```
import { Injectable, OnModuleInit } from '@nestjs/common';
import * as admin from 'firebase-admin';
import { ConfigService } from '@nestjs/config';

@Injectable()
export class FirestoreService implements OnModuleInit {
  private db: admin.firestore.Firestore;

  constructor(private configService: ConfigService) {}

  onModuleInit() {
    // Em um ambiente real, as credenciais viriam do ConfigService ou
variáveis de ambiente
    // Para este desenvolvimento, preparamos a estrutura para
inicialização
    if (!admin.apps.length) {
      admin.initializeApp({
        credential: admin.credential.applicationDefault(),
      });
    }
    this.db = admin.firestore();
  }

  getDb(): admin.firestore.Firestore {
    return this.db;
  }

  /**
   * Executa uma operação dentro de uma transação Firestore.
   * Essencial para garantir a integridade do encadeamento de hashes
na auditoria.
   */
  async runTransaction<T>(updateFunction: (transaction:
admin.firestore.Transaction) => Promise<T>): Promise<T> {
    return this.db.runTransaction(updateFunction);
  }

  /**
   * Helper para garantir o isolamento por tenantId em todas as
queries.
   * Implementa a Reivindicação 2 da Patente.
   */
  collection(path: string, tenantId: string) {
    return this.db.collection(path).where('tenantId', '==', tenantId);
  }
}




import { AuditService } from './src/kernel/audit.service';
import { EventOrchestratorService, EventIntent } from
'./src/kernel/event-orchestrator.service';

async function testKernel() {
  const auditService = new AuditService();
  const orchestrator = new EventOrchestratorService(auditService);

  const tenantId = 'FAZENDA_MODELO_01';
  const actor = {
    uid: 'user_123',
```

```
    role: 'OPERATOR',
    permissions: ['stock_withdraw.execute', 'milk_collection.execute']
  };

  console.log('--- TESTE 1: Execução de Evento Crítico com Evidência -
--');
  const intent1: EventIntent = {
    type: 'STOCK_WITHDRAW',
    tenantId,
    actor,
    payload: { itemId: 'SAL_MINERAL', quantity: 50 },
    evidenceIds: ['photo_001.jpg']
  };

  try {
    const res1 = await orchestrator.execute(intent1, async () => {
      return { status: 'Estoque atualizado', newBalance: 150 };
    });
    console.log('Resultado 1:', res1);
  } catch (e) {
    console.error('Erro 1:', e.message);
  }

  console.log('\n--- TESTE 2: Tentativa de Evento Crítico SEM
Evidência (Deve Falhar) ---');
  const intent2: EventIntent = {
    type: 'STOCK_WITHDRAW',
    tenantId,
    actor,
    payload: { itemId: 'SAL_MINERAL', quantity: 10 },
    evidenceIds: [] // Sem evidência
  };

  try {
    await orchestrator.execute(intent2, async () => {});
  } catch (e) {
    console.log('Sucesso no Teste: O sistema bloqueou a operação sem
evidência conforme a patente.');
    console.log('Mensagem de erro:', e.message);
  }

  console.log('\n--- TESTE 3: Verificação de Integridade da Auditoria
---');
  const isIntegrityOk = auditService.verifyIntegrity(tenantId);
  console.log('Integridade da Cadeia de Hashes:', isIntegrityOk ?
'VÁLIDA ✅' : 'CORROMPIDA ❌');

  // Simulação de corrupção
  console.log('\n--- TESTE 4: Simulação de Adulteração (Deve Detectar)
---');
  // @ts-ignore - Acessando privado para simular ataque
  auditService.auditTrail[0].meta.quantity = 9999;
  const isIntegrityAfterAttack =
auditService.verifyIntegrity(tenantId);
  console.log('Integridade após adulteração:', isIntegrityAfterAttack
? 'VÁLIDA ❌' : 'CORROMPIDA ✅ (Ataque detectado!)');
}

testKernel();
```

```
import { Injectable, UnauthorizedException, BadRequestException } from
'@nestjs/common';
import { AuditService } from './audit.service';

export interface EventIntent {
  type: string;
  tenantId: string;
  actor: {
    uid: string;
    role: string;
    permissions: string[];
  };
  payload: any;
  evidenceIds?: string[];
}

@Injectable()
export class EventOrchestratorService {
  constructor(private readonly auditService: AuditService) {}

  /**
   * O "Coração" do Sistema: Orquestra a validação e execução de
eventos.
   * Segue a sequência obrigatória da Patente (Reivindicação 1 e 12).
   */
  async execute(intent: EventIntent, action: () => Promise<any>) {
    console.log(`[Orchestrator] Iniciando processamento de evento:
${intent.type}`);

    // 1. Autenticação & Tenant Isolation (Reivindicação 1.b, 1.d)
    if (!intent.actor.uid || !intent.tenantId) {
      throw new UnauthorizedException('Identidade ou Organização não
fornecida.');
    }

    // 2. RBAC / Permissões (Reivindicação 1.c)
    // Exemplo simplificado: verifica se o ator tem permissão para o
tipo de evento
    const requiredPermission = `${intent.type.toLowerCase()}.execute`;
    if (!intent.actor.permissions.includes(requiredPermission)) {
      throw new UnauthorizedException(`Permissão insuficiente:
${requiredPermission}`);
    }

    // 3. Motor de Regras (Pré-execução) (Reivindicação 1.e)
    // Aqui entraria a lógica de validação de limites, saldos, etc.
    this.validateRules(intent);

    // 4. Máquina de Estados (Reivindicação 1.f)
    // Valida se a transição de estado é permitida
    this.validateStateTransition(intent);

    // 5. Evidência Obrigatória (Reivindicação 1.g)
    // Verifica se eventos críticos possuem evidência
    this.validateEvidence(intent);

    // 6. Auditoria Imutável (Reivindicação 1.h)
    // Registra o evento ANTES da execução final para garantir o
lastro
```

```
    const auditEntry = await this.auditService.logEvent(
      intent.tenantId,
      intent.type,
      { uid: intent.actor.uid, role: intent.actor.role },
      intent.payload,
      intent.evidenceIds,
    );

    // 7. Execução da Lógica de Negócio
    try {
      const result = await action();

      console.log(`[Orchestrator] Evento ${intent.type} executado com
sucesso.`);
      return {
        success: true,
        auditId: auditEntry.auditId,
        eventHash: auditEntry.eventHash,
        result,
      };
    } catch (error) {
      console.error(`[Orchestrator] Erro na execução do evento
${intent.type}:`, error);
      throw new BadRequestException(`Falha na execução:
${error.message}`);
    }
  }

  private validateRules(intent: EventIntent) {
    // Simulação de regras: por exemplo, não permitir retiradas
negativas
    if (intent.type === 'STOCK_WITHDRAW' && intent.payload.quantity <=
0) {
      throw new BadRequestException('Quantidade de retirada deve ser
positiva.');
    }
  }

  private validateStateTransition(intent: EventIntent) {
    // Simulação de máquina de estados
    console.log(`[State] Validando transição para ${intent.type}...
OK`);
  }

  private validateEvidence(intent: EventIntent) {
    // Regra da Patente: Eventos críticos exigem evidência (Tipo B)
    const criticalEvents = ['STOCK_WITHDRAW', 'PAYMENT_RELEASE',
'ASSET_TRANSFER'];
    if (criticalEvents.includes(intent.type) && (!intent.evidenceIds
|| intent.evidenceIds.length === 0)) {
      throw new BadRequestException(`Evidência obrigatória não
fornecida para evento crítico: ${intent.type}`);
    }
  }
}
```

Detalhamento do Event Orchestrator e Auditoria Imutável - Kernel Ciclo+

Este documento aprofunda a arquitetura do Event Orchestrator e da Auditoria Imutável no Kernel do sistema Ciclo+. Estes componentes são o coração da inovação patenteada (BR 10 2026 002755 3), garantindo que cada operação seja validada rigorosamente e registrada de forma inalterável.

1. O Event Orchestrator: O Guardião das Operações

O Event Orchestrator é o ponto de entrada unificado para todas as operações transacionais no sistema Ciclo+. Ele atua como um dispatcher central, garantindo que cada "intenção" (requisição de uma ação) passe por uma cadeia técnica obrigatória de validação antes de ser executada. Este mecanismo é fundamental para cumprir as reivindicações de método da patente.

1.1. Cadeia de Validação Obrigatória

Cada evento submetido ao Orchestrator segue a sequência exata de validações definida na patente:

1.Autenticação Digital (Reivindicação 1.b): Verifica a identidade do usuário que iniciou a operação. Sem autenticação, a operação é rejeitada.
2.Autorização por Papéis e Permissões (RBAC) (Reivindicação 1.c): O sistema consulta os roles e permissions do usuário (users collection) para determinar se ele tem autorização para executar o eventType solicitado.
3.Filtragem por Identificador Organizacional (tenantId) (Reivindicação 1.d, 2): Automaticamente, todas as operações são filtradas pelo tenantId do usuário. Isso garante que um usuário de uma organização nunca acesse ou modifique dados de outra organização.
4.Motor de Regras (Pré-execução) (Reivindicação 1.e, 4): Antes de qualquer mudança de estado, o sistema consulta um motor de regras configurável para verificar se a operação viola quaisquer limites (financeiros, operacionais, técnicos, contratuais ou regulatórios). Ex: Bloquear retirada de estoque se o saldo for insuficiente.
5.Máquina de Estados Formal (Reivindicação 1.f, 5): Para entidades que possuem um ciclo de vida (ex: Contract, MilkCollection), o Orchestrator verifica se a transição de estado solicitada é permitida pela state_machines collection. Transições inválidas são bloqueadas.
6.Captura Obrigatória de Evidência Digital (Reivindicação 1.g, 7): Dependendo da criticidade do eventType, o sistema exige a anexação de Evidência Tipo A (identificador/código) ou Tipo B (mídia, GPS). A ausência ou invalidade da evidência bloqueia a progressão.
7.Registro em Auditoria Imutável (Reivindicação 1.h, 8): Após todas as validações e antes da execução final, o evento é registrado na coleção audit_events com seu hash criptográfico e encadeamento.
8.Execução e Liquidação Condicionada (Reivindicação 1.i, 9): Somente após passar por todas as etapas anteriores, a operação é executada. Se aplicável, o módulo de liquidação condicionada gerencia a custódia lógica (escrow) e a liberação de valores (split) com base em marcos verificáveis.

1.2. Fluxo Simplificado do Event Orchestrator

mermaid
Fonte

2. Auditoria Imutável: O Lastro Criptográfico

A coleção audit_events não é apenas um log; ela é um ledger imutável que utiliza princípios criptográficos para garantir a integridade e a não-repudiação de cada registro. Isso atende diretamente às reivindicações 1.h e 8 da patente.

2.1. Estrutura do Documento audit_events

Conforme definido no schema do banco de dados, cada documento de auditoria contém:
•auditId: ID único do evento.
•tenantId: ID da organização.
•type: Tipo do evento (ex: STOCK_WITHDRAW).
•actor: Quem executou (UID, papel).
•meta: Detalhes específicos do evento.
•ts: Carimbo de tempo.
•evidenceIds: IDs das evidências digitais associadas.
•stateBefore, stateAfter: Estados da entidade antes e depois (opcional).
•prevHash: Hash criptográfico do evento de auditoria anterior para o mesmo tenantId.
•eventHash: Hash criptográfico do conteúdo completo deste evento.

2.2. Geração e Encadeamento de Hashes

O processo de geração e encadeamento de hashes é o que confere a imutabilidade:
1.Cálculo do prevHash: Antes de registrar um novo evento de auditoria, o Orchestrator consulta o último audit_events registrado para o tenantId em questão e recupera seu eventHash. Este será o prevHash do novo evento.
2.Cálculo do eventHash: O eventHash é calculado a partir de uma representação serializada (ex: JSON stringificado) de todos os campos relevantes do documento de auditoria, incluindo o prevHash recém-obtido. Isso cria uma dependência criptográfica.
3.Registro Atômico: O registro do novo evento na coleção audit_events deve ser uma operação atômica (transação) para garantir que o prevHash seja sempre o do último evento válido e que o eventHash seja calculado corretamente.
Pseudocódigo para Geração de Hash:
TypeScript

```
import { createHash } from 'crypto'; // Função para calcular o hash de um objeto function calculateHash(data: any): string { const dataString = JSON.stringify(data); // Serializa o objeto para uma string return createHash('sha256').update(dataString).digest('hex'); } // Exemplo de como um novo evento de auditoria seria criado async function createAuditEvent(eventData: any, tenantId: string) { // 1. Obter o hash do último evento para este tenant const
```

```
lastAuditEvent = await db.collection('audit_events')
.where('tenantId', '==', tenantId) .orderBy('ts', 'desc')
.limit(1) .get(); const prevHash = lastAuditEvent.empty ?
'0'.repeat(64) : lastAuditEvent.docs[0].data().eventHash; // 2.
Preparar os dados do novo evento, incluindo o prevHash const
newAuditEvent = { auditId: makeId('aud'), tenantId: tenantId,
type: eventData.type, actor: eventData.actor, meta:
eventData.meta, ts: Date.now(), evidenceIds:
eventData.evidenceIds || [], stateBefore: eventData.stateBefore
|| null, stateAfter: eventData.stateAfter || null, prevHash:
prevHash, eventHash: '' // Será preenchido após o cálculo }; //
3. Calcular o eventHash com base em todos os dados, incluindo o
prevHash newAuditEvent.eventHash = calculateHash(newAuditEvent);
// 4. Registrar o evento (idealmente dentro de uma transação
Firestore) await
db.collection('audit_events').doc(newAuditEvent.auditId).set(new
AuditEvent); return newAuditEvent; }
```

2.3. Verificação de Integridade

Para validar a integridade da cadeia de auditoria, um serviço de verificação pode re-calcular os hashes de trás para frente. Se qualquer eventHash não corresponder ao prevHash do evento subsequente, ou se o eventHash de um documento não corresponder ao seu próprio conteúdo, significa que houve adulteração.

3. Conformidade com a Patente

O Event Orchestrator e a Auditoria Imutável, quando implementados conforme descrito, garantem a conformidade com as seguintes reivindicações da patente:
•Reivindicação 1 (Sistema e Método): A estrutura geral do sistema e o método de validação são diretamente implementados.
•Reivindicação 4 (Motor de Regras): O Orchestrator invoca o motor de regras para bloquear operações inválidas.
•Reivindicação 5 (Máquina de Estados): As transições de estado são controladas pelo Orchestrator.
•Reivindicação 6 (Registro Automático): Cada evento é registrado com todos os metadados necessários.
•Reivindicação 7 (Evidência Digital): A exigência de evidência é parte integrante da cadeia de validação.
•Reivindicação 8 (Auditoria Imutável): O encadeamento criptográfico de hashes é a essência da Auditoria Imutável.
•Reivindicação 9 (Liquidação Condicionada): A execução da liquidação é o passo final da cadeia, após todas as validações.

4. Próximos Passos

Com o detalhamento do Event Orchestrator e da Auditoria Imutável concluído, o próximo passo lógico seria iniciar a implementação desses componentes no backend

(NestJS), criando os serviços e controladores que darão vida a essa arquitetura. Isso incluiria:
•Criação de um módulo AuditService para gerenciar a coleção audit_events.
•Criação de um EventOrchestratorService que encapsule a cadeia de validação.
•Implementação de um LockService e RulesEngineService básicos.
Estou pronto para começar a codificação desses serviços, se desejar.

Arquitetura do Banco de Dados (Firestore) - Kernel Ciclo+

Este documento detalha a arquitetura inicial do banco de dados Firestore para o **Kernel do sistema Ciclo+**, conforme as reivindicações da patente BR 10 2026 002755 3. O foco está em estabelecer as coleções essenciais que garantem a segurança, a imutabilidade e o isolamento multi-organizacional.

---

1. Princípios de Design do Schema

Para aderir à patente, o design do schema segue os seguintes princípios:

* **`tenantId` Obrigatório:** Cada documento em coleções transacionais deve conter um `tenantId` como atributo obrigatório, garantindo o isolamento de dados entre organizações.
* **Imutabilidade:** Coleções de auditoria são *append-only*, com hashes criptográficos para integridade.
* **Event-Driven:** O sistema é projetado para registrar e processar eventos, que são a base da auditoria e das transições de estado.
* **Normalização Seletiva:** Prioriza a performance e a simplicidade de leitura para dados frequentemente acessados, enquanto mantém a integridade para dados críticos.

---

2. Coleções Essenciais do Kernel

2.1. `tenants` (Organizações)

Esta coleção armazena as informações básicas de cada organização que utiliza o sistema. O `tenantId` é a chave primária para o isolamento de dados.

| Campo | Tipo | Descrição | Requisitos da Patente |

| :--- | :--- | :--- | :--- |

| `tenantId` | `string` | ID único da organização. | **Reivindicação 1.d, 2:** Identificador organizacional obrigatório. |

| `name` | `string` | Nome da organização. | |

| `status` | `string` | Status da organização (e.g., `ACTIVE`, `INACTIVE`). | |

| `createdAt` | `timestamp` | Data de criação do registro. | |

| `updatedAt` | `timestamp` | Última atualização do registro. | |

Exemplo de Documento:

```json
{
  "tenantId": "TENANT_FARM_001",
  "name": "Fazenda Esperança",
  "status": "ACTIVE",
  "createdAt": 1706985600000,
  "updatedAt": 1706985600000
}
```

2.2. `users` (Usuários)

Armazena as informações dos usuários, vinculando-os a uma organização (`tenantId`) e definindo seus papéis e permissões.

| Campo | Tipo | Descrição | Requisitos da Patente |

| :--- | :--- | :--- | :--- |

| `uid` | `string` | ID único do usuário (e.g., Firebase Auth UID). | **Reivindicação 1.b, 3:** Autenticação digital de usuários. |

| `tenantId` | `string` | ID da organização à qual o usuário pertence. | **Reivindicação 1.d, 3:** Filtragem por identificador organizacional. |

| `email` | `string` | E-mail do usuário. | |

| `name` | `string` | Nome completo do usuário. | |

| `roles` | `array<string>` | Papéis do usuário (e.g., `ADMIN`, `MANAGER`, `OPERATOR`). | **Reivindicação 1.c, 3:** Autorização hierárquica por papéis. |

| `permissions` | `array<string>` | Permissões específicas do usuário. | **Reivindicação 1.c, 3:** Autorização hierárquica por permissões. |

| `status` | `string` | Status do usuário (e.g., `ACTIVE`, `INACTIVE`). | |

| `createdAt` | `timestamp` | Data de criação. | |

| `updatedAt` | `timestamp` | Última atualização. | |

Exemplo de Documento:

```json
{
  "uid": "user_abc123",
  "tenantId": "TENANT_FARM_001",
  "email": "joao.silva@fazenda.com",
  "name": "João Silva",
  "roles": ["OPERATOR"],
  "permissions": ["intake.create", "stock.withdraw"],
  "status": "ACTIVE",
  "createdAt": 1706985600000,
  "updatedAt": 1706985600000
}
```

2.3. `audit_events` (Eventos de Auditoria Imutável)

Esta é uma coleção crucial para a patente, registrando cada evento operacional de forma imutável com encadeamento criptográfico de hashes. É uma coleção *append-only*.

| Campo | Tipo | Descrição | Requisitos da Patente |

| :--- | :--- | :--- | :--- |

| `auditId` | `string` | ID único do evento de auditoria. | **Reivindicação 1.h, 8:** Auditoria imutável. |

| `tenantId` | `string` | ID da organização. | **Reivindicação 1.d:** Filtragem organizacional. |

| `type` | `string` | Tipo do evento (e.g., `STOCK_WITHDRAW`, `MILK_COLLECTION_ESCROW_REQUESTED`). | **Reivindicação 6:** Registrar evento operacional. |

| `actor` | `map` | Informações do usuário que executou a ação (`uid`, `role`). | **Reivindicação 6:** Registrar usuário executor. |

| `meta` | `map` | Metadados específicos do evento (e.g., `itemId`, `qtyBase`, `collectionId`, `escrowId`). | **Reivindicação 6:** Registrar entidade afetada. |

| `ts` | `timestamp` | Carimbo de tempo do evento. | **Reivindicação 6:** Registrar marca temporal. |

| `prevHash` | `string` | Hash criptográfico do evento de auditoria anterior do mesmo `tenantId`. | **Reivindicação 1.h, 8:** Encadeamento criptográfico de hashes. |

| `eventHash` | `string` | Hash criptográfico do conteúdo deste evento. | **Reivindicação 1.h, 8:** Hash do evento. |

| `evidenceIds` | `array<string>` | IDs das evidências digitais associadas a este evento. | **Reivindicação 6, 8:** Evidências associadas. |

| `stateBefore` | `map` | Estado da entidade antes do evento (opcional). | **Reivindicação 6:** Estado anterior. |

| `stateAfter` | `map` | Estado da entidade após o evento (opcional). | **Reivindicação 6:** Estado posterior. |

Exemplo de Documento:

```json
{
  "auditId": "aud_001",
  "tenantId": "TENANT_FARM_001",
  "type": "STOCK_WITHDRAW",
  "actor": { "uid": "user_abc123", "role": "OPERATOR" },
  "meta": { "itemId": "item_salt_01", "qtyBase": 75, "unitBase": "KG" },
  "ts": 1706985600000,
  "prevHash": "a1b2c3d4e5f6...",
```

```
  "eventHash": "f6e5d4c3b2a1...",
  "evidenceIds": ["evg_001"],
  "stateBefore": { "stockBalance": 100 },
  "stateAfter": { "stockBalance": 25 }
}
```

2.4. `state_machines` (Máquinas de Estado)

Define as máquinas de estado formais para diferentes entidades (e.g., `CONTRACT`, `MILK_COLLECTION`), incluindo as transições permitidas e as condições para cada transição.

Campo	Tipo	Descrição	Requisitos da Patente
`machineId`	`string`	ID único da máquina de estados (e.g., `CONTRACT_MACHINE`).	**Reivindicação 1.f, 5:** Máquina de estados formal.
`tenantId`	`string`	ID da organização (se a máquina de estados for específica do tenant).	
`entityType`	`string`	Tipo de entidade que esta máquina de estados governa (e.g., `Contract`, `MilkCollection`).	
`initialState`	`string`	Estado inicial da entidade.	
`transitions`	`array<map>`	Lista de transições permitidas (`fromState`, `toState`, `eventTrigger`, `conditions`).	**Reivindicação 1.f, 5:** Define transições permitidas.

Exemplo de Documento:

```json
{
  "machineId": "MILK_COLLECTION_MACHINE",
  "tenantId": "GLOBAL", // Ou específico do tenant
  "entityType": "MilkCollection",
  "initialState": "CREATED",
  "transitions": [
```

{ "fromState": "CREATED", "toState": "READY_FOR_PAYMENT", "eventTrigger": "LAB_REPORT_ATTACHED", "conditions": { "labApproved": true } },

{ "fromState": "READY_FOR_PAYMENT", "toState": "PAYMENT_REQUESTED", "eventTrigger": "ESCROW_REQUESTED", "conditions": { "priceSet": true } },

{ "fromState": "PAYMENT_REQUESTED", "toState": "PAID", "eventTrigger": "ESCROW_PAID", "conditions": { "pspConfirmed": true } }

]

}

```

### 2.5. `locks` (Bloqueios)

Gerencia os bloqueios ativos que podem impedir operações críticas, conforme a reivindicação 8 da patente (Lock Service).

| Campo | Tipo | Descrição | Requisitos da Patente |

| :--- | :--- | :--- | :--- |

| `lockId` | `string` | ID único do bloqueio. | **Reivindicação 1.h:** Lock Service. |

| `tenantId` | `string` | ID da organização afetada pelo bloqueio. | **Reivindicação 1.d:** Filtragem organizacional. |

| `scope` | `map` | Escopo do bloqueio (e.g., `{ type: 'TENANT', tenantId: '...' }`, `{ type: 'PROPERTY', propertyId: '...' }`). | |

| `reason` | `map` | Motivo do bloqueio (`code`, `title`, `details`). | |

| `effects` | `map` | Efeitos do bloqueio (e.g., `blockNewContracts: true`, `blockSale: true`). | **Reivindicação 4:** Motor de regras bloqueia operações. |

| `status` | `string` | Status do bloqueio (e.g., `ACTIVE`, `RESOLVED`). | |

| `createdAt` | `timestamp` | Data de criação do bloqueio. | |

| `createdBy` | `map` | Usuário ou sistema que criou o bloqueio. | |

| `expiresAt` | `timestamp` | Data de expiração do bloqueio (opcional). | |

**Exemplo de Documento:**

```json

{

"lockId": "lock_non_payment_001",

```
  "tenantId": "TENANT_FARM_001",
  "scope": { "type": "TENANT", "tenantId": "TENANT_INDUSTRY_001" },
  "reason": { "code": "NON_PAYMENT", "title": "Inadimplência", "details": "Escrow leite não pago: esc_milk_001" },
  "effects": { "blockNewContracts": true, "blockSale": true },
  "status": "ACTIVE",
  "createdAt": 1706985600000,
  "createdBy": { "uid": "system", "role": "SYSTEM" },
  "expiresAt": null
}
```

---

3. Próximos Passos

Com este schema inicial do Kernel, o próximo passo seria detalhar a implementação do **Event Orchestrator** e do **Serviço de Auditoria Imutável**, incluindo a lógica para o cálculo e encadeamento dos hashes criptográficos. Isso solidificará a base do sistema e permitirá que os módulos operacionais sejam construídos sobre uma fundação segura e patenteada.

Estou pronto para prosseguir com o detalhamento do Event Orchestrator e da Auditoria Imutável, se desejar.

Roadmap Tecnológico do Sistema Ciclo+

Este roadmap detalha as fases de desenvolvimento do sistema Ciclo+, priorizando a implementação do **Núcleo Patenteável** conforme descrito no pedido de patente BR 10 2026 002755 3. O objetivo é construir uma base robusta e em conformidade com as reivindicações de propriedade intelectual, garantindo a inovação e a utilidade do sistema.

---

1. Visão Geral do Roadmap

O desenvolvimento será dividido em fases estratégicas, começando pela fundação do sistema (o Kernel) e progredindo para a integração dos módulos operacionais. Cada fase é projetada para entregar valor incremental e validar os pilares da patente.

| Fase | Objetivo Principal | Entregáveis Chave | Prioridade | Duração Estimada |

| :--- | :--- | :--- | :--- | :--- |

| **Fase 0: Setup e Arquitetura Base** | Configurar o ambiente de desenvolvimento e definir a arquitetura inicial do Kernel. | Repositório de código, ambiente de nuvem configurado, modelo de dados inicial do Kernel. | Essencial | 2 semanas |

| **Fase 1: Núcleo de Confiança (Kernel MVP)** | Implementar os componentes essenciais do Kernel que garantem a patenteabilidade e a segurança. | Identity Service, RBAC/Permissions, Tenant Isolation, Event Orchestrator (esqueleto), Auditoria Imutável (com hashes). | Crítica | 6 semanas |

| **Fase 2: Evidência e Liquidação Condicionada** | Desenvolver os módulos de Evidência Digital e Liquidação Condicionada, integrando-os ao Event Orchestrator. | Módulo de Evidência (Tipo A/B), Máquina de Estados, Módulo de Escrow/Split (funcionalidade básica). | Alta | 8 semanas |

| **Fase 3: Módulo Operacional Piloto** | Implementar um módulo operacional (ex: Estoque ou Leite) integrado ao Kernel, validando o fluxo ponta a ponta. | Módulo de Estoque (Intake por Foto/Voz), Integração com Kernel, Testes de Fluxo. | Média | 10 semanas |

| **Fase 4: Refinamento e Expansão** | Otimização, testes de segurança, documentação e planejamento para módulos adicionais. | Testes de performance, hardening de segurança, documentação técnica, roadmap para próximos módulos. | Contínua | Variável |

---

2. Detalhamento das Fases

Fase 0: Setup e Arquitetura Base

Esta fase inicial foca na preparação do terreno para o desenvolvimento. É crucial estabelecer as ferramentas e a estrutura básica que suportarão todo o projeto.

*   **Tecnologias Selecionadas:**

    *   **Backend:** Node.js com NestJS (TypeScript).

    *   **Banco de Dados:** Google Firestore (para o Kernel e dados transacionais).

    *   **Controle de Versão:** Git (GitHub/GitLab).

    *   **Infraestrutura:** Google Cloud Platform (GCP) ou similar para Firestore, Cloud Functions/Run para backend.

*   **Atividades:**

    *   Criação de repositórios de código (backend, mobile).

    *   Configuração de ambiente de desenvolvimento local e de CI/CD (integração contínua/entrega contínua).

    *   Definição inicial da estrutura de pastas e módulos no backend.

    *   Configuração do Firestore e regras de segurança básicas.

Fase 1: Núcleo de Confiança (Kernel MVP)

Esta é a fase mais crítica, onde os pilares da patente são construídos. O foco é na imutabilidade, segurança e orquestração de eventos.

*   **Componentes a Desenvolver:**

    *   **Identity Service:** Geração e validação de IDs únicos para `tenantId`, `entityId` (produtos, ativos, eventos).

    *   **Auth, RBAC e Permissions:** Implementação do sistema de autenticação, autorização por papéis e gerenciamento de permissões, conforme a reivindicação 3 da patente.

    *   **Tenant Isolation:** Garantir que todas as operações sejam filtradas por `tenantId` (reivindicação 2).

    *   **Event Orchestrator (Esqueleto):** Um dispatcher central que recebe "intenções" e as roteia através da cadeia de validação (RBAC -> Tenant -> Locks -> Rules -> States -> Evidence -> Audit -> Settlement).

    *   **Auditoria Imutável:** Implementação do `audit_events` com encadeamento criptográfico de hashes (reivindicação 8), garantindo a não-repudiação e a integridade dos registros.

*   **Entregáveis:** APIs para autenticação/autorização, serviços de identidade, estrutura do Event Orchestrator, serviço de auditoria com hashes.

Fase 2: Evidência e Liquidação Condicionada

Nesta fase, o sistema ganha a capacidade de exigir provas e gerenciar fluxos financeiros complexos de forma automatizada e segura.

*   **Componentes a Desenvolver:**

    *   **Módulo de Evidência Digital:** Implementação da lógica para Evidência Tipo A (identificador/código) e Tipo B (mídia, GPS) (reivindicação 7). Integração com serviços de armazenamento de mídia (ex: Firebase Storage).

    *   **Máquina de Estados Formal:** Desenvolvimento da lógica de transição de estados para entidades críticas (ex: contratos, coletas de leite), impedindo transições inválidas (reivindicação 5).

    *   **Motor de Regras:** Implementação de um motor de regras configurável para validação pré-execução (reivindicação 4), bloqueando operações que violem limites.

    *   **Módulo de Liquidação Condicionada (Escrow/Split):** Integração inicial com o PSP (Asaas) para criar cobranças, gerenciar custódia lógica (escrow) e executar splits de pagamento condicionados a marcos verificáveis (reivindicação 9).

*   **Entregáveis:** APIs para gestão de evidências, serviços de máquina de estados e motor de regras, integração básica com Asaas para escrow/split.

Fase 3: Módulo Operacional Piloto

Com o Kernel funcionando, podemos desenvolver um módulo de negócio completo para validar a arquitetura e o fluxo ponta a ponta.

*   **Escolha do Módulo:** Sugere-se o módulo de **Estoque** com **Intake por Foto/Voz**, pois ele valida a captura de evidências em campo e a integração com o Event Orchestrator.

*   **Componentes a Desenvolver:**

    *   **Módulo de Estoque:** Cadastro de itens, gestão de saldo por local, movimentações (entrada/saída).

    *   **Módulo de Intake (Foto/Voz):** Desenvolvimento do frontend mobile (React Native/Expo) para captura de fotos/áudios, OCR/ASR (on-device ou cloud), criação de `intakes` e `draft_movements`.

    *   **Integração:** Conectar o fluxo de Intake ao Event Orchestrator para que as movimentações de estoque passem por todas as validações do Kernel (evidência, regras, estados, auditoria).

*   **Entregáveis:** Aplicação mobile funcional para Intake de Estoque, APIs de estoque, testes de integração.

Fase 4: Refinamento e Expansão

Esta fase é contínua e foca na melhoria, segurança e crescimento do sistema.

*   **Atividades:**
    *   **Testes Abrangentes:** Testes de unidade, integração, ponta a ponta e de carga.
    *   **Hardening de Segurança:** Auditorias de segurança, proteção contra vulnerabilidades.
    *   **Documentação:** Documentação técnica detalhada das APIs, arquitetura e processos.
    *   **Monitoramento e Alertas:** Implementação de ferramentas de monitoramento de performance e alertas de erros.
    *   **Expansão:** Planejamento e desenvolvimento de outros módulos (Leite, Marketplace, Contratos) seguindo a mesma arquitetura.

---

3. Próximos Passos

Com este roadmap, podemos iniciar o desenvolvimento. O próximo passo seria detalhar a **Arquitetura do Banco de Dados** para os componentes do Kernel, garantindo que as coleções e seus atributos reflitam as reivindicações da patente, especialmente o `tenantId` e os campos de auditoria.

Estou pronto para prosseguir com o detalhamento da arquitetura do banco de dados para o Kernel, se desejar.))

Análise Técnica: Pedido de Patente Ciclo+ (BR 10 2026 002755 3)

Este documento analisa o pedido de patente de invenção depositado por **Mateus Sousa Nascimento** em 04/02/2026, intitulado *"Sistema e Método Computacional Multi-Organizacional com Evidência Digital Obrigatória, Auditoria Imutável e Liquidação Condicionada por Marcos Verificáveis"*.

---

1. Identificação do Documento

*   **Número do Processo:** BR 10 2026 002755 3

*   **Depositante:** Mateus Sousa Nascimento (Produtor Agropecuário)

*   **Data do Depósito:** 04/02/2026

*   **Natureza:** Patente de Invenção (PI)

---

2. Núcleo Tecnológico Patenteável

O documento define um "Núcleo Patenteável" que deve permanecer inalterado independentemente do módulo setorial aplicado. Os componentes essenciais são:

Componente	Descrição Técnica Reivindicada
Identificador Organizacional	Atributo obrigatório (`orgId`/`tenantId`) em todas as entidades de dados para filtragem nativa e isolamento total entre organizações.
Motor de Regras	Validação pré-execução que bloqueia operações que violem limites financeiros, técnicos ou regulatórios.
Máquina de Estados	Define transições permitidas e condiciona o avanço à validade das evidências.
Evidência Digital	Classificada em **Tipo A** (identificador + timestamp) para rotina e **Tipo B** (mídia + GPS + timestamp) para eventos críticos.
Auditoria Imutável	Registro *append-only* com encadeamento criptográfico de hashes (evento + evidência + evento anterior).
Liquidação Condicionada	Custódia lógica (*escrow*) com liberação de valores (*split*) vinculada a marcos operacionais verificáveis.

---

3. Reivindicações de Método e Sistema

O quadro reivindicatório protege tanto o **sistema** (infraestrutura e módulos) quanto o **método** (sequência de etapas de validação). Pontos de destaque:

*   **Filtragem Nativa:** O isolamento de dados não é apenas lógico, mas um requisito técnico de todas as operações de leitura/escrita.

*   **Proporcionalidade ao Risco:** A exigência de evidência varia conforme a criticidade, otimizando a performance em campo (SOP - Procedimento Operacional Padrão).

*   **Interface de Fiscalização:** Acesso *read-only* para auditores, garantindo que registros críticos nunca sejam modificados.

---

4. Relação com o Planejamento do Sistema

Este documento de patente serve como a **especificação técnica mestre** para o desenvolvimento do Ciclo+. Ele impõe restrições arquiteturais que devem ser seguidas rigorosamente:

1. **Imutabilidade:** A auditoria não pode ser um simples log; deve haver encadeamento de hashes.

2. **Condicionalidade:** O código de liquidação financeira deve estar tecnicamente acoplado à validação de evidências da máquina de estados.

3. **Isolamento:** O `tenantId` deve ser um cidadão de primeira classe em todas as tabelas/coleções do banco de dados.

---

5. Conclusão e Próximos Passos

O depósito da patente protege a inovação do Ciclo+ no INPI, garantindo a exclusividade sobre o método de "Liquidação Condicionada por Marcos Verificáveis" e o uso de "Evidência Digital Obrigatória" como trava técnica.

Recomendação para o Desenvolvimento:

*   Assegurar que a implementação do **Event Orchestrator** siga exatamente a sequência de validação descrita no Relatório Descritivo (Autenticação -> RBAC -> Filtro Org -> Regras -> Estados -> Evidência -> Auditoria -> Liquidação).

*   Implementar o encadeamento de hashes na coleção de auditoria desde o primeiro dia para cumprir a reivindicação 8.

Roadmap Tecnológico do Sistema Ciclo+

Este roadmap detalha as fases de desenvolvimento do sistema Ciclo+, priorizando a implementação do **Núcleo Patenteável** conforme descrito no pedido de patente BR 10 2026 002755 3. O objetivo é construir uma base robusta e em conformidade com as reivindicações de propriedade intelectual, garantindo a inovação e a utilidade do sistema.

---

1. Visão Geral do Roadmap

O desenvolvimento será dividido em fases estratégicas, começando pela fundação do sistema (o Kernel) e progredindo para a integração dos módulos operacionais. Cada fase é projetada para entregar valor incremental e validar os pilares da patente.

| Fase | Objetivo Principal | Entregáveis Chave | Prioridade | Duração Estimada |

| :--- | :--- | :--- | :--- | :--- |

| **Fase 0: Setup e Arquitetura Base** | Configurar o ambiente de desenvolvimento e definir a arquitetura inicial do Kernel. | Repositório de código, ambiente de nuvem configurado, modelo de dados inicial do Kernel. | Essencial | 2 semanas |

| **Fase 1: Núcleo de Confiança (Kernel MVP)** | Implementar os componentes essenciais do Kernel que garantem a patenteabilidade e a segurança. | Identity Service, RBAC/Permissions, Tenant Isolation, Event Orchestrator (esqueleto), Auditoria Imutável (com hashes). | Crítica | 6 semanas |

| **Fase 2: Evidência e Liquidação Condicionada** | Desenvolver os módulos de Evidência Digital e Liquidação Condicionada, integrando-os ao Event Orchestrator. | Módulo de Evidência (Tipo A/B), Máquina de Estados, Módulo de Escrow/Split (funcionalidade básica). | Alta | 8 semanas |

| **Fase 3: Módulo Operacional Piloto** | Implementar um módulo operacional (ex: Estoque ou Leite) integrado ao Kernel, validando o fluxo ponta a ponta. | Módulo de Estoque (Intake por Foto/Voz), Integração com Kernel, Testes de Fluxo. | Média | 10 semanas |

| **Fase 4: Refinamento e Expansão** | Otimização, testes de segurança, documentação e planejamento para módulos adicionais. | Testes de performance, hardening de segurança, documentação técnica, roadmap para próximos módulos. | Contínua | Variável |

---

2. Detalhamento das Fases

Fase 0: Setup e Arquitetura Base

Esta fase inicial foca na preparação do terreno para o desenvolvimento. É crucial estabelecer as ferramentas e a estrutura básica que suportarão todo o projeto.

* **Tecnologias Selecionadas:**
  * **Backend:** Node.js com NestJS (TypeScript).
  * **Banco de Dados:** Google Firestore (para o Kernel e dados transacionais).
  * **Controle de Versão:** Git (GitHub/GitLab).
  * **Infraestrutura:** Google Cloud Platform (GCP) ou similar para Firestore, Cloud Functions/Run para backend.
* **Atividades:**
  * Criação de repositórios de código (backend, mobile).
  * Configuração de ambiente de desenvolvimento local e de CI/CD (integração contínua/entrega contínua).
  * Definição inicial da estrutura de pastas e módulos no backend.
  * Configuração do Firestore e regras de segurança básicas.

Fase 1: Núcleo de Confiança (Kernel MVP)

Esta é a fase mais crítica, onde os pilares da patente são construídos. O foco é na imutabilidade, segurança e orquestração de eventos.

* **Componentes a Desenvolver:**
  * **Identity Service:** Geração e validação de IDs únicos para `tenantId`, `entityId` (produtos, ativos, eventos).
  * **Auth, RBAC e Permissions:** Implementação do sistema de autenticação, autorização por papéis e gerenciamento de permissões, conforme a reivindicação 3 da patente.
  * **Tenant Isolation:** Garantir que todas as operações sejam filtradas por `tenantId` (reivindicação 2).
  * **Event Orchestrator (Esqueleto):** Um dispatcher central que recebe "intenções" e as roteia através da cadeia de validação (RBAC -> Tenant -> Locks -> Rules -> States -> Evidence -> Audit -> Settlement).

    *   **Auditoria Imutável:** Implementação do `audit_events` com encadeamento criptográfico de hashes (reivindicação 8), garantindo a não-repudiação e a integridade dos registros.

*   **Entregáveis:** APIs para autenticação/autorização, serviços de identidade, estrutura do Event Orchestrator, serviço de auditoria com hashes.

Fase 2: Evidência e Liquidação Condicionada

Nesta fase, o sistema ganha a capacidade de exigir provas e gerenciar fluxos financeiros complexos de forma automatizada e segura.

*   **Componentes a Desenvolver:**

    *   **Módulo de Evidência Digital:** Implementação da lógica para Evidência Tipo A (identificador/código) e Tipo B (mídia, GPS) (reivindicação 7). Integração com serviços de armazenamento de mídia (ex: Firebase Storage).

    *   **Máquina de Estados Formal:** Desenvolvimento da lógica de transição de estados para entidades críticas (ex: contratos, coletas de leite), impedindo transições inválidas (reivindicação 5).

    *   **Motor de Regras:** Implementação de um motor de regras configurável para validação pré-execução (reivindicação 4), bloqueando operações que violem limites.

    *   **Módulo de Liquidação Condicionada (Escrow/Split):** Integração inicial com o PSP (Asaas) para criar cobranças, gerenciar custódia lógica (escrow) e executar splits de pagamento condicionados a marcos verificáveis (reivindicação 9).

*   **Entregáveis:** APIs para gestão de evidências, serviços de máquina de estados e motor de regras, integração básica com Asaas para escrow/split.

Fase 3: Módulo Operacional Piloto

Com o Kernel funcionando, podemos desenvolver um módulo de negócio completo para validar a arquitetura e o fluxo ponta a ponta.

*   **Escolha do Módulo:** Sugere-se o módulo de **Estoque** com **Intake por Foto/Voz**, pois ele valida a captura de evidências em campo e a integração com o Event Orchestrator.

*   **Componentes a Desenvolver:**

    *   **Módulo de Estoque:** Cadastro de itens, gestão de saldo por local, movimentações (entrada/saída).

*   **Módulo de Intake (Foto/Voz):** Desenvolvimento do frontend mobile (React Native/Expo) para captura de fotos/áudios, OCR/ASR (on-device ou cloud), criação de `intakes` e `draft_movements`.

*   **Integração:** Conectar o fluxo de Intake ao Event Orchestrator para que as movimentações de estoque passem por todas as validações do Kernel (evidência, regras, estados, auditoria).

*   **Entregáveis:** Aplicação mobile funcional para Intake de Estoque, APIs de estoque, testes de integração.

Fase 4: Refinamento e Expansão

Esta fase é contínua e foca na melhoria, segurança e crescimento do sistema.

*   **Atividades:**

*   **Testes Abrangentes:** Testes de unidade, integração, ponta a ponta e de carga.

*   **Hardening de Segurança:** Auditorias de segurança, proteção contra vulnerabilidades.

*   **Documentação:** Documentação técnica detalhada das APIs, arquitetura e processos.

*   **Monitoramento e Alertas:** Implementação de ferramentas de monitoramento de performance e alertas de erros.

*   **Expansão:** Planejamento e desenvolvimento de outros módulos (Leite, Marketplace, Contratos) seguindo a mesma arquitetura.

---

3. Próximos Passos

Com este roadmap, podemos iniciar o desenvolvimento. O próximo passo seria detalhar a **Arquitetura do Banco de Dados** para os componentes do Kernel, garantindo que as coleções e seus atributos reflitam as reivindicações da patente, especialmente o `tenantId` e os campos de auditoria.

Estou pronto para prosseguir com o detalhamento da arquitetura do banco de dados para o Kernel, se desejar.))