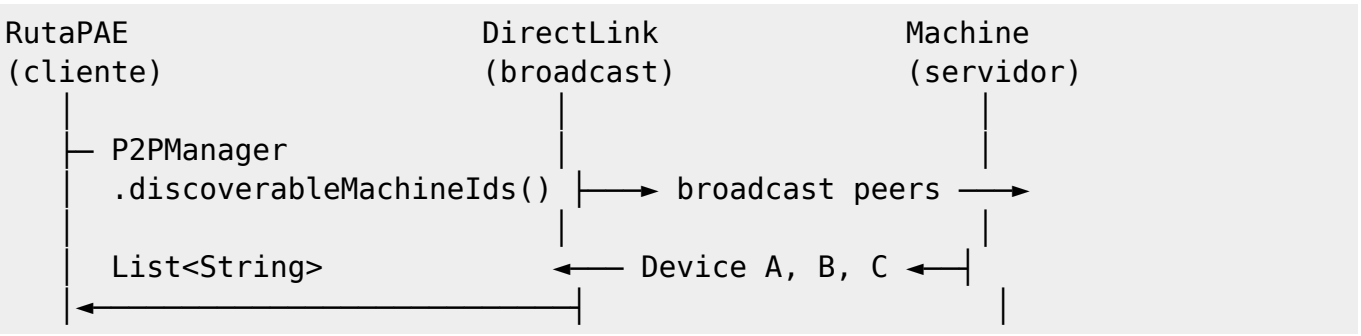
Ubicación: RutaPAEData/src/main/java/co/ada/rutapaedata/services/

```
object MachineService {
    fun getAll(): List<Machine>
    fun getById(id: Long): Machine?
    fun create(machine: Machine): Machine
    fun update(machine: Machine): Boolean
}

object DeliveryService {
    fun getAll(): List<Delivery>
    fun getByMachineId(machineId: Long): List<Delivery>
    fun create(delivery: Delivery): Delivery
    fun createMany(deliveries: List<Delivery>): List<Delivery>
}
```

Conexión P2P

Descubrimiento de máquinas



Conexión a máquina

```
// RutaPAE
val p2pGestor = p2pManager.connectMachine(machineId = 1L)
    ?: return // conexión fallida

// Ahora disponible
p2pGestor.connection.peer.get("/p2p/machine")
```

Endpoints disponibles en Machine P2P

```
GET /p2p/machine
    ├─ P2PMachineState (máquina actual)

GET /p2p/machine/deliveries/{campusId}/{shiftId}
    ├─ P2PDeliveriesPageResponse (entregas paginadas)
```

```
GET /p2p/machine/configuration
    ├─ P2PMachineConfiguration (config de máquina)

POST /p2p/machine/configuration
    └─ actualiza configuración

DELETE /p2p/machine/deliveries/{deliveryId}
    └─ marca entrega como confirmada en remoto
```

Modelos principales

Entidades de base de datos

Delivery

```kotlin
@DataTable("Delivery")
data class Delivery(
    @PrimaryKey
    val id: Long = 0,

    val weight: Float = 0f,
    val beneficiaryPhotoPath: String? = null,
    val alimentPhotoPath: String? = null,
    val similarity: Float = 0f,
    val processTimeMs: Long = 0L,

    val serverDeliveryId: String? = null,
    val synchronized: Boolean = false,
    val createdAt: Long = 0L
)
```

Beneficiary

```kotlin
@DataTable("Beneficiary")
data class Beneficiary(
    @PrimaryKey
    val id: Long = 0,

    @VectorColumn(dimensions = 512, distanceMetric = DistanceMetric.COSINE)
    val embedding: FloatArray,

    val name: String,
    val remoteBeneficiaryId: String? = null,
    val enrollmentId: String? = null,
    val lastRecognitionAtMs: Long = 0L
```

```
)
```

Machine (RutaPAE)

```
@DataTable("Machine")
data class Machine(
    @PrimaryKey
    val id: Long = 0,

    val machineId: String,
    val name: String,
    val machineStatus: P2PMachineStatus = P2PMachineStatus.Inactive,
    val unsyncedDeliveries: Long = 0,
    val localSyncedDeliveries: Long = 0
)
```

Modelos P2P (Contract)

```
data class P2PMachineState(
    val id: String = "",
    val name: String = "",
    val unsyncedDeliveries: Long = 0,
    val serverSyncedDeliveries: Long = 0,
    val status: String = "Inactive" // Active, Inactive, Error
)

data class P2PDeliveryData(
    val id: String,
    val weight: Float,
    val beneficiaryPhoto: String? = null, // URL
    val alimentPhoto: String? = null,     // URL
    val similarity: Float = 0f,
    val processTimeMs: Long = 0L
)
```

Configuración de proyecto

Settings

Ubicación: `settings.gradle.kts`

```
include(
    ":Machine",
```

```
    ":RutaPAE",
    ":MachineDomain",
    ":MachineData",
    ":RutaPAEDomain",
    ":RutaPAEData",
    ":Contract",
    ":DirectLink",
    ":VectorialDB",
    ":ComputerVision",
    ":Core"
)
```

Versiones

Ubicación: `gradle/libs.versions.toml`

```
[versions]
kotlin = "2.3.10"
compose = "1.5.4"
min-sdk = "27"
compile-sdk = "36"
jvm-target = "17"
```

Android (min/target)

- **minSdk**: 27 (Android 8.1)
- **targetSdk**: 36 (Android 15)
- **compileSdk**: 36.1

Testing

Estructura recomendada

```
Machine/src/
├── main/
│   └── java/co/ada/paemachine/
│       ├── screens/
│       ├── viewmodels/
│       └── di/
│
├── test/
│   └── java/co/ada/paemachine/
│       ├── viewmodels/
│       └── ...
│
```

```
└── androidTest/
    └── java/co/ada/paemachine/
        ├── screens/
        └── ...
```

Testing del StateManager

```
@Test
fun testStateTransition() {
    val manager = StateManager(
        repository = mockRepository,
        hardware = mockHardware
    )

    manager.init(mockContext)
    // assert state is WaitingForWeight
}
```

Testing de servicios

```
@Test
fun testBeneficiarySearch() {
    val service = BeneficiaryService
    val results = service.findSimilar(embedding, similarity = 0.8f)

    assertTrue(results.isNotEmpty())
}
```

Debugging

Logs

Core logging: `Core/src/main/java/co/ada/core/logging/`

```
Logger.d("StateManager", "Transitioning to ${state.name}")
Logger.e("P2PManager", "Connection failed: $error")
Logger.i("DeliveryService", "Delivery saved: $deliveryId")
```

Debug points

Machine: Observa emisores en Logcat

```
adb logcat | grep StateNameEmitter
adb logcat | grep DeliverySyncUiEmitter
```

RutaPAE: Verifica logs de Worker

```
adb logcat | grep DeliverySyncWorker
adb logcat | grep P2PManager
```

Common Tasks

Agregar nuevo estado a Machine

1. Crear: `MachineDomain/src/…/state/MyNewState.kt`
2. Implementar interfaz `State`
3. Agregar a lista en `StateManager.workflow`
4. Emitir cambios vía `StateNameEmitter`

Agregar nuevo servicio de datos

1. Crear: `MachineData/src/…/services/MyService.kt`
2. Implementar CRUD
3. Usar `Repository.myTable.create/update/delete()`
4. Mapear entidades si es necesario

Cambiar endpoints HTTP

1. Editar: `MachineDomain/src/…/endpoints/MachineEndpoints.kt`
2. Actualizar URL base si es necesario (Fuel)
3. Actualizar modelos de respuesta si cambia schema
4. Actualizar mapping en servicios

Agregar nueva ruta P2P

1. En Machine, agregar endpoint: `Machine/src/…/p2p/P2PMachine.kt`
2. En Contract, agregar modelo: `Contract/src/…/models/`
3. Usar en RutaPAE: `P2PManager.syncXxx()`

Referencias rápidas

Tarea	Archivo	Función
Ver estado actual	`MachineDomain/…/StateManager.kt`	`getCurrentState()`

Tarea	Archivo	Función
Emitir evento UI	`MachineDomain/…/emitters/StateNameEmitter.kt`	`emit(name)`
Buscar beneficiarios	`MachineData/…/BeneficiaryService.kt`	`findSimilar()`
Guardar entrega	`MachineData/…/DeliveryService.kt`	`create()`
Conectar P2P	`RutaPAEDomain/…/P2PManager.kt`	`connectMachine()`
Sincronizar entregas	`RutaPAEDomain/…/P2PManager.kt`	`syncDeliveriesFromMachine()`
Configurar Database	`gradle/libs.versions.toml`	versiones SDK
Ver modelos P2P	`Contract/src/…/models/`	estructuras P2P
Emitir progreso sync	`RutaPAE/…/emitters/DeliverySyncUiEmitter.kt`	`update()`
Enqueue sync worker	`RutaPAE/…/DeliverySyncScheduler.kt`	`enqueue()`