PULSE PROTOCOL - Propuesta Técnica Completa

Documento de Arquitectura, Diseño y Especificación Técnica

Versión: 1.0 **Fecha:** Febrero 2026 **Estado:** Aprobado para desarrollo **Confidencial:** Borrador interno para equipo de desarrollo

---

Tabla de Contenidos

---

1. Resumen Ejecutivo

Pulse Protocol es un sistema de herencia criptográfica descentralizada construido sobre **Stellar/Soroban** que utiliza inteligencia artificial para verificar continuamente la "prueba de vida" del usuario. El sistema hereda no solo activos digitales (tokens), sino también **documentos públicos importantes, manifiestos, herencias y escrituras**, todo registrado y ejecutado en la red de Stellar.

Visión

> *Un testamento digital universal, descentralizado e inteligente que protege el legado completo de una persona — activos financieros y documentos — mediante verificación biométrica pasiva con IA, eliminando intermediarios legales y ejecutando la herencia de forma automática, segura y privada.*

Características Diferenciadoras

- **No intrusivo:** El usuario no realiza acciones especiales; el sistema observa patrones normales de vida.
- **Personalizado:** Cada usuario tiene un perceptrón de ML único, entrenado con sus propios patrones.
- **Multi-activo:** Hereda tokens, documentos, manifiestos, escrituras y cualquier registro digital.
- **Gradual:** Detecta degradación (enfermedad prolongada) antes de eventos súbitos.
- **Resistente a fraude:** Combina 3 pilares biométricos y comportamentales.
- **Descentralizado:** Lógica de herencia en smart contracts Soroban, documentos en IPFS.
- **Privado:** Privacy by design. Datos biométricos raw nunca salen del dispositivo.

---

2. Problema y Solución

2.1 El Problema

La adopción masiva de criptomonedas ha creado un problema sin precedentes: la **pérdida irrecuperable de activos por fallecimiento del titular**. Se estima que entre 3 y 4 millones de Bitcoin (~20% del suministro total) están permanentemente perdidos, una proporción significativa por muerte de propietarios sin transferir claves privadas.

Los mecanismos tradicionales de herencia presentan limitaciones fundamentales:

1. **Dependencia de intermediarios:** Abogados, notarios y sistemas judiciales que no comprenden ni pueden acceder a activos criptográficos.
2. **Jurisdicción limitada:** Los testamentos están sujetos a leyes locales; las criptomonedas son globales.
3. **Rigidez:** Los testamentos son documentos estáticos que no se adaptan a condiciones cambiantes.
4. **Falta de privacidad:** En muchas jurisdicciones, los testamentos son documentos públicos.
5. **Sin cobertura documental digital:** No existe un mecanismo descentralizado para heredar documentos digitales importantes (escrituras, manifiestos, registros).

2.2 La Solución: Pulse Protocol

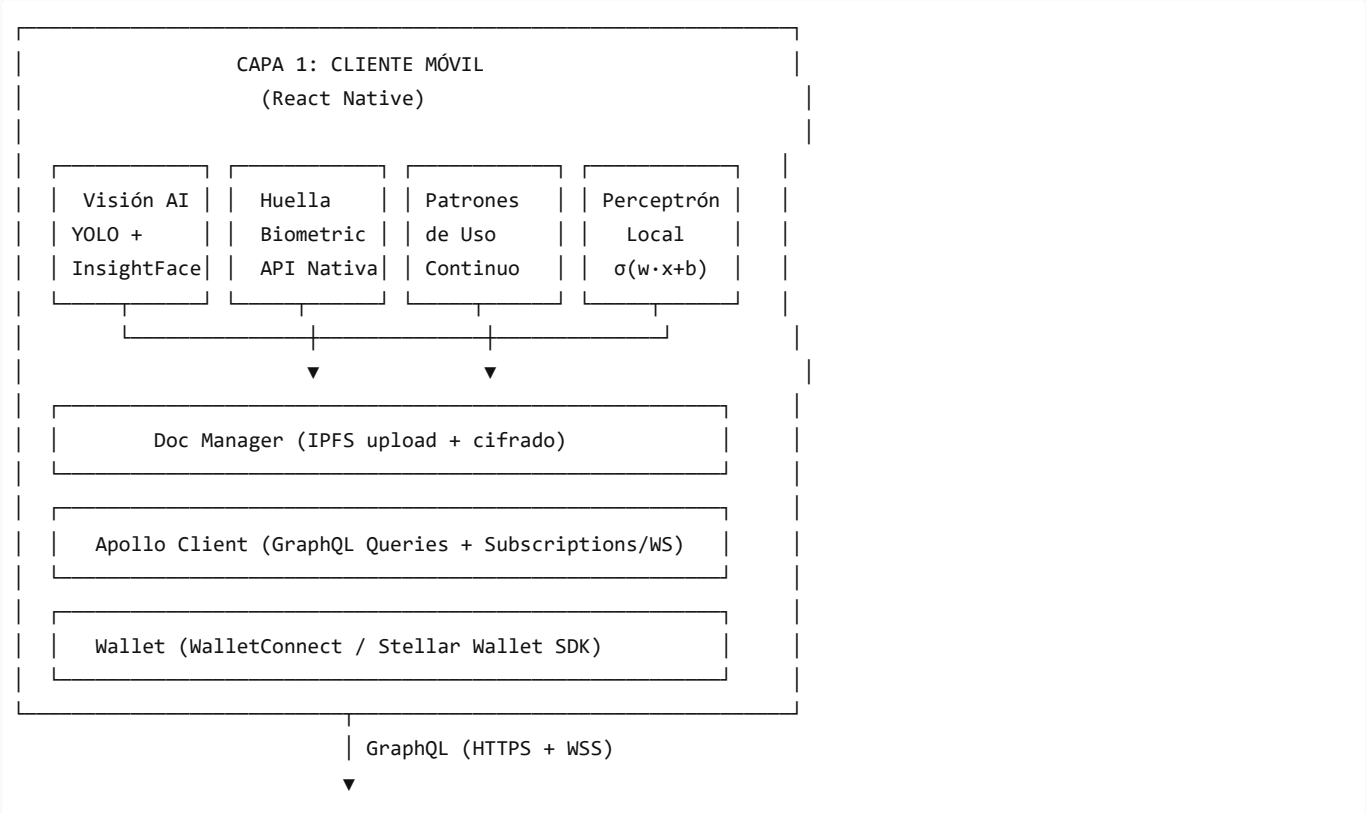
Pulse Protocol introduce la **herencia programable con verificación biométrica pasiva**:

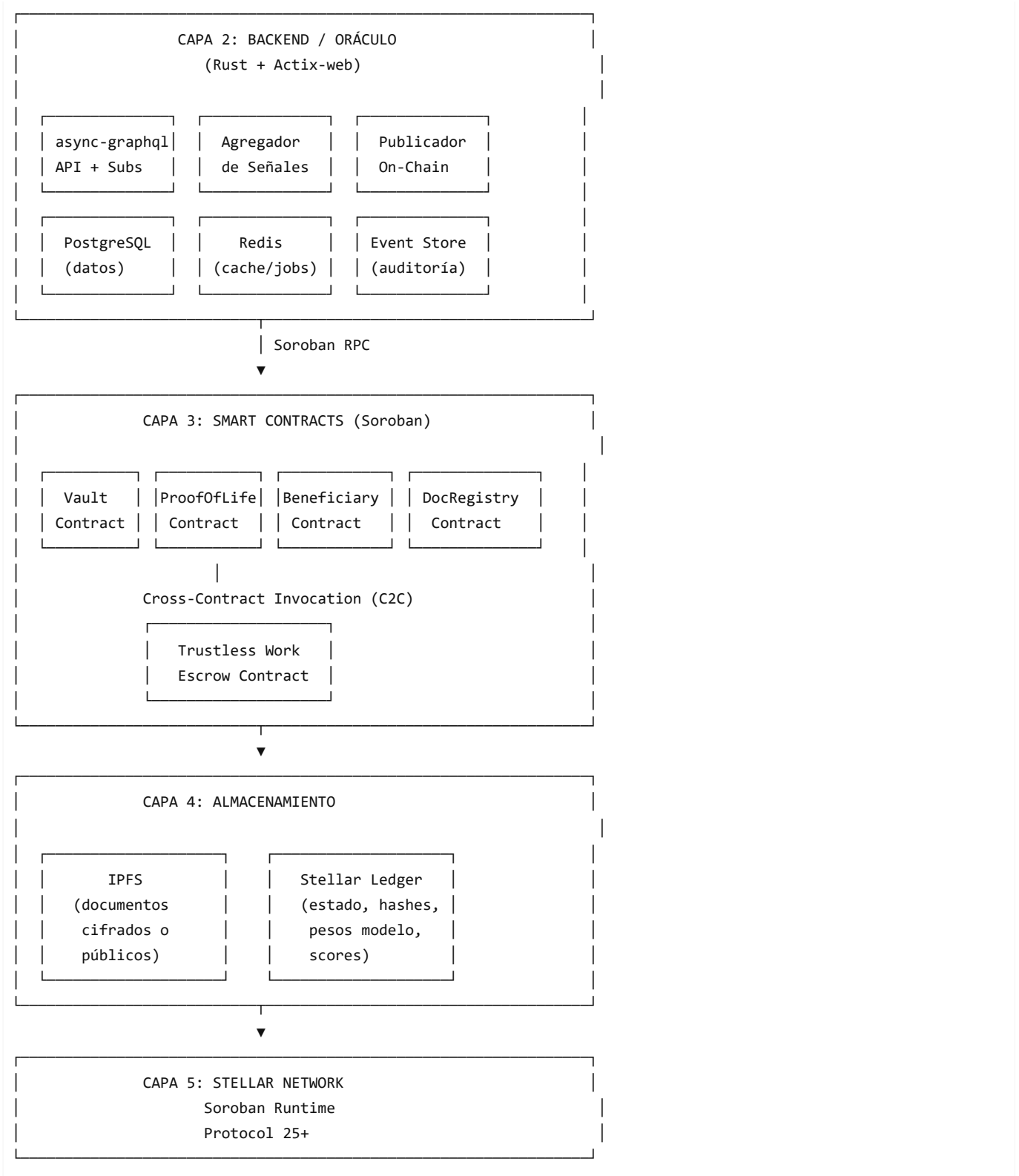
- Monitorea pasivamente al usuario mediante su teléfono móvil.
- Usa **3 pilares de verificación**: huella dactilar (pasiva), reconocimiento facial (automático adaptativo), y análisis de patrones de comportamiento.
- Entrena un **perceptrón personalizado** por usuario que genera un "liveness score" (0-1).
- Almacena tokens en **vaults con escrow** (Trustless Work) y documentos en **IPFS** con hash on-chain.
- Ejecuta la herencia automáticamente cuando se confirma inactividad prolongada.
- Los beneficiarios reciben tokens + acceso a documentos según reglas predefinidas.

---

3. Arquitectura General del Sistema

Pulse Protocol se estructura en **5 capas** que interactúan para proporcionar un sistema robusto de herencia criptográfica y documental:





---

4. Capa 1: Cliente Móvil (React Native)

4.1 Descripción General

El cliente móvil es la interfaz principal y única del usuario con Pulse Protocol. Es una aplicación React Native que concentra:

- Recolección de datos biométricos y comportamentales (procesados localmente).
- Ejecución del perceptrón de prueba de vida.
- Gestión de vaults, beneficiarios y documentos.

- Conexión con wallet Stellar.
- Comunicación con el backend vía GraphQL.

No hay dashboard web. Todo se gestiona desde la app móvil.

4.2 Módulos del Cliente

4.2.1 Módulo de Visión Artificial (Face Recognition)

Tecnología: YOLO (detección) + InsightFace/MobileFaceNet (verificación) + Liveness detection

Flujo:

```
Frame de cámara frontal
    │
    ▼
YOLO v8-face (¿hay un rostro? ¿dónde está?)
    │
    ▼
Crop del rostro detectado
    │
    ▼
InsightFace / MobileFaceNet (~1MB, 512-d embedding)
    │
    ├── Genera embedding del rostro actual
    ├── Compara con embedding de referencia (registrado en onboarding)
    └── Calcula cosine similarity → face_match_score [0, 1]
    │
    ▼
Liveness Detection (anti-spoofing)
    │
    ├── Detección de parpadeo natural
    ├── Micro-movimientos involuntarios del rostro
    ├── Análisis de textura (pantalla vs piel real)
    ├── Análisis temporal (secuencia de frames, no foto estática)
    └── Genera face_liveness_score [0, 1]
    │
    ▼
Scores → Perceptrón Local
```

Franjas horarias de verificación facial:

Las franjas son **adaptativas por comportamiento**:

1. **Calibración (semanas 1-4):** El sistema registra cuándo el usuario usa más activamente el teléfono, identificando ventanas de alta actividad (ej: 7:30am, 1:00pm, 8:30pm).
2. **Post-calibración:** Programa verificación facial automática en esas ventanas aprendidas.
3. **Captura silenciosa:** Cuando el usuario está mirando la pantalla activamente durante una ventana, la cámara frontal captura y verifica. Si no hay actividad, espera al siguiente momento natural.
4. **Re-adaptación continua:** Las ventanas se ajustan gradualmente mediante online learning para seguir cambios legítimos de rutina.

Registro (onboarding):

- El usuario captura N fotos desde diferentes ángulos.
- Se genera el **face embedding de referencia** usando InsightFace.
- Solo se almacena el embedding (vector 512-d), **nunca la imagen**.
- El embedding se guarda en el Secure Enclave del dispositivo.

Modelos pre-entrenados utilizados:

Modelo	Propósito	Tamaño	Precisión
YOLO v8-face / v9	Detección de rostros en frame	~6MB	Alta velocidad, tiempo real
MobileFaceNet (InsightFace)	Embedding facial para verificación	~1MB	99.5% en LFW
Liveness model (custom/pretrained)	Anti-spoofing	~2MB	Multi-señal

Todos exportados a **TFLite** u **ONNX** para ejecución en React Native.

4.2.2 Módulo de Huella Dactilar

Tecnología: API nativa del sistema operativo

- **Android:** `BiometricPrompt` API
- **iOS:** `LocalAuthentication` framework (`LAContext`)

Principio: Captura **pasiva**. Cada vez que el usuario toca el sensor de huella del teléfono (para desbloquear, autenticar en apps, etc.), el sistema registra:

```
Evento de huella:
├── timestamp: cuándo ocurrió
├── resultado: match / no_match / sensor_no_disponible
├── NO se capturan datos biométricos raw (el OS valida internamente)
└── Solo se registra el evento y metadatos temporales
```

Señales derivadas para el perceptrón:

Señal	Descripción	Cálculo
`fingerprint_frequency`	¿Cuántas veces tocó el sensor hoy vs promedio diario?	`matches_hoy / promedio_historico` normalizado [0,1]
`fingerprint_consistency`	¿El patrón horario de uso del sensor es normal?	Similitud del histograma horario vs baseline [0,1]

Registro (onboarding):

- El usuario registra su huella a través de la API biométrica del OS.
- La huella queda almacenada en el **Secure Enclave/TEE** del dispositivo (gestionado por el OS).
- Pulse Protocol nunca accede a datos raw de la huella.

4.2.3 Módulo de Análisis de Patrones de Comportamiento

Este módulo captura el **fingerprint comportamental** único del usuario sin acceder a contenido personal.

Señales capturadas:

Categoría	Señal	Descripción	Método de captura
Temporal	`wake_time`	Hora típica del primer uso diario	Timestamp primer evento
	`sleep_time`	Hora típica del último uso diario	Timestamp último evento
	`active_hours`	Distribución de uso durante el día	Histograma por hora
Interacción	`typing_speed`	Velocidad promedio de escritura	Eventos de teclado (sin contenido)
	`scroll_pattern`	Patrón y velocidad de scroll	Eventos de scroll
	`touch_pressure`	Presión típica en pantalla	API de presión táctil
	`gesture_style`	Estilo de gestos (swipe, tap, etc.)	Eventos táctiles

Uso	`app_frequency`	Frecuencia de uso por aplicación	Usage Stats API
	`session_duration`	Duración típica de sesiones	Tiempo entre foreground/background
	`notification_response`	Tiempo de respuesta a notificaciones	Timestamp notificación vs apertura
Movimiento	`device_orientation`	Cómo sostiene el dispositivo	Giroscopio
	`movement_signature`	Patrón de acelerómetro al caminar	Acelerómetro

Señales agregadas para el perceptrón:

Feature	Derivada de	Cálculo
`time_of_day_normality`	Temporal	Similitud distribución horaria actual vs baseline
`typing_pattern_match`	Interacción	Similitud velocidad/ritmo actual vs baseline
`app_usage_match`	Uso	Similitud distribución de apps actual vs baseline
`movement_pattern_match`	Movimiento	Similitud patrón acelerómetro/giroscopio vs baseline
`session_behavior`	Uso	Similitud duración/frecuencia sesiones vs baseline

4.2.4 Módulo de Gestión de Documentos

Permite al usuario subir, cifrar y registrar documentos para herencia.

Flujo de registro de documento:

```
1. Usuario selecciona documento desde el dispositivo
2. Elige visibilidad: PÚBLICO o PRIVADO
3. Si PRIVADO:
   ├── Genera clave AES-256 aleatoria
   ├── Cifra el documento con AES-256-GCM
   ├── Cifra la clave AES con la public key de cada beneficiario asignado
   └── Almacena claves cifradas en el contrato
4. Sube documento (cifrado o no) a IPFS → obtiene CID
5. Registra en DocumentRegistry Contract:
   ├── IPFS CID
   ├── SHA-256 hash del documento original
   ├── Tipo de documento
   ├── Flag de cifrado
   └── Metadata
6. Vincula documento al vault correspondiente
```

4.2.5 Wallet Integration

No se usa Freighter SDK (no compatible con React Native).

Opciones de conexión:

- **WalletConnect:** Protocolo estándar para conectar wallets móviles
- **`@stellar/typescript-wallet-sdk`:** SDK oficial de Stellar para wallets
  - Nota: usar `createKeypairFromRandom` en lugar de `createKeypair` en React Native
- **Deep linking:** Para interacción con Freighter Mobile u otras wallets nativas

Funcionalidad wallet en la app:

- Crear cuenta Stellar / importar existente

- Firmar transacciones (depósitos, claims, registros)
- Ver balances de vault
- Gestionar trustlines para tokens

4.2.6 Apollo Client (GraphQL)

Comunicación con el backend mediante **Apollo Client for React Native**:

- **Queries:** Lectura de datos (vaults, documentos, scores, beneficiarios)
- **Mutations:** Escritura de datos (crear vault, registrar documento, emergency check-in)
- **Subscriptions (WebSocket):** Eventos en tiempo real (cambios de estado, alertas de liveness, claims iniciados)

---

5. Capa 2: Backend / Oráculo (Rust)

5.1 Descripción General

El backend es un servicio Rust que actúa como:

1. **API GraphQL:** Interfaz entre el cliente móvil y el sistema.
2. **Oráculo:** Agrega señales de prueba de vida y publica scores on-chain.
3. **Event processor:** Gestiona eventos del sistema y notificaciones.
4. **Job scheduler:** Ejecuta tareas periódicas (verificación de timeouts, transiciones de estado).

5.2 Stack del Backend

Componente	Tecnología	Versión/Crate
HTTP Server	Actix-web	latest
GraphQL	async-graphql	latest
GraphQL Subscriptions	async-graphql + actix-web-actors (WebSocket)	latest
Base de datos	PostgreSQL via sqlx o diesel	latest
Cache / Jobs	Redis via redis-rs + background workers	latest
Event Store	PostgreSQL (tabla de eventos append-only)	custom
Stellar SDK	stellar-sdk (Rust)	latest
IPFS Client	ipfs-api o HTTP client a IPFS gateway	latest

5.3 Módulos del Backend

5.3.1 API GraphQL (async-graphql)

Expone toda la funcionalidad del sistema mediante GraphQL:

Queries principales:

```
type Query {
  # Vaults
  vault(id: ID!): Vault
  myVaults: [Vault!]!

  # Prueba de vida
  livenessScore(userId: ID!): LivenessData
  verificationHistory(userId: ID!, limit: Int): [VerificationRecord!]!

  # Beneficiarios
```

```
  beneficiaries(vaultId: ID!): [Beneficiary!]!

  # Documentos
  documents(vaultId: ID!): [Document!]!
  document(docId: ID!): Document

  # Estado del sistema
  systemStatus: SystemStatus
}
```

Mutations principales:

```
type Mutation {
  # Vaults
  createVault(input: CreateVaultInput!): Vault!
  deposit(vaultId: ID!, amount: String!, token: String!): Transaction!
  withdraw(vaultId: ID!, amount: String!, token: String!): Transaction!

  # Beneficiarios
  setBeneficiaries(vaultId: ID!, beneficiaries: [BeneficiaryInput!]!): [Beneficiary!]!

  # Documentos
  registerDocument(input: RegisterDocumentInput!): Document!
  linkDocumentToVault(docId: ID!, vaultId: ID!): Boolean!

  # Prueba de vida
  submitVerification(input: VerificationInput!): VerificationResult!
  emergencyCheckin: CheckinResult!

  # Modelo
  updateModelWeights(weights: [String!]!, bias: String!): ModelUpdateResult!

  # Claims
  claimInheritance(vaultId: ID!): ClaimResult!
}
```

Subscriptions (tiempo real vía WebSocket):

```
type Subscription {
  # Cambios de estado del vault
  vaultStatusChanged(vaultId: ID!): VaultStatusEvent!

  # Alertas de prueba de vida
  livenessAlert(userId: ID!): LivenessAlertEvent!

  # Notificación de claim iniciado
  claimStarted(vaultId: ID!): ClaimEvent!

  # Actualizaciones de verificación
  verificationCompleted(userId: ID!): VerificationEvent!

  # Eventos de documentos
  documentAccessGranted(beneficiaryId: ID!): DocumentAccessEvent!
}
```

5.3.2 Agregador de Señales

Recibe los scores individuales del cliente móvil y los procesa:

```
Señales del móvil (por verificación):
├── face_match_score: f64
├── face_liveness_score: f64
├── fingerprint_frequency: f64
├── fingerprint_consistency: f64
├── time_of_day_normality: f64
├── typing_pattern_match: f64
├── app_usage_match: f64
├── movement_pattern_match: f64
├── days_since_last_verify: f64
├── session_behavior: f64
└── perceptron_output: f64 (calculado localmente)
    │
    ▼
Agregador:
├── Valida rangos (todos en [0,1])
├── Verifica consistencia temporal (no saltos imposibles)
├── Compara con historial del usuario (detección de anomalías)
├── Aplica rate limiting (máximo N verificaciones por hora)
└── Genera score final agregado
    │
    ▼
Publica score on-chain via Publicador
```

5.3.3 Publicador On-Chain

Toma scores validados y los publica en el smart contract ProofOfLife:

```
Score validado + firma del oráculo
    │
    ▼
Construye transacción Soroban:
├── Llama submit_verification() en ProofOfLife Contract
├── Incluye: user, score (u32 0-10000), source, oracle_signature
└── Firma con clave del oráculo
    │
    ▼
Envía a Stellar Network via Soroban RPC
    │
    ▼
Verifica resultado de la transacción
    │
    ▼
Registra en Event Store
```

5.3.4 Job Scheduler (Redis-backed)

Tareas periódicas ejecutadas por workers en background:

Job	Frecuencia	Descripción
`check_alert_timeouts`	Cada 1 hora	Detecta usuarios en ALERT que deben pasar a GRACE
`check_grace_timeouts`	Cada 30 min	Detecta usuarios en GRACE cuyo período expiró → TRIGGERED

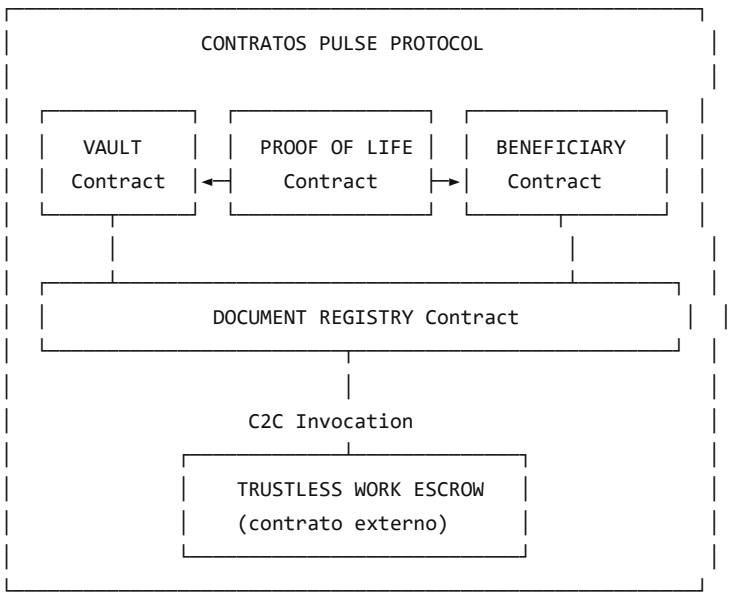
`send_notifications`	Event-driven	Envía push notifications según estado
`sync_chain_state`	Cada 5 min	Sincroniza estado on-chain con base de datos local
`cleanup_expired_sessions`	Cada 6 horas	Limpia datos de sesión expirados
`model_weight_sync`	Event-driven	Sincroniza pesos del perceptrón con la blockchain

---

6. Capa 3: Smart Contracts (Soroban)

6.1 Contratos del Sistema

Pulse Protocol implementa **4 contratos propios** + integración **Contract-to-Contract (C2C)** con Trustless Work:



6.2 Vault Contract

Custodia los activos del usuario y coordina el flujo de herencia.

Interfaz:

```
pub trait VaultTrait {
    /// Crear un nuevo vault para el usuario
    fn create_vault(
        env: Env,
        owner: Address,
        token: Address,
    ) -> VaultId;

    /// Depositar activos en el vault
    fn deposit(
        env: Env,
        vault_id: VaultId,
        amount: i128,
        token: Address,
    );

    /// Retirar activos (solo si estado ACTIVE)
```

```
    fn withdraw(
        env: Env,
        vault_id: VaultId,
        amount: i128,
        token: Address,
    );

    /// Configurar lista de beneficiarios con porcentajes
    fn set_beneficiaries(
        env: Env,
        vault_id: VaultId,
        beneficiaries: Vec<Beneficiary>,
    );

    /// Obtener estado actual del vault
    fn get_status(env: Env, vault_id: VaultId) -> VaultStatus;

    /// Obtener balance del vault por token
    fn get_balance(env: Env, vault_id: VaultId, token: Address) -> i128;

    /// Vincular con contrato ProofOfLife
    fn link_proof_of_life(env: Env, vault_id: VaultId, pol_contract: Address);

    /// Transicionar estado (llamado por ProofOfLife o internamente)
    fn transition_status(env: Env, vault_id: VaultId, new_status: VaultStatus);
}
```

Estructuras de datos:

```
#[contracttype]
#[derive(Clone)]
pub struct Vault {
    pub id: VaultId,
    pub owner: Address,
    pub status: VaultStatus,
    pub beneficiaries: Vec<Beneficiary>,
    pub proof_of_life_contract: Address,
    pub escrow_contract: Option<Address>,     // Trustless Work escrow
    pub document_ids: Vec<DocId>,
    pub created_at: u64,
    pub last_updated: u64,
}

#[contracttype]
#[derive(Clone, PartialEq)]
pub enum VaultStatus {
    Active,       // Score > 0.70 — operación normal
    Alert,        // 0.30 < Score < 0.70 — baja actividad
    GracePeriod,  // Score < 0.30 — período de gracia activo
    Triggered,    // Timeout en GracePeriod — herencia activada
    Distributed,  // Todos los assets reclamados — terminal
}
```

6.3 Proof of Life Contract

Gestiona la verificación de vida y el modelo de IA del usuario.

Interfaz:

```
pub trait ProofOfLifeTrait {
    /// Registrar modelo de perceptrón del usuario
    fn register_model(
        env: Env,
        user: Address,
        initial_weights: Vec<i128>,  // Fixed-point, 6 decimales
        bias: i128,
    );

    /// Enviar resultado de verificación (llamado por el oráculo)
    fn submit_verification(
        env: Env,
        user: Address,
        score: u32,                      // 0-10000
        source: VerificationSource,
        oracle_sig: BytesN<64>,
    );

    /// Actualizar pesos del modelo (tras re-entrenamiento)
    fn update_model(
        env: Env,
        user: Address,
        new_weights: Vec<i128>,
        new_bias: i128,
    );

    /// Obtener score de vida actual
    fn get_liveness_score(env: Env, user: Address) -> u32;

    /// Check-in manual de emergencia
    fn emergency_checkin(env: Env, user: Address);

    /// Obtener datos del modelo
    fn get_model(env: Env, user: Address) -> LifeModel;

    /// Obtener historial de verificaciones
    fn get_verification_history(env: Env, user: Address, limit: u32) -> Vec<VerificationRecord>;
}
```

Estructuras de datos:

```
#[contracttype]
#[derive(Clone)]
pub struct LifeModel {
    pub weights: Vec<i128>,          // Pesos del perceptrón (fixed-point, 6 decimales)
    pub bias: i128,                  // Bias del perceptrón
    pub version: u32,                // Versión del modelo
    pub last_updated: u64,           // Timestamp última actualización
    pub calibration_complete: bool,  // ¿Calibración finalizada?
    pub total_verifications: u64,    // Total de verificaciones realizadas
    pub avg_confidence: u32,         // Confianza promedio 0-10000
    pub alert_threshold: u32,        // Default: 7000 (0.70)
    pub critical_threshold: u32,     // Default: 3000 (0.30)
```

```
    pub grace_period_days: u32,       // Default: 30
}

#[contracttype]
#[derive(Clone)]
pub struct VerificationRecord {
    pub timestamp: u64,
    pub liveness_score: u32,          // 0-10000
    pub source: VerificationSource,
    pub oracle_signature: BytesN<64>,
}

#[contracttype]
#[derive(Clone)]
pub enum VerificationSource {
    FacialRecognition,    // Verificación facial automática
    Fingerprint,          // Evento de huella dactilar
    BehaviorPattern,      // Patrón de comportamiento
    PerceptronAggregate,  // Score agregado del perceptrón
    ManualCheckin,        // Check-in manual de emergencia
    WitnessAttestation,   // Atestación de testigo (futuro)
}
```

6.4 Beneficiary Contract

Gestiona beneficiarios y la distribución de herencia.

Interfaz:

```
pub trait BeneficiaryTrait {
    /// Agregar beneficiario a un vault
    fn add_beneficiary(
        env: Env,
        vault_id: VaultId,
        beneficiary: Beneficiary,
    );

    /// Remover beneficiario
    fn remove_beneficiary(
        env: Env,
        vault_id: VaultId,
        beneficiary_address: Address,
    );

    /// Ejecutar claim de herencia (solo si vault está TRIGGERED)
    fn claim(
        env: Env,
        vault_id: VaultId,
        claimer: Address,
    );

    /// Obtener lista de beneficiarios
    fn get_beneficiaries(env: Env, vault_id: VaultId) -> Vec<Beneficiary>;

    /// Verificar si un claim es válido
```

```
    fn can_claim(env: Env, vault_id: VaultId, claimer: Address) -> bool;
}
```

Estructuras de datos:

```
#[contracttype]
#[derive(Clone)]
pub struct Beneficiary {
    pub address: Address,        // Wallet Stellar del beneficiario
    pub percentage: u32,         // 0-10000 (100.00%)
    pub claimed: bool,           // ¿Ya reclamó?
    pub claimed_at: Option<u64>, // Timestamp del claim
}

// REGLA: La suma de todos los percentages DEBE ser exactamente 10000
// REGLA: Mínimo 1 beneficiario para activar vault
// REGLA: Modificación solo en estado ACTIVE
```

6.5 Document Registry Contract

Registra documentos en la blockchain con referencia a IPFS.

Interfaz:

```
pub trait DocumentRegistryTrait {
    /// Registrar un nuevo documento
    fn register_document(
        env: Env,
        owner: Address,
        ipfs_cid: String,           // CID de IPFS
        doc_hash: BytesN<32>,        // SHA-256 del documento original
        doc_type: DocumentType,
        is_encrypted: bool,
        metadata: Map<String, String>,
    ) -> DocId;

    /// Vincular documento a un vault
    fn link_to_vault(
        env: Env,
        doc_id: DocId,
        vault_id: VaultId,
    );

    /// Almacenar clave de descifrado cifrada por beneficiario
    fn store_encrypted_key(
        env: Env,
        doc_id: DocId,
        beneficiary: Address,
        encrypted_key: Bytes,        // Clave AES cifrada con pubkey del beneficiario
    );

    /// Revelar acceso post-trigger (llamado internamente al activar herencia)
    fn grant_access(
        env: Env,
        doc_id: DocId,
        beneficiary: Address,
```

```
    ) -> DocumentAccess;

    /// Verificar existencia y timestamp de un documento
    fn verify_document(env: Env, doc_id: DocId) -> DocumentProof;

    /// Obtener documentos de un vault
    fn get_vault_documents(env: Env, vault_id: VaultId) -> Vec<DocumentInfo>;
}
```

Estructuras de datos:

```
#[contracttype]
#[derive(Clone)]
pub enum DocumentType {
    PublicManifest,     // Manifiesto público
    Deed,               // Escritura
    Will,               // Testamento digital
    Certificate,        // Certificado
    LegalDocument,      // Documento legal genérico
    PersonalLetter,     // Carta personal
    Other,              // Otro tipo
}

#[contracttype]
#[derive(Clone)]
pub struct DocumentInfo {
    pub id: DocId,
    pub owner: Address,
    pub ipfs_cid: String,
    pub doc_hash: BytesN<32>,
    pub doc_type: DocumentType,
    pub is_encrypted: bool,
    pub registered_at: u64,
    pub vault_id: Option<VaultId>,
}

#[contracttype]
#[derive(Clone)]
pub struct DocumentProof {
    pub exists: bool,
    pub doc_hash: BytesN<32>,
    pub registered_at: u64,
    pub ipfs_cid: String,
}

#[contracttype]
#[derive(Clone)]
pub struct DocumentAccess {
    pub ipfs_cid: String,
    pub encrypted_key: Option<Bytes>,  // None si el documento es público
    pub doc_type: DocumentType,
}
```

6.6 Representación de Punto Fijo en Soroban

Soroban **no soporta floats nativos**. Todos los valores decimales se representan con fixed-point de 6 decimales:

```
// Conversión: valor real → almacenamiento on-chain
// 0.847523 se almacena como 847523i128
let weight_stored: i128 = (weight_real * 1_000_000.0) as i128;

// Conversión inversa: almacenamiento → valor real
let weight_real: f64 = weight_stored as f64 / 1_000_000.0;

// Constante de escala
const FIXED_POINT_SCALE: i128 = 1_000_000; // 10^6
```

Representación de scores:

- Scores se almacenan como `u32` en rango **0-10000**.
- Representa 0.00% a 100.00% (2 decimales de precisión).
- Ejemplo: `7523` = 75.23% = 0.7523

---

7. Capa 4: Almacenamiento (IPFS + Stellar)

7.1 IPFS para Documentos

Los documentos se almacenan en IPFS (InterPlanetary File System):

Aspecto	Decisión
Red	IPFS pública
Pinning	Servicio de pinning (Pinata, Infura, o nodo propio)
Direccionamiento	CID (Content Identifier)
Cifrado	AES-256-GCM (para documentos privados, antes de subir)

Flujo de almacenamiento:

```
Documento público:
  Documento original → IPFS upload → CID → registrado on-chain

Documento privado:
  Documento original → AES-256-GCM encrypt → IPFS upload → CID → registrado on-chain
  Clave AES → cifrada con pubkey de cada beneficiario → almacenada on-chain
```

7.2 Stellar Ledger para Estado

Todo el estado del sistema vive on-chain en Soroban:

Dato	Almacenamiento
Vaults y sus estados	Vault Contract storage
Scores de prueba de vida	ProofOfLife Contract storage
Pesos del perceptrón	ProofOfLife Contract storage
Historial de verificaciones	ProofOfLife Contract storage
Beneficiarios y porcentajes	Beneficiary Contract storage
Registros de documentos (hash + CID)	DocumentRegistry Contract storage

Escrows de herencia	Trustless Work Contract storage

---

8. Capa 5: Stellar Network

8.1 Red y Protocolo

Aspecto	Detalle
Red	Stellar Mainnet (producción) / Testnet (desarrollo)
Smart Contracts	Soroban Runtime
Protocolo	Protocol 25+
Consenso	Stellar Consensus Protocol (SCP)
Finalidad	~5 segundos
Costo	Fees muy bajos (~0.00001 XLM por operación)

8.2 Tokens Soportados

Todos los tokens nativos de Stellar son soportables en los vaults:

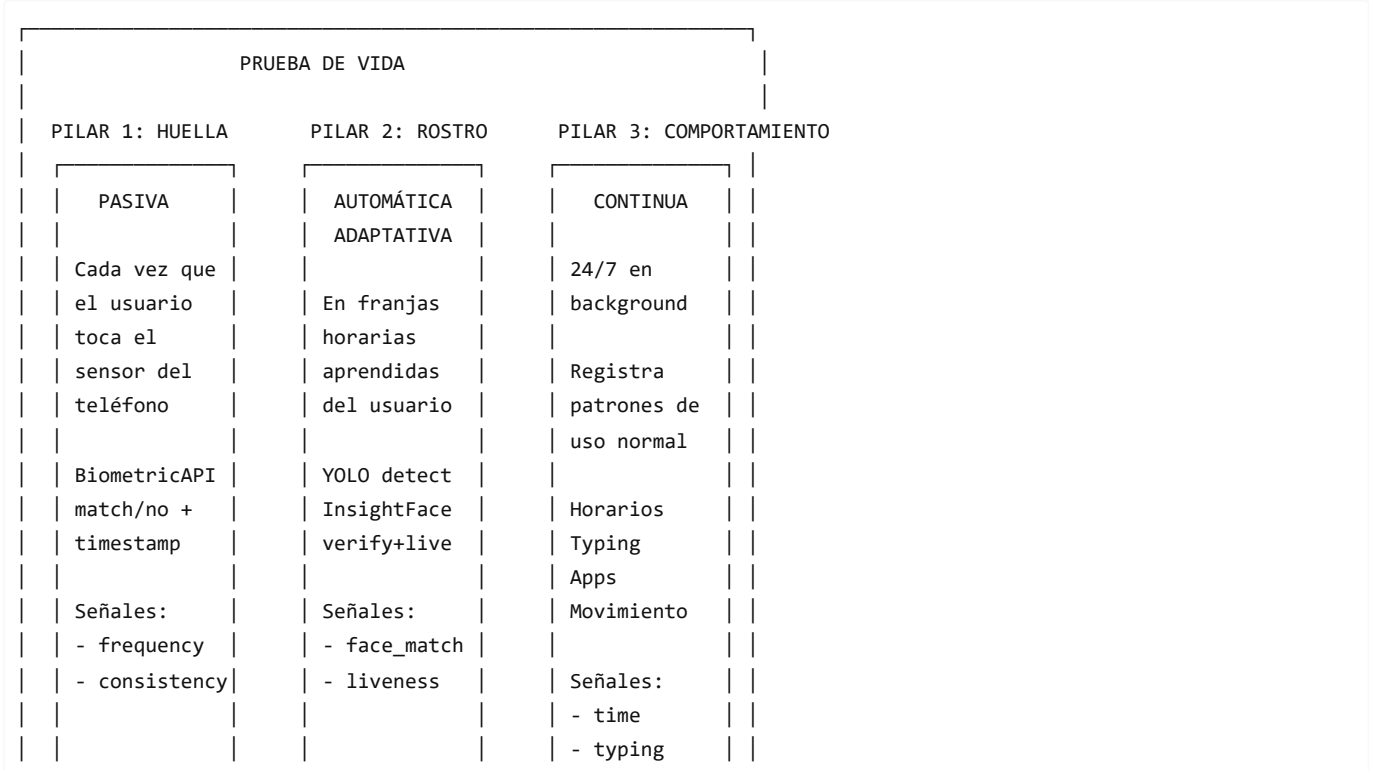
- **XLM** (Lumens, token nativo)
- **USDC** (Circle, principal stablecoin en Stellar)
- **Cualquier Stellar Asset** con trustline configurada
- **Token PULSE** (futuro, Fase 4)

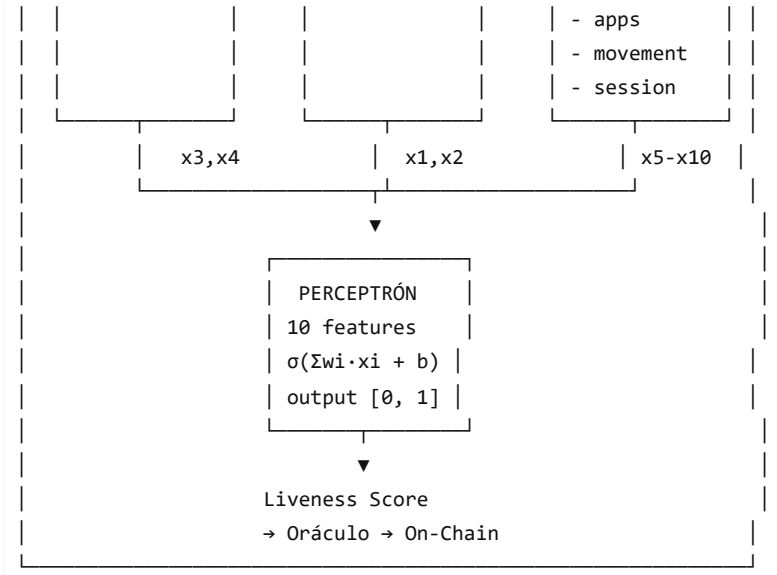
---

9. Sistema de Prueba de Vida

9.1 Los 3 Pilares

El sistema de prueba de vida se basa en **3 pilares de verificación** que operan simultáneamente:





9.2 Flujo Completo de Verificación

```
1. CAPTURA (en el dispositivo)
   ├── Huella: evento de BiometricAPI → timestamp + match/no
   ├── Rostro: cámara frontal en franja adaptativa → YOLO → InsightFace → scores
   └── Comportamiento: sensores + usage stats → features normalizadas

2. INFERENCIA LOCAL (en el dispositivo)
   ├── Construye vector x = [x1, x2, ..., x10]
   ├── Ejecuta perceptrón: z = Σ(wi * xi) + b
   ├── Aplica sigmoide: ŷ = 1 / (1 + e^(-z))
   └── ŷ ∈ (0, 1) = probabilidad de vida

3. ENVÍO AL ORÁCULO (GraphQL mutation)
   ├── Envía: features individuales + score del perceptrón
   └── El oráculo valida, agrega y publica

4. PUBLICACIÓN ON-CHAIN (por el oráculo)
   ├── submit_verification(user, score_u32, source, oracle_sig)
   └── Actualiza estado del vault si hay transición

5. TRANSICIÓN DE ESTADO (si aplica)
   ├── Score > 7000 → mantiene ACTIVE
   ├── 3000 < Score < 7000 → ALERT (si estaba ACTIVE)
   ├── Score < 3000 → GRACE (si estaba ALERT)
   └── Timeout 30 días en GRACE → TRIGGERED
```

9.3 Onboarding Biométrico

Paso a paso del registro:

```
1. INSTALACIÓN
   └── Usuario descarga e instala la app

2. CREACIÓN DE CUENTA
   ├── Crea wallet Stellar nueva o conecta existente
   └── Acepta términos y condiciones
```

```
3. REGISTRO DE HUELLA
   ├── La app solicita permiso de biometría
   ├── Usuario registra huella vía BiometricPrompt / LocalAuth
   └── Confirmación de registro exitoso

4. REGISTRO FACIAL
   ├── La app solicita permiso de cámara
   ├── Guía al usuario para capturar N fotos (frente, perfil, diferentes ángulos)
   ├── YOLO detecta rostro en cada frame
   ├── InsightFace genera embedding de referencia (512-d)
   ├── Se almacena SOLO el embedding en Secure Enclave
   └── NUNCA se almacenan las fotos

5. CALIBRACIÓN COMPORTAMENTAL (2-4 semanas)
   ├── El sistema recolecta datos de comportamiento en background
   ├── Todos los datos se etiquetan como "vivo" (y=1)
   ├── Se aprenden: horarios, velocidad typing, apps, movimiento
   ├── Se identifican franjas óptimas para verificación facial
   └── Al finalizar: perceptrón entrenado, pesos publicados on-chain

6. ACTIVACIÓN
   ├── Usuario crea vault y deposita activos
   ├── Configura beneficiarios (con porcentajes)
   ├── Registra documentos (opcional)
   └── Sistema pasa a estado ACTIVE con monitoreo continuo
```

---

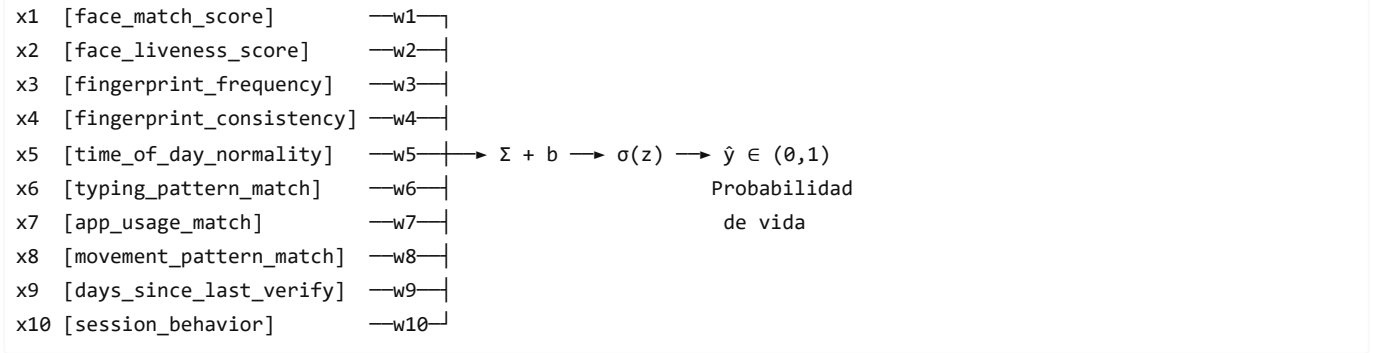
10. El Perceptrón: Modelo de IA

10.1 Fundamento

El corazón de Pulse Protocol es un **perceptrón personalizado** por usuario. Se eligió un perceptrón por:

1. **Interpretabilidad:** Los pesos tienen significado directo (importancia de cada señal).
2. **Eficiencia computacional:** Ejecutable en dispositivos móviles con batería limitada.
3. **Tamaño reducido:** Los pesos (10 + 1 bias = 11 valores) se almacenan on-chain económicamente.
4. **Robustez:** Menos propenso a overfitting que modelos más complejos.

10.2 Arquitectura



10.3 Formulación Matemática

Cálculo de la suma ponderada:

```
z = Σ(wi * xi) + b = w^T * x + b
```

Función de activación (sigmoide):

```
ŷ = σ(z) = 1 / (1 + e^(-z))
```

Donde:

- `x = [x1, x2, ..., x10]^T` — vector de features normalizadas, todas en [0, 1]
- `w = [w1, w2, ..., w10]^T` — vector de pesos (personalizado por usuario)
- `b` — término de sesgo (bias)
- `σ` — función sigmoide
- `ŷ ∈ (0, 1)` — probabilidad de que el usuario esté vivo

10.4 Vector de Features (10 dimensiones)

```
x = [
  x1   face_match_score         ∈ [0, 1]    ← InsightFace cosine similarity
  x2   face_liveness_score      ∈ [0, 1]    ← Anti-spoofing multi-señal
  x3   fingerprint_frequency    ∈ [0, 1]    ← matches_hoy / promedio_diario
  x4   fingerprint_consistency  ∈ [0, 1]    ← similitud patrón horario huella vs baseline
  x5   time_of_day_normality    ∈ [0, 1]    ← similitud distribución horaria vs baseline
  x6   typing_pattern_match     ∈ [0, 1]    ← similitud velocidad/ritmo vs baseline
  x7   app_usage_match          ∈ [0, 1]    ← similitud distribución apps vs baseline
  x8   movement_pattern_match   ∈ [0, 1]    ← similitud acelerómetro/giroscopio vs baseline
  x9   days_since_last_verify   ∈ [0, 1]    ← normalizado inverso (1=recién, 0=mucho tiempo)
  x10  session_behavior         ∈ [0, 1]    ← similitud duración/frecuencia sesiones vs baseline
]
```

10.5 Entrenamiento del Modelo

Fase de Calibración (2-4 semanas)

```
Algoritmo: Calibración Inicial del Perceptrón

1. D ← ∅                                    // Dataset vacío
2. Para cada día en período de calibración:
3.   Para cada interacción del usuario:
4.     Extraer features xt
5.     yt ← 1                               // Etiqueta: usuario vivo
6.     D ← D ∪ {(xt, yt)}
7. Inicializar w ← 0, b ← 0
8. Entrenar con descenso de gradiente:
9.   Para cada epoch:
10.    Para cada (x, y) ∈ D:
11.      ŷ ← σ(w^T * x + b)
12.      w ← w - η * (ŷ - y) * x
13.      b ← b - η * (ŷ - y)
14. return w, b
```

Función de Pérdida

Binary Cross-Entropy:

```
L(w, b) = -(1/N) * Σ[yi * log(ŷi) + (1 - yi) * log(1 - ŷi)]
```

Actualización Continua (Online Learning)

El modelo se adapta gradualmente post-calibración:

```
w(t+1) = w(t) - η * λ * (ŷt - yt) * xt
```

Donde `λ ∈ (0, 1)` es un **factor de olvido** que:

- Permite adaptación lenta a cambios genuinos de comportamiento.
- Resiste cambios abruptos sospechosos.
- Previene envenenamiento gradual del modelo.

10.6 Almacenamiento On-Chain de Pesos

Los pesos del perceptrón se almacenan en Soroban en **fixed-point de 6 decimales**:

```
w_stored = floor(w_real × 10^6)

Ejemplo: peso 0.847523 → almacenado como 847523 (i128)
Total on-chain: 10 pesos + 1 bias = 11 valores i128 = ~176 bytes
```

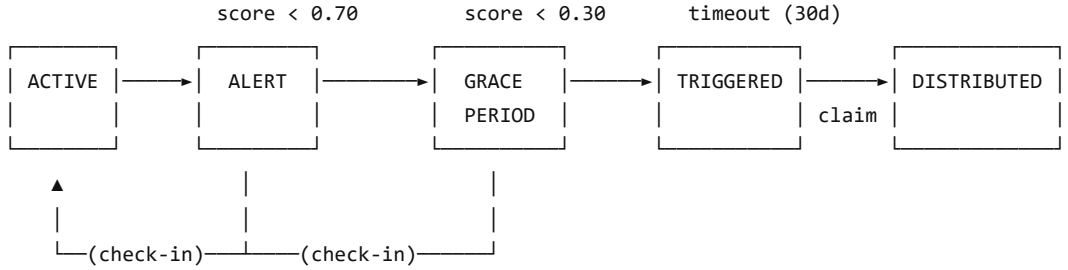
Motivación de almacenar on-chain:

- **Inmutabilidad auditada:** Historial completo de cambios de pesos.
- **Propiedad del usuario:** El modelo le pertenece al usuario.
- **Portabilidad:** Puede restaurarse en cualquier dispositivo nuevo.
- **Transparencia:** Los beneficiarios pueden auditar el sistema.

---

11. Máquina de Estados de Herencia

11.1 Diagrama de Estados



11.2 Descripción Detallada de Estados

Estado	Condición de entrada	Score	Verificación	Acciones Owner	Acciones Beneficiario
ACTIVE	Estado inicial post-calibración	> 0.70 (7000)	Normal (franjas adaptativas)	Depositar, retirar, modificar beneficiarios, registrar documentos	Ninguna
ALERT	Score cae por debajo de 0.70	0.30 - 0.70 (3000-7000)	Frecuencia x3	Depositar, emergency check-in. NO puede retirar ni modificar beneficiarios	Notificado (información solamente)
GRACE	Score cae por debajo de 0.30, o timeout prolongado en ALERT	< 0.30 (0-3000)	Frecuencia x10 + notificaciones push diarias	Solo emergency check-in	Notificado, puede preparar claim
TRIGGERED	Timeout de 30 días en GRACE sin	N/A (verificación	Se detiene	Totalmente bloqueado	Puede ejecutar `claim()`

	verificación exitosa	detenida)			
DISTRIBUTED	Todos los beneficiarios han reclamado	N/A	N/A	Estado terminal	Estado terminal

11.3 Thresholds por Defecto

Parámetro	Valor	Representación on-chain
Alert threshold	0.70	`7000` (u32)
Critical threshold	0.30	`3000` (u32)
Grace period	30 días	`30` (u32)

11.4 Frecuencia de Verificación por Estado

Estado	Frecuencia facial	Frecuencia comportamiento	Notificaciones
ACTIVE	Franjas adaptativas (~2-3x/día)	Continua (background)	Ninguna
ALERT	Franjas x3 (~6-9x/día)	Continua (más muestras)	Push semanal al owner
GRACE	Cada hora si hay actividad	Continua (máxima frecuencia)	Push diaria al owner + contactos de emergencia
TRIGGERED	Detenida	Detenida	Notificación a beneficiarios

11.5 Emergency Check-in

El usuario puede forzar un reset a ACTIVE desde ALERT o GRACE:

```
Emergency check-in:
├── Requiere: verificación facial exitosa (face_match > 0.9 + liveness > 0.8)
├── Requiere: autenticación biométrica del dispositivo (huella)
├── Resultado: score se resetea a 1.0, estado vuelve a ACTIVE
└── Se registra como VerificationSource::ManualCheckin on-chain
```

---

12. Reglas de Beneficiarios y Distribución

12.1 Modelo MVP: Porcentaje Fijo

En el MVP, las reglas de distribución son simples:

```
Beneficiary {
    address: Address,        // Wallet Stellar del beneficiario
    percentage: u32,         // 0-10000 (ej: 5000 = 50.00%)
}
```

12.2 Restricciones de Negocio

Regla	Descripción
Suma = 100%	La suma de todos los porcentajes DEBE ser exactamente 10000
Mínimo 1	Se requiere al menos 1 beneficiario para activar el vault
Solo en ACTIVE	Los beneficiarios solo se pueden modificar cuando el vault está en ACTIVE

Claim en TRIGGERED	Los beneficiarios solo pueden ejecutar `claim()` cuando el vault está en TRIGGERED
Un claim por beneficiario	Cada beneficiario puede reclamar una sola vez
Documentos incluidos	Al reclamar, el beneficiario también obtiene acceso a los documentos del vault

12.3 Flujo de Distribución

```
TRIGGERED activado
    │
    ▼
Vault Contract → crea/activa escrow en Trustless Work
    │
    ├── Milestone 1: Beneficiario A — 50% ($5,000 USDC)
    ├── Milestone 2: Beneficiario B — 30% ($3,000 USDC)
    └── Milestone 3: Beneficiario C — 20% ($2,000 USDC)
    │
    ▼
Cada beneficiario:
    ├── Llama claim() en Beneficiary Contract
    ├── Se valida: vault en TRIGGERED + beneficiario registrado + no ha reclamado
    ├── Se ejecuta: approve_milestone → release_funds (via Trustless Work C2C)
    ├── Fondos transferidos a wallet del beneficiario
    └── Acceso a documentos:
        ├── Públicos: CID de IPFS visible inmediatamente
        └── Privados: se revela clave de descifrado (cifrada con pubkey del beneficiario)
    │
    ▼
Cuando TODOS los beneficiarios han reclamado → estado DISTRIBUTED
```

12.4 Acceso a Documentos Post-Herencia

Tipo de documento	Al hacer claim
Público	Beneficiario ya tiene acceso (IPFS CID visible en el contrato)
Privado (cifrado)	El contrato revela la clave AES cifrada con la pubkey del beneficiario. El beneficiario la descifra con su private key y luego descifra el documento de IPFS

---

13. Sistema de Documentos (Herencia Documental)

13.1 Tipos de Documentos Soportados

Tipo	Enum	Descripción	Ejemplo
Manifiesto público	`PublicManifest`	Declaraciones públicas, principios	Manifiesto empresarial
Escritura	`Deed`	Títulos de propiedad digital	Escritura de propiedad
Testamento digital	`Will`	Instrucciones de herencia	Carta de voluntad
Certificado	`Certificate`	Certificados digitales	Título académico
Documento legal	`LegalDocument`	Documentos legales genéricos	Contrato, poder notarial
Carta personal	`PersonalLetter`	Correspondencia privada	Carta a hijos/familia

Otro	`Other`	Cualquier otro tipo	Fotos, videos, datos

13.2 Flujo de Registro

```
1. Usuario selecciona archivo desde el dispositivo
2. Elige tipo de documento (DocumentType)
3. Elige visibilidad: PÚBLICO o PRIVADO
4. Si PRIVADO:
   ├── App genera clave AES-256 aleatoria
   ├── Cifra documento con AES-256-GCM
   ├── Para cada beneficiario del vault:
   │   └── Cifra la clave AES con la pubkey Stellar del beneficiario
   └── Resultado: documento cifrado + N claves cifradas
5. App calcula SHA-256 del documento original (antes de cifrar)
6. App sube documento (cifrado o no) a IPFS → obtiene CID
7. App llama mutation registerDocument() al backend
8. Backend registra en DocumentRegistry Contract:
   ├── IPFS CID
   ├── SHA-256 hash
   ├── DocumentType
   ├── is_encrypted flag
   └── Metadata
9. Si privado: almacena claves cifradas on-chain (store_encrypted_key)
10. Vincula documento al vault (link_to_vault)
```

13.3 Validez Legal

Los documentos registrados en Pulse Protocol son **registros descentralizados** que proporcionan:

- **Prueba de existencia:** El hash SHA-256 demuestra que el documento existía en una fecha específica.
- **Prueba de integridad:** Cualquiera puede verificar que el documento no ha sido alterado comparando hashes.
- **Timestamp inmutable:** La blockchain proporciona un timestamp confiable.
- **No pretende validez legal formal:** No reemplaza notarización legal, pero sirve como evidencia sólida.

---

14. Integración con Trustless Work (Escrow)

14.1 Descripción

Trustless Work es un protocolo de Escrow-as-a-Service (EaaS) sobre Stellar/Soroban. Pulse Protocol integra Trustless Work mediante **invocación Contract-to-Contract (C2C)** directa desde los smart contracts de Soroban.

14.2 Mapping de Roles

Rol Trustless Work	Equivalente Pulse Protocol	Responsabilidad
`service_provider`	Oráculo de Pulse	Reporta prueba de vida, puede disputar
`approver`	ProofOfLife Contract	Valida scores y aprueba milestones
`release_signer`	Sistema Pulse (tras TRIGGERED)	Autoriza liberación de fondos
`dispute_resolver`	Sistema de testigos / emergencia	Arbitra disputas
`receiver`	Beneficiario(s)	Recibe los fondos de herencia
`platform_address`	Pulse Protocol	Cobra fee de plataforma

14.3 Flujo de Integración C2C

```
1. CREACIÓN DEL VAULT
   └── Usuario crea vault → Vault Contract despliega escrow en Trustless Work
       via factory contract (deploy)

2. DEPÓSITO
   └── Usuario deposita tokens → fondos van al escrow de Trustless Work
       via fund_escrow()

3. CONFIGURACIÓN
   └── Se crean milestones en el escrow, uno por beneficiario:
       Milestone { description: "Beneficiario A - 50%", amount: ..., status: "Pending" }

4. TRIGGERED (herencia activada)
   └── ProofOfLife confirma TRIGGERED → approve_milestone() para cada beneficiario

5. CLAIM
   └── Beneficiario llama claim() → release_funds() via C2C
       → fondos transferidos al beneficiario (menos fees)

6. FEES
   ├── Trustless Work fee: fee del protocolo TW
   └── Platform fee: fee de Pulse Protocol (configurable)
```

14.4 Estructura del Escrow por Vault

```
// Escrow creado por cada vault
Escrow {
    engagement_id: vault_id,
    title: "Pulse Protocol Inheritance - Vault #XXX",
    roles: Roles {
        approver: proof_of_life_contract_address,
        service_provider: oracle_address,
        platform_address: pulse_protocol_address,
        release_signer: vault_contract_address,
        dispute_resolver: witness_system_address,
        receiver: beneficiary_address,   // Para single-release
    },
    amount: vault_total_balance,
    platform_fee: pulse_fee_percentage,
    milestones: [
        Milestone { description: "Beneficiary A - 50%", status: "Pending", ... },
        Milestone { description: "Beneficiary B - 30%", status: "Pending", ... },
        Milestone { description: "Beneficiary C - 20%", status: "Pending", ... },
    ],
    trustline: Trustline { address: usdc_token_address },
}
```

---

15. Stack Tecnológico Completo

15.1 Tabla de Stack

Capa	Componente	Tecnología	Versión
Cliente Móvil	Framework	React Native	latest
	ML Runtime	TensorFlow Lite	latest
	Face Detection	YOLO v8-face / v9 (TFLite)	latest
	Face Verification	InsightFace / MobileFaceNet (TFLite)	latest
	GraphQL Client	Apollo Client for React Native	latest
	Wallet	@stellar/typescript-wallet-sdk + WalletConnect	latest
	Biometrics	react-native-biometrics	latest
	Camera	react-native-camera / expo-camera	latest
	Secure Storage	Secure Enclave (react-native-keychain)	latest
Backend	Lenguaje	Rust	stable
	HTTP Server	Actix-web	latest
	GraphQL API	async-graphql	latest
	WebSocket	actix-web-actors	latest
	ORM/DB	sqlx (async PostgreSQL)	latest
	Cache	redis-rs	latest
	Job Queue	Custom Redis-based workers	custom
	Stellar SDK	stellar-sdk (Rust)	latest
	IPFS Client	reqwest (HTTP a IPFS gateway)	latest
Smart Contracts	Plataforma	Soroban (Stellar)	Protocol 25+
	SDK	soroban-sdk	23.4.1
	Testing	soroban-sdk (testutils) + soroban-test-helpers	latest
	Escrow	Trustless Work Smart Escrow (C2C)	latest
Base de Datos	Relacional	PostgreSQL	16+
	Cache/Jobs	Redis	7+
	Event Store	PostgreSQL (tabla append-only)	custom
Almacenamiento	Documentos	IPFS + servicio de pinning	latest
	Estado	Stellar Ledger (Soroban storage)	—
Infraestructura	Hosting	Fly.io / Railway	—
	CI/CD	GitHub Actions	—
	Monitoreo	Grafana + Prometheus	—

---

16. Herramientas de Desarrollo Stellar

16.1 CLI y Scaffold

Herramienta	Comando de instalación	Versión actual	Uso
Stellar CLI	`winget install --id Stellar.StellarCLI` (Windows)	v25.1.0	Build, deploy, invoke contracts
Stellar Scaffold	`cargo install --locked stellar-scaffold-cli`	0.0.18	Scaffold full-stack (contracts + React + TS clients auto)
Stellar Registry	`cargo install --locked stellar-registry-cli`	latest	Publicar y gestionar WASM on-chain

16.2 Comandos Principales

```
# Scaffold: crear proyecto full-stack
stellar scaffold init pulse-protocol

# Build: compilar contratos + generar clientes TypeScript
stellar scaffold build

# Dev mode con auto-rebuild
stellar scaffold watch

# Compilar contrato individual
stellar contract build

# Deploy a testnet
stellar contract deploy \
  --wasm target/wasm32-unknown-unknown/release/vault.wasm \
  --source alice \
  --network testnet

# Invocar función de contrato
stellar contract invoke \
  --id <CONTRACT_ID> \
  --source alice \
  --network testnet \
  -- create_vault --owner <ADDRESS> --token <TOKEN_ADDRESS>

# Gestión de identidades
stellar keys generate alice --network testnet
stellar keys address alice

# Gestión de redes
stellar network add testnet --rpc-url https://soroban-testnet.stellar.org
```

16.3 SDKs

SDK	Package	Instalación	Uso
JS/TS principal	`@stellar/stellar-sdk`	`npm i @stellar/stellar-sdk`	TransactionBuilder, Contract, SorobanRpc

Wallet TS	`@stellar/typescript-wallet-sdk`	`npm i @stellar/typescript-wallet-sdk`	Gestión de wallets en React Native
Rust (contratos)	`soroban-sdk`	Cargo.toml: `soroban-sdk = "23.4.1"`	Desarrollo de smart contracts
Multi-wallet	`@creit.tech/stellar-wallets-kit`	`npm i @creit.tech/stellar-wallets-kit`	Abstracción multi-wallet
Trustless Work	`@trustless-work/escrow`	`npm i @trustless-work/escrow`	Hooks de escrow (si se necesitan en frontend)

16.4 Entornos

Gestionados via `environments.toml` (generado por `stellar scaffold`):

```toml
[development]
network = "standalone"
rpc_url = "http://localhost:8000"

[staging]
network = "testnet"
rpc_url = "https://soroban-testnet.stellar.org"

[production]
network = "mainnet"
rpc_url = "https://soroban.stellar.org"
```

---

17. Comunicación y Eventos (GraphQL + Subscriptions)

17.1 Arquitectura de Comunicación

```
React Native App
    │
    ├── HTTPS (queries + mutations)
    │    └── Apollo Client → async-graphql (Actix-web)
    │
    └── WSS (subscriptions)
         └── Apollo Client WebSocket → actix-web-actors → Event broker
```

17.2 Schema GraphQL Completo

```graphql
# ==================== TYPES ====================

type Vault {
  id: ID!
  owner: String!
  status: VaultStatus!
  beneficiaries: [Beneficiary!]!
  documents: [Document!]!
  balance: [TokenBalance!]!
  proofOfLifeContract: String!
  escrowContract: String
  createdAt: DateTime!
```

```
  lastUpdated: DateTime!
}

type Beneficiary {
  address: String!
  percentage: Int!       # 0-10000
  claimed: Boolean!
  claimedAt: DateTime
}

type Document {
  id: ID!
  ipfsCid: String!
  docHash: String!
  docType: DocumentType!
  isEncrypted: Boolean!
  metadata: JSON
  registeredAt: DateTime!
  vaultId: ID
}

type LivenessData {
  score: Int!             # 0-10000
  lastVerified: DateTime!
  calibrationComplete: Boolean!
  totalVerifications: Int!
  avgConfidence: Int!
  modelVersion: Int!
}

type VerificationRecord {
  timestamp: DateTime!
  score: Int!
  source: VerificationSource!
}

type TokenBalance {
  token: String!
  amount: String!        # String para precisión de i128
}

# ==================== ENUMS ====================

enum VaultStatus {
  ACTIVE
  ALERT
  GRACE_PERIOD
  TRIGGERED
  DISTRIBUTED
}

enum DocumentType {
  PUBLIC_MANIFEST
  DEED
  WILL
  CERTIFICATE
```

```
  LEGAL_DOCUMENT
  PERSONAL_LETTER
  OTHER
}

enum VerificationSource {
  FACIAL_RECOGNITION
  FINGERPRINT
  BEHAVIOR_PATTERN
  PERCEPTRON_AGGREGATE
  MANUAL_CHECKIN
  WITNESS_ATTESTATION
}

# ==================== INPUTS ====================

input CreateVaultInput {
  token: String!
  initialDeposit: String
}

input BeneficiaryInput {
  address: String!
  percentage: Int!        # 0-10000, suma debe ser 10000
}

input RegisterDocumentInput {
  ipfsCid: String!
  docHash: String!
  docType: DocumentType!
  isEncrypted: Boolean!
  metadata: JSON
  vaultId: ID
  encryptedKeys: [EncryptedKeyInput!]  # Si es privado
}

input EncryptedKeyInput {
  beneficiaryAddress: String!
  encryptedKey: String!               # Base64
}

input VerificationInput {
  facMatchScore: Int         # 0-10000
  faceLivenessScore: Int
  fingerprintFrequency: Int
  fingerprintConsistency: Int
  timeOfDayNormality: Int
  typingPatternMatch: Int
  appUsageMatch: Int
  movementPatternMatch: Int
  daysSinceLastVerify: Int
  sessionBehavior: Int
  perceptronOutput: Int      # Score final del perceptrón
}

# ==================== QUERIES ====================
```

```
type Query {
  vault(id: ID!): Vault
  myVaults: [Vault!]!
  livenessScore(userId: ID!): LivenessData
  verificationHistory(userId: ID!, limit: Int = 50): [VerificationRecord!]!
  beneficiaries(vaultId: ID!): [Beneficiary!]!
  documents(vaultId: ID!): [Document!]!
  document(docId: ID!): Document
  canClaim(vaultId: ID!): Boolean!
}

# ==================== MUTATIONS ====================

type Mutation {
  createVault(input: CreateVaultInput!): Vault!
  deposit(vaultId: ID!, amount: String!, token: String!): TransactionResult!
  withdraw(vaultId: ID!, amount: String!, token: String!): TransactionResult!
  setBeneficiaries(vaultId: ID!, beneficiaries: [BeneficiaryInput!]!): [Beneficiary!]!
  registerDocument(input: RegisterDocumentInput!): Document!
  linkDocumentToVault(docId: ID!, vaultId: ID!): Boolean!
  submitVerification(input: VerificationInput!): VerificationResult!
  emergencyCheckin: CheckinResult!
  updateModelWeights(weights: [String!]!, bias: String!): ModelUpdateResult!
  claimInheritance(vaultId: ID!): ClaimResult!
}

# ==================== SUBSCRIPTIONS ====================

type Subscription {
  vaultStatusChanged(vaultId: ID!): VaultStatusEvent!
  livenessAlert(userId: ID!): LivenessAlertEvent!
  claimStarted(vaultId: ID!): ClaimEvent!
  verificationCompleted(userId: ID!): VerificationEvent!
  documentAccessGranted(beneficiaryId: ID!): DocumentAccessEvent!
}

# ==================== SUBSCRIPTION EVENTS ====================

type VaultStatusEvent {
  vaultId: ID!
  previousStatus: VaultStatus!
  newStatus: VaultStatus!
  timestamp: DateTime!
  triggerScore: Int
}

type LivenessAlertEvent {
  userId: ID!
  level: AlertLevel!
  currentScore: Int!
  message: String!
  timestamp: DateTime!
}

type ClaimEvent {
```

```
  vaultId: ID!
  beneficiaryAddress: String!
  amount: String!
  timestamp: DateTime!
}

type VerificationEvent {
  userId: ID!
  score: Int!
  source: VerificationSource!
  timestamp: DateTime!
}

type DocumentAccessEvent {
  docId: ID!
  beneficiaryAddress: String!
  ipfsCid: String!
  encryptedKey: String
  timestamp: DateTime!
}

enum AlertLevel {
  INFO
  WARNING
  CRITICAL
}
```

---

18. Base de Datos y Capa de Datos

18.1 PostgreSQL - Modelo Relacional

Tablas principales:

```
-- Usuarios
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    stellar_address VARCHAR(56) UNIQUE NOT NULL,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    calibration_complete BOOLEAN NOT NULL DEFAULT FALSE,
    calibration_started_at TIMESTAMPTZ
);

-- Vaults (cache local del estado on-chain)
CREATE TABLE vaults (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    contract_id VARCHAR(56) UNIQUE NOT NULL,  -- Soroban contract ID
    owner_id UUID NOT NULL REFERENCES users(id),
    status VARCHAR(20) NOT NULL DEFAULT 'active',
    escrow_contract_id VARCHAR(56),
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    last_synced_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Beneficiarios (cache local)
CREATE TABLE beneficiaries (
```

```
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    vault_id UUID NOT NULL REFERENCES vaults(id),
    stellar_address VARCHAR(56) NOT NULL,
    percentage INT NOT NULL CHECK (percentage > 0 AND percentage <= 10000),
    claimed BOOLEAN NOT NULL DEFAULT FALSE,
    claimed_at TIMESTAMPTZ,
    UNIQUE(vault_id, stellar_address)
);

-- Documentos
CREATE TABLE documents (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    owner_id UUID NOT NULL REFERENCES users(id),
    vault_id UUID REFERENCES vaults(id),
    ipfs_cid VARCHAR(100) NOT NULL,
    doc_hash VARCHAR(64) NOT NULL,          -- SHA-256 hex
    doc_type VARCHAR(30) NOT NULL,
    is_encrypted BOOLEAN NOT NULL DEFAULT FALSE,
    metadata JSONB,
    contract_doc_id VARCHAR(100),           -- ID en el contrato Soroban
    registered_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Claves cifradas de documentos privados
CREATE TABLE document_encrypted_keys (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    document_id UUID NOT NULL REFERENCES documents(id),
    beneficiary_address VARCHAR(56) NOT NULL,
    encrypted_key TEXT NOT NULL,            -- Base64 encoded
    revealed BOOLEAN NOT NULL DEFAULT FALSE,
    UNIQUE(document_id, beneficiary_address)
);

-- Verificaciones (historial local)
CREATE TABLE verifications (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users(id),
    score INT NOT NULL CHECK (score >= 0 AND score <= 10000),
    source VARCHAR(30) NOT NULL,
    face_match_score INT,
    face_liveness_score INT,
    fingerprint_frequency INT,
    fingerprint_consistency INT,
    time_of_day_normality INT,
    typing_pattern_match INT,
    app_usage_match INT,
    movement_pattern_match INT,
    days_since_last_verify INT,
    session_behavior INT,
    perceptron_output INT,
    on_chain_tx_hash VARCHAR(64),
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Modelo del perceptrón (cache local)
CREATE TABLE user_models (
```

```
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID UNIQUE NOT NULL REFERENCES users(id),
    weights JSONB NOT NULL,               -- Array de i128 como strings
    bias VARCHAR(30) NOT NULL,
    version INT NOT NULL DEFAULT 1,
    calibration_complete BOOLEAN NOT NULL DEFAULT FALSE,
    last_updated TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Índices
CREATE INDEX idx_verifications_user_created ON verifications(user_id, created_at DESC);
CREATE INDEX idx_vaults_owner ON vaults(owner_id);
CREATE INDEX idx_documents_vault ON documents(vault_id);
CREATE INDEX idx_beneficiaries_vault ON beneficiaries(vault_id);
```

18.2 Redis - Cache y Jobs

Uso de Redis:

Clave	Tipo	TTL	Uso
`user:{id}:score`	String (int)	5 min	Cache del último liveness score
`user:{id}:session`	Hash	24h	Datos de sesión activa
`vault:{id}:status`	String	5 min	Cache del estado del vault
`rate_limit:verify:{user_id}`	Counter	1 hora	Rate limiting de verificaciones
`job:check_timeouts`	Sorted Set	—	Job queue para verificación de timeouts
`job:send_notifications`	List	—	Cola de notificaciones pendientes
`pubsub:vault:{id}`	PubSub channel	—	Canal de eventos para subscriptions
`pubsub:user:{id}`	PubSub channel	—	Canal de alertas de liveness

18.3 Event Store - Auditoría Completa

Tabla append-only en PostgreSQL para auditoría completa:

```
CREATE TABLE events (
    id BIGSERIAL PRIMARY KEY,
    event_type VARCHAR(50) NOT NULL,
    aggregate_type VARCHAR(30) NOT NULL,   -- 'vault', 'user', 'document'
    aggregate_id UUID NOT NULL,
    payload JSONB NOT NULL,
    metadata JSONB,                        -- IP, device info, etc.
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    on_chain_tx_hash VARCHAR(64)           -- Si el evento se publicó on-chain
);

CREATE INDEX idx_events_aggregate ON events(aggregate_type, aggregate_id, created_at);
CREATE INDEX idx_events_type ON events(event_type, created_at);
```

Tipos de eventos registrados:

Evento	aggregate_type	Descripción
`VaultCreated`	vault	Vault creado
`DepositMade`	vault	Depósito de tokens
`WithdrawalMade`	vault	Retiro de tokens
`BeneficiariesUpdated`	vault	Beneficiarios modificados
`StatusTransition`	vault	Cambio de estado (ACTIVE→ALERT, etc.)
`VerificationSubmitted`	user	Verificación de prueba de vida enviada
`VerificationPublished`	user	Verificación publicada on-chain
`ModelUpdated`	user	Pesos del perceptrón actualizados
`CalibrationCompleted`	user	Calibración finalizada
`EmergencyCheckin`	user	Check-in de emergencia ejecutado
`DocumentRegistered`	document	Documento registrado
`DocumentLinkedToVault`	document	Documento vinculado a vault
`GracePeriodStarted`	vault	Período de gracia iniciado
`HeritageTriggered`	vault	Herencia activada
`ClaimExecuted`	vault	Beneficiario reclamó su parte
`DocumentAccessGranted`	document	Acceso a documento revelado
`AllClaimsCompleted`	vault	Distribución completa

---

19. Privacidad y Seguridad

19.1 Principios de Privacidad (Privacy by Design)

NUNCA salen del dispositivo:

- Imágenes faciales o frames de cámara
- Datos raw de huella dactilar
- Ubicación GPS exacta
- Contenido de mensajes o aplicaciones
- Historial de navegación
- Datos biométricos sin procesar
- Face embeddings de referencia

SÍ se procesan/envían (al backend/oráculo):

- Scores normalizados (0-1 / 0-10000) sin contexto
- Pesos del modelo (no reversibles a datos personales)
- Timestamps de verificación
- Hashes de patrones (no los patrones en sí)
- Resultados de face match (score numérico, no la imagen)
- Frecuencia de uso del sensor biométrico (no los datos del sensor)

19.2 Modelo de Amenazas

Amenaza	Descripción	Mitigación
Suplantación facial	Atacante usa foto/video del usuario	Liveness detection multi-señal (parpadeo, micro-movimientos, textura, análisis temporal)
Robo de dispositivo	Atacante obtiene acceso al teléfono	Patrones de comportamiento detectan anomalías + biometría del dispositivo
Colusión de testigos	Testigos declaran muerte falsa	Requiere múltiples fuentes independientes + período de gracia largo (30d)
Oráculo malicioso	Oráculo reporta datos falsos	Verificación on-chain de firmas + múltiples oráculos (futuro)
Ataque al modelo	Envenenamiento gradual de pesos	Factor de olvido λ limita tasa de cambio + auditoría de actualizaciones
Replay attack	Reutilización de verificaciones antiguas	Timestamps + nonces + verificación de frescura
Man-in-the-middle	Interceptación de comunicación app-backend	HTTPS/WSS + certificate pinning

19.3 Anti-Spoofing (Liveness Detection)

El módulo implementa múltiples técnicas:

1. **Detección de parpadeo:** Frecuencia y naturalidad del parpadeo.
2. **Micro-movimientos:** Pequeños movimientos involuntarios del rostro.
3. **Análisis de textura:** Detecta pantallas/fotos impresas vs piel real.
4. **Análisis temporal:** Secuencias de frames para detectar videos pregrabados.
5. **Consistencia multi-señal:** Face score debe correlacionar con fingerprint y comportamiento.

---

20. Modelo de Negocio

20.1 Modelo Híbrido de 4 Vías

Pulse Protocol monetiza mediante 4 mecanismos complementarios:

1. Freemium + Premium

Tier	Precio	Incluye
Free	$0	1 vault, 1 beneficiario, solo tokens, check-in manual (sin IA)
Pro	~$5-10/mes	Vaults ilimitados, hasta 5 beneficiarios, documentos (IPFS), prueba de vida completa (3 pilares + IA), notificaciones avanzadas
Enterprise	Custom	Multi-firma, condiciones avanzadas, API dedicada, soporte prioritario, SLA

2. Fees por Transacción

Operación	Fee
Depósito al vault	~0.1% del monto
Claim de herencia	~0.3% del monto
Registro de documento	Fee fijo pequeño (~$0.50-$1.00 equivalente)

3. Suscripción (Monitoreo Activo)

- La prueba de vida con IA requiere infraestructura del oráculo (servidores, validación, publicación on-chain).
- Los tiers Pro y Enterprise pagan por este servicio continuo de monitoreo.
- Incluye: storage de verificaciones, procesamiento de signals, notificaciones push.

4. Token PULSE (Fase 4 - Futuro)

- Emitido como **Stellar Asset** (SAP-0010).
- **Staking:** Operadores de oráculo hacen stake de PULSE como garantía.
- **Governance:** Holders votan sobre parámetros del protocolo (fees, thresholds, upgrades).
- **Descuentos:** Pagar servicios con PULSE ofrece descuento sobre USD.
- **Burn mechanism:** Parte de los fees se usan para quemar PULSE (deflación).

---

21. Estructura de Directorios del Proyecto

```
pulse-protocol/
│
├── contracts/                          # Smart Contracts Soroban (Rust)
│   ├── vault/
│   │   ├── src/
│   │   │   └── lib.rs                # Vault Contract
│   │   └── Cargo.toml
│   ├── proof-of-life/
│   │   ├── src/
│   │   │   └── lib.rs                # Proof of Life Contract
│   │   └── Cargo.toml
│   ├── beneficiary/
│   │   ├── src/
│   │   │   └── lib.rs                # Beneficiary Contract
│   │   └── Cargo.toml
│   ├── document-registry/
│   │   ├── src/
│   │   │   └── lib.rs                # Document Registry Contract
│   │   └── Cargo.toml
│   └── Cargo.toml                    # Workspace
│
├── oracle/                             # Backend / Oráculo (Rust)
│   ├── src/
│   │   ├── main.rs                   # Entry point + Actix-web setup
│   │   ├── graphql/
│   │   │   ├── mod.rs
│   │   │   ├── schema.rs            # GraphQL schema (async-graphql)
│   │   │   ├── queries.rs           # Query resolvers
│   │   │   ├── mutations.rs         # Mutation resolvers
│   │   │   └── subscriptions.rs     # Subscription resolvers
│   │   ├── services/
│   │   │   ├── aggregator.rs        # Agregación de señales
│   │   │   ├── publisher.rs         # Publicación on-chain
│   │   │   ├── notification.rs      # Sistema de notificaciones
│   │   │   └── ipfs.rs             # Cliente IPFS
│   │   ├── models/
│   │   │   ├── vault.rs
│   │   │   ├── user.rs
│   │   │   ├── verification.rs
│   │   │   ├── document.rs
│   │   │   └── event.rs
```

```
│   │   ├── jobs/
│   │   │   ├── timeout_checker.rs    # Verifica timeouts de estados
│   │   │   ├── chain_sync.rs         # Sincroniza estado on-chain
│   │   │   └── worker.rs             # Worker de Redis jobs
│   │   ├── db/
│   │   │   ├── mod.rs
│   │   │   ├── postgres.rs           # Conexión PostgreSQL
│   │   │   ├── redis.rs              # Conexión Redis
│   │   │   └── migrations/           # Migraciones SQL
│   │   └── config.rs                 # Configuración de entorno
│   └── Cargo.toml
│
├── mobile/                             # App React Native
│   ├── src/
│   │   ├── App.tsx                   # Entry point
│   │   ├── screens/
│   │   │   ├── Onboarding/           # Registro + biometría
│   │   │   ├── Dashboard/            # Vista principal
│   │   │   ├── Vault/               # Gestión de vaults
│   │   │   ├── Documents/           # Gestión de documentos
│   │   │   ├── Beneficiaries/       # Gestión de beneficiarios
│   │   │   ├── Settings/            # Configuración
│   │   │   └── Claim/              # Vista de claim (beneficiarios)
│   │   ├── services/
│   │   │   ├── vision/
│   │   │   │   ├── yolo.ts          # Detección facial (YOLO)
│   │   │   │   ├── insightface.ts   # Verificación facial (InsightFace)
│   │   │   │   └── liveness.ts      # Anti-spoofing
│   │   │   ├── biometrics/
│   │   │   │   └── fingerprint.ts   # Integración BiometricPrompt/LocalAuth
│   │   │   ├── patterns/
│   │   │   │   ├── behavior.ts      # Análisis de patrones
│   │   │   │   ├── typing.ts        # Patrón de escritura
│   │   │   │   └── movement.ts      # Patrón de movimiento
│   │   │   ├── perceptron/
│   │   │   │   ├── model.ts         # Perceptrón (inferencia)
│   │   │   │   ├── training.ts      # Entrenamiento on-device
│   │   │   │   └── calibration.ts   # Lógica de calibración
│   │   │   ├── documents/
│   │   │   │   ├── ipfs.ts          # Upload/download IPFS
│   │   │   │   └── encryption.ts    # Cifrado AES-256-GCM
│   │   │   ├── wallet/
│   │   │   │   └── stellar.ts       # Integración wallet Stellar
│   │   │   └── graphql/
│   │   │       ├── client.ts        # Apollo Client setup
│   │   │       ├── queries.ts       # GraphQL queries
│   │   │       ├── mutations.ts     # GraphQL mutations
│   │   │       └── subscriptions.ts # GraphQL subscriptions
│   │   ├── hooks/                    # React hooks personalizados
│   │   ├── components/               # Componentes UI reutilizables
│   │   ├── navigation/               # React Navigation setup
│   │   ├── store/                    # Estado local (Zustand/Redux)
│   │   ├── utils/                    # Utilidades
│   │   └── types/                    # TypeScript types
│   ├── models/                       # Modelos TFLite/ONNX
│   │   ├── yolo_face.tflite         # YOLO face detection
```

```
│   │   ├── mobilefacenet.tflite    # InsightFace verification
│   │   └── liveness.tflite         # Liveness detection
│   ├── package.json
│   └── tsconfig.json
│
├── ml/                               # Scripts de ML (desarrollo)
│   ├── perceptron/
│   │   ├── train.py                # Entrenamiento del perceptrón
│   │   ├── inference.py            # Inferencia de prueba
│   │   └── export_tflite.py        # Exportación a TFLite
│   ├── face/
│   │   ├── export_insightface.py   # Exportación de InsightFace a TFLite
│   │   └── test_verification.py    # Tests de verificación facial
│   └── requirements.txt
│
├── docs/
│   ├── CLAUDE.md                   # Contexto rápido para Claude
│   ├── CLAUDE_CODE_CONTEXT.md      # Contexto detallado de desarrollo
│   ├── Pulse_Protocol_Propuesta.md  # ESTE DOCUMENTO
│   └── pulse_whitepaper.pdf        # Whitepaper original
│
├── environments.toml                # Configuración multi-entorno (Stellar scaffold)
├── .env.example                     # Variables de entorno ejemplo
└── README.md
```

---

22. Convenciones de Código

22.1 Rust (Contracts + Backend)

Convención	Detalle
Naming	`snake_case` para funciones, variables. `PascalCase` para tipos/structs. `SCREAMING_SNAKE` para constantes.
Documentación	Documentar todas las funciones públicas con `///`
Decimales	Fixed-point con 6 decimales (multiplicar por `1_000_000`)
Scores	`u32` en rango 0-10000 (representa 0.00%-100.00%)
Errores	Enums de error descriptivos con códigos numéricos
Testing	Tests unitarios con `#[test]` + `soroban-sdk testutils`
Datos biométricos	NUNCA almacenar datos raw. Solo scores normalizados.

22.2 TypeScript (Mobile)

Convención	Detalle
Naming	`camelCase` para funciones/variables. `PascalCase` para componentes/tipos.
Types	Estricto. No usar `any`. Definir interfaces para todo.
GraphQL	Tipos auto-generados desde el schema.
Estado	Hooks + Context o Zustand para estado global.
Async	`async/await` siempre. No callbacks.

22.3 General

Regla	Detalle
Commits	Conventional Commits (`feat:`, `fix:`, `docs:`, `refactor:`)
Branches	`feature/xxx`, `fix/xxx`, `release/vX.X.X`
PR	Requerido para merge a `main`. Mínimo 1 review.
Secrets	Nunca commitear `.env`, credenciales, o API keys

---

23. Glosario

Término	Definición
Liveness Detection	Técnica para verificar que un rostro pertenece a una persona real presente, no a una foto o video.
Perceptrón	Modelo simple de red neuronal con una sola capa, utilizado para clasificación binaria.
Soroban	Plataforma de smart contracts de Stellar, escrita en Rust.
Oráculo	Servicio que proporciona datos externos a una blockchain.
Vault	Contrato inteligente que custodia activos del usuario.
Prueba de Vida	Verificación de que el propietario de un vault sigue vivo y activo.
InsightFace	Framework open-source de análisis facial que incluye face detection, recognition y alignment.
YOLO	"You Only Look Once" — arquitectura de detección de objetos en tiempo real.
Face Embedding	Representación vectorial (512-d) de un rostro que permite comparación por similitud.
Cosine Similarity	Métrica de similitud entre vectores. Usada para comparar face embeddings.
Fixed-point	Representación de decimales como enteros multiplicados por una potencia de 10.
CID	Content Identifier — dirección única de un contenido en IPFS.
Trustless Work	Protocolo de Escrow-as-a-Service sobre Stellar/Soroban.
C2C	Contract-to-Contract — invocación directa entre smart contracts en Soroban.
BiometricPrompt	API de Android para autenticación biométrica (huella, face).
LocalAuthentication	Framework de iOS para autenticación biométrica (Touch ID, Face ID).
AES-256-GCM	Algoritmo de cifrado simétrico con autenticación. Usado para cifrar documentos privados.
Online Learning	Entrenamiento continuo del modelo con nuevos datos, sin re-entrenar desde cero.
Grace Period	Período de espera antes de activar la herencia, para prevenir falsos positivos.
Event Store	Base de datos append-only que registra cada evento del sistema para auditoría completa.
Stellar Asset	Token emitido en la red Stellar.
Trustline	Autorización que un usuario da para aceptar un Stellar Asset específico.
XDR	External Data Representation — formato binario usado por Stellar para serializar transacciones.

IPFS	InterPlanetary File System — red de almacenamiento descentralizado de archivos.

Documento generado: Febrero 2026 Pulse Protocol — Tu legado digital, protegido por inteligencia artificial.