Referencia de APIs Principales

StateManager

Ubicación: `MachineDomain/src/…/state/StateManager.kt`

Métodos públicos

```
class StateManager(
    private val repository: Repository,
    private val hardware: Hardware
) {
    /**
     * Inicializa la máquina de estados.
     * @param context contexto Android
     * @return true si inicialización exitosa
     */
    fun init(context: Context): Boolean

    /**
     * Ejecuta un ciclo de la máquina de estados.
     * @return resultado del estado actual
     */
    suspend fun cycle(): StateResult

    /**
     * Obtiene nombre del estado actual.
     * @return nombre string del estado (ej: "CaptureFace")
     */
    fun getCurrentState(): String

    /**
     * Retrocede a estado anterior.
     * @param reason razón del retroceso (mostrado en logs)
     */
    fun goBack(reason: String): Boolean

    /**
     * Para la máquina de estados.
     */
    fun stop()

    // State data access
    /**
     * Acceso a datos del estado válidos solo durante ejecución.
     */
    val stateData: StateData
```

```
}

data class StateResult(
    val success: Boolean,
    val message: String,
    val nextState: String?
)

data class StateData(
    var weight: Float = 0f,
    var beneficiaryPhoto: Bitmap? = null,
    var faceEmbedding: FloatArray = FloatArray(512),
    var similarity: Float = 0f
)
```

Uso típico

```
// En Machine app
val stateManager = StateManager(repository, hardware)

// Iniciar flujo
viewModelScope.launch {
    if (stateManager.init(context)) {
        while (stateManager.getCurrentState() != "Done") {
            stateManager.cycle()
            StateNameEmitter.emit(stateManager.getCurrentState())
        }
    }
}
```

P2PManager

RutaPAE

Ubicación: RutaPAEDomain/src/…/P2PManager.kt

```
interface P2PManager {
    /**
     * Lista IDs de máquinas que se pueden descubrir.
     * @return Set de string IDs de máquinas
     */
    suspend fun discoverableMachineIds(): Set<String>

    /**
```

```
     * Obtiene candidatos de máquinas descubiertas.
     * @return Set de peers disponibles
     */
    suspend fun discoveredMachineCandidates(): Set<P2PPeer>

    /**
     * Conecta a máquina por ID.
     * @param machineId identificador de la máquina
     * @return P2PGestor si conexión exitosa, null otherwise
     */
    suspend fun connectMachine(machineId: String): P2PGestor?

    /**
     * Conecta a máquina vía hotspot.
     * @param ssid nombre de la red hotspot
     * @return P2PGestor si conexión exitosa, null otherwise
     */
    suspend fun connectMachineHotspot(ssid: String): P2PGestor?

    /**
     * Sincroniza entregas de máquina remota.
     * @param machineId ID de máquina
     * @param machineDatabaseId ID en BD local
     * @return resultado de sincronización
     */
    suspend fun syncDeliveriesFromMachine(
        machineId: String,
        machineDatabaseId: Long
    ): SyncDeliveriesResult
}

data class P2PPeer(
    val peerId: String,
    val name: String,
    val status: String  // "Connected", "Available", "Unavailable"
)

data class SyncDeliveriesResult(
    val success: Boolean,
    val message: String,
    val fetched: Int = 0,
    val failed: Int = 0
)
```

Uso típico

```
// En RutaPAE
val p2pManager = DomainManager.p2pManager ?: return

// Descubrir máquinas
```

```
val machines = p2pManager.discoverableMachineIds()

// Conectar
val gestor = p2pManager.connectMachine(machineId) ?: return

// Sincronizar
val result = p2pManager.syncDeliveriesFromMachine(machineId, dbId)
if (result.success) {
    DeliverySyncUiEmitter.update(result)
}
```

Repository

Interfaz común

```
interface Repository {
    /**
     * Acceso a tabla de entregas.
     */
    val deliveries: Table<Delivery>

    /**
     * Acceso a tabla de beneficiarios.
     */
    val beneficiaries: Table<Beneficiary>

    /**
     * Servicio CRUD para entregas.
     */
    val deliveryService: DeliveryService

    /**
     * Servicio CRUD para beneficiarios.
     */
    val beneficiaryService: BeneficiaryService
}

/**
 * Interfaz genérica para tablas.
 */
interface Table<T> {
    /**
     * Obtiene todos los registros.
     */
    suspend fun getAll(): List<T>
```

```
    /**
     * Obtiene por ID primario.
     */
    suspend fun getById(id: Long): T?

    /**
     * Crea nuevo registro.
     */
    suspend fun create(entity: T): T

    /**
     * Actualiza registro existente.
     */
    suspend fun update(entity: T): Boolean

    /**
     * Elimina registro.
     */
    suspend fun delete(id: Long): Boolean

    /**
     * Consulta personalizada.
     */
    suspend fun where(predicate: (T) -> Boolean): List<T>
}
```

Búsqueda vectorial (Beneficiarios)

```
/**
 * Interfaz de búsqueda vectorial en VectorDB.
 */
interface VectorSearchable<T> : Table<T> {
    /**
     * Búsqueda por similitud de vectores.
     * @param query vector de búsqueda (512 dims)
     * @param k número de resultados
     * @return lista ordenada por similitud descendente
     */
    suspend fun nearestNeighbors(
        query: FloatArray,
        k: Int = 5
    ): List<T>

    /**
     * Búsqueda por rango de distancia.
     * @param query vector de búsqueda
     * @param maxDistance distancia máxima permitida
     * @return resultados dentro del rango
     */
    suspend fun rangeSearch(
```

```
        query: FloatArray,
        maxDistance: Float
    ): List<T>
}
```

Services (Data Layer)

BeneficiaryService

```
object BeneficiaryService {
    /**
     * Busca beneficiarios similares al embedding.
     * @param embedding vector 512d
     * @param similarity threshold mínimo (0-1)
     * @param limit cantidad máxima de resultados
     * @return lista ordenada por similitud
     */
    suspend fun findSimilar(
        embedding: FloatArray,
        similarity: Float = 0.7f,
        limit: Int = 10
    ): List<Beneficiary>

    /**
     * Crea nuevo beneficiario.
     */
    suspend fun create(beneficiary: Beneficiary): Beneficiary

    /**
     * Actualiza beneficiario.
     */
    suspend fun update(beneficiary: Beneficiary): Boolean

    /**
     * Obtiene by ID remoto (para matching).
     */
    suspend fun getByRemoteId(remoteId: String): Beneficiary?

    /**
     * Obtiene todos no sincronizados.
     */
    suspend fun getLocalOnly(): List<Beneficiary>
}
```

DeliveryService

```
object DeliveryService {
    /**
     * Obtiene todas las entregas.
     */
    suspend fun getAll(): List<Delivery>

    /**
     * Obtiene entregas sin sincronizar.
     */
    suspend fun getUnsynced(): List<Delivery>

    /**
     * Crea nueva entrega.
     */
    suspend fun create(delivery: Delivery): Delivery

    /**
     * Actualiza entrega.
     */
    suspend fun update(delivery: Delivery): Boolean

    /**
     * Marca as synced en servidor.
     */
    suspend fun markSynced(deliveryId: Long, serverDeliveryId: String)

    /**
     * Obtiene por rango de fecha.
     */
    suspend fun getByDateRange(
        startMs: Long,
        endMs: Long
    ): List<Delivery>

    /**
     * Obtiene entregas por máquina.
     */
    suspend fun getByMachineId(machineId: Long): List<Delivery>
}
```

Emitters - Observable Streams

StateNameEmitter

```
/**
 * Emite nombre del estado actual.
 */
object StateNameEmitter {
    /**
     * Flow observable.
     */
    val flow: Flow<String>

    /**
     * Emitir cambio de estado.
     * @param stateName nombre del nuevo estado
     */
    suspend fun emit(stateName: String)

    /**
     * Obtener estado actual.
     */
    fun current(): String?

    /**
     * Escuchar cambios (helper).
     */
    suspend inline fun collect(action: (String) -> Unit) {
        flow.collect(action)
    }
}
```

DeliverySyncUiEmitter

```
object DeliverySyncUiEmitter {
    /**
     * Flow de resultados de sincronización.
     */
    val flow: Flow<SyncProgress>

    /**
     * Actualizar progreso de sincronización.
     */
    suspend fun update(progress: SyncProgress)
}

data class SyncProgress(
    val state: SyncState,      // Syncing, Success, Failed
    val current: Int,          // entregas procesadas
    val total: Int,            // entregas totales
    val message: String,
    val startTimeMs: Long,
```

```
    val estimatedRemaining: Long? = null
)

enum class SyncState { Syncing, Success, Failed, Cancelled }
```

Hardware Interfaces

Camera2Service

```
class Camera2Service(context: Context) {
    /**
     * Verifica si se requieren permisos.
     */
    fun requiresCameraPermission(): Boolean

    /**
     * Solicita permiso de cámara.
     */
    fun requestCameraPermission(): Boolean

    /**
     * Captura foto de rostro (selfie).
     * @return Bitmap de rostro capturado
     * @throws Exception si hay error de hardware
     */
    suspend fun takeSelfie(): Bitmap?

    /**
     * Captura foto de documento/producto.
     */
    suspend fun captureDocument(): Bitmap?

    /**
     * Cierra recurso de cámara.
     */
    fun close()
}
```

Scale (Balanza)

```
interface Scale {
    /**
     * Obtiene peso actual.
     * @return peso en kg, atau null si no disponible
     */
    suspend fun getWeight(): Float?
```

```
    /**
     * Vuelve escala a cero (tare).
     */
    suspend fun tare()

    /**
     * Agrega listener para cambios.
     */
    fun addEventListener(listener: ScaleEventListener)

    /**
     * Cierra conexión.
     */
    fun close()
}

interface ScaleEventListener {
    /**
     * Peso cambió.
     */
    fun onWeightChanged(weight: Float)

    /**
     * Estado de conexión cambió.
     */
    fun onConnectionStatusChanged(connected: Boolean)

    /**
     * Error en balanza.
     */
    fun onError(error: String)
}
```

Models (Data Classes)

Delivery

```
data class Delivery(
    @PrimaryKey
    val id: Long = 0,

    // Datos de captura
    val weight: Float = 0f,
    val beneficiaryPhotoPath: String? = null,  // ruta local
    val alimentPhotoPath: String? = null,      // ruta local
```

```
    val similarity: Float = 0f,                 // 0-1
    val processTimeMs: Long = 0L,

    // Sincronización
    val serverDeliveryId: String? = null,
    val synchronized: Boolean = false,
    val syncAttempts: Int = 0,
    val lastSyncAttemptMs: Long? = null,

    // Metadata
    val createdAt: Long = System.currentTimeMillis(),
    val beneficiaryId: Long = 0,
    val enrollmentShiftId: Long = 0
)
```

Beneficiary

```
data class Beneficiary(
    @PrimaryKey
    val id: Long = 0,

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

    val name: String,
    val remoteBeneficiaryId: String? = null,
    val enrollmentId: String? = null,

    val lastRecognitionAtMs: Long? = null,
    val lastRecognitionSimilarity: Float? = null,

    val createdAt: Long = System.currentTimeMillis()
)
```

P2PMachineState

```
data class P2PMachineState(
    val id: String = "",
    val name: String = "",
    val status: String = "Inactive",  // Active, Inactive, Error

    val unsyncedDeliveries: Long = 0,
    val serverSyncedDeliveries: Long = 0,

    val configuration: P2PMachineConfiguration? = null,
    val lastSync: Long? = null
)
```

```
data class P2PMachineConfiguration(
    val campusId: String? = null,
    val modalityId: String? = null,
    val shiftId: String? = null,
    val gradeId: String? = null,
    val operatorName: String? = null
)
```

Enums

```
enum class P2PMachineStatus {
    Active,    // funcionando
    Inactive,  // sin usar
    Error      // fallo
}

enum class DeliveryStatus {
    WaitingForWeight,
    CaptureImages,
    CaptureFace,
    ComparingWeights,
    GenerateEmbedding,
    VerifyInDatabase,
    SaveDelivery,
    WaitForWeightRemoved
}

enum class SyncState {
    Syncing,     // en progreso
    Success,     // completado exitosamente
    Failed,      // error
    Cancelled    // cancelado por usuario
}

enum class DistanceMetric {
    COSINE,      // similitud coseno (recomendado)
    EUCLIDEAN,   // distancia euclidiana
    MANHATTAN    // distancia manhattan
}
```

Excepciones Personalizadas

```
/**
```

```
 * Excepción de conectividad P2P.
 */
class DirectLinkException(message: String, cause: Throwable? = null)
    : Exception(message, cause)

/**
 * Excepción de operación de hardware.
 */
class HardwareException(message: String, cause: Throwable? = null)
    : Exception(message, cause)

/**
 * Excepción de validación de datos.
 */
class ValidationException(message: String, cause: Throwable? = null)
    : Exception(message, cause)

/**
 * Excepción de acceso a base de datos.
 */
class DatabaseException(message: String, cause: Throwable? = null)
    : Exception(message, cause)
```

Extension Functions

```
// En Core module

/**
 * Convertir Bitmap a ByteArray JPG.
 */
fun Bitmap.toJpgBytes(quality: Int = 90): ByteArray {
    val stream = ByteArrayOutputStream()
    compress(Bitmap.CompressFormat.JPEG, quality, stream)
    return stream.toByteArray()
}

/**
 * Calcular similitud coseno entre embeddings.
 */
fun FloatArray.cosineSimilarity(other: FloatArray): Float {
    require(size == 512 && other.size == 512)
    // implementación
}

/**
 * Validar peso válido.
 */
fun Float.isValidWeight(): Boolean = this > 0.5f && this < 100f
```

```
/**
 * Formatear timestamp a string legible.
 */
fun Long.toReadableDate(): String {
    val sdf = SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.default)
    return sdf.format(Date(this))
}
```

Constants

```
// MachineDomain
const val MIN_VALID_WEIGHT = 0.5f
const val MAX_VALID_WEIGHT = 100f
const val SIMILARITY_THRESHOLD = 0.7f

// Timeouts
const val API_TIMEOUT_MS = 30000L
const val P2P_TIMEOUT_MS = 20000L
const val CAMERA_TIMEOUT_MS = 10000L

// Configuración
const val EMBEDDING_DIMENSIONS = 512
const val MAX_PHOTO_QUALITY = 95
const val PHOTO_COMPRESSION_QUALITY = 80

// Paths
const val PHOTOS_DIR = "photos"
const val DB_NAME = "pae_database.db"
const val VECTOR_INDEX_NAME = "beneficiary_embedding_index"
```

Kotlin DSL Help

Configurar módulo en settings.gradle.kts

```
include(":NombreModulo")

project(":NombreModulo").projectDir = file("NombreModulo")
```

Dependencias comunes en build.gradle.kts

```
dependencies {
    // Android
    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.activity.compose)

    // Compose
    implementation(platform(libs.androidx.compose.bom))
    implementation(libs.androidx.compose.ui)
    implementation(libs.androidx.compose.material3)

    // Coroutines
    implementation(libs.kotlinx.coroutines.core)
    implementation(libs.kotlinx.coroutines.android)

    // Inyección
    implementation(libs.hilt.android)
    kapt(libs.hilt.compiler)

    // Testing
    testImplementation(libs.junit)
    testImplementation(libs.mockito.kotlin)
}
```

From:
http://wiki.adacsc.co/ - **Wiki**

Permanent link:
http://wiki.adacsc.co/doku.php?id=ada:howto:sicoferp:factory:new-migracion-sicoferp:api-reference

Last update: **2026/04/07 19:58**