Código do Desenho (Mermaid) — FIG. 1

```
flowchart LR
subgraph CH["Canais / Interfaces"]
  CH1["Web/Mobile (UI)"]
  CH2["API / Integrações"]
  CH3["Voz / Assistente"]
  CH4["Leitor/Camera/GPS"]
end
subgraph CORE["Núcleo Patenteável"]
  A["Autenticação"]
  B["RBAC / Permissões"]
  C["Filtro orgId/tenantId"]
  D["Motor de Regras (pré-execução)"]
  E["Máquina de Estados"]
  F["Evidência Obrigatória (Tipo A/B)"]
  G["Auditoria Imutável (log encadeado)"]
  H["Liquidação Condicionada (escrow/tranches/split)"]
  I["Auditoria/Fiscalização (read-only)"]
end
subgraph MOD["Módulos Habilitáveis por Setor"]
  M1["Inventory / Estoque por Local"]
  M2["Projects/RTF / Limites"]
  M3["Custody/Consignment"]
  M4["Intermediation/Marketplace"]
  M5["Service Orders / Etapas"]
  M6["Compliance/Gov"]
  M7["Analytics/KPIs"]
end
CH1-->A
CH2-->A
CH3-->A
CH4-->A
A-->B-->C-->D-->E
E--"válido"-->F-->G-->H-->I
E--"inválido"-->G
G-->M1
G-->M2
G-->M3
G-->M4
G-->M5
G-->M6
G-->M7
```

Perfeito — com base no seu **CICLO_+.rtf** e amarrando ao **núcleo patenteável** (eventos + evidência Tipo A/B + auditoria imutável + liquidação condicionada/escrow), segue a **Especificação v1** em 3 blocos: **(1) dicionário de dados**, **(2) tabela de operações**, **(3) fluxos de liquidação por marcos**.

---

1) Dicionário de dados v1 (entidades + campos mínimos)

Regra estrutural: **todas as entidades persistidas têm `orgId/tenantId`** e toda operação gera um **Evento** com evidência e auditoria.

Identidade e multi-organização

- **Organization (Tenant)**
  - `orgId`, `nome`, `tipoOrg` (loja/cooperativa/indústria/fazenda/serviço), `status`
- **User**
  - `userId`, `orgId`, `nome`, `email/telefone`, `status`, `lastLoginAt`
- **Role / Permission (RBAC)**
  - `roleId`, `orgId`, `nomeRole` (ex.: PRODUTOR, TECNICO, FORNECEDOR, INVESTIDOR, CAIXA, AUDITOR), `permissoes[]`

Cadastros do Agro (bovinos incluído)

- **Producer (Produtor)**
  - `producerId`, `orgId`, `dadosPessoais`, `tipo` (PF/PR), `status`, `propriedades[]`
- **Property (Propriedade)**
  - `propertyId`, `orgId`, `producerId`, `nome`, `geo` (lat/lon/área), `status`, `relatoriosTecnicos[]`
- **Animal (Bovino)**
  - `animalId`, `orgId`, `propertyId`, `identificacao` (brinco/RFID/QR), `categoria` (cria/recria/engorda/matriz), `statusSanitario`, `status`
- **Asset (Ativo genérico)**
  - `assetId`, `orgId`, `tipo` (ANIMAL/INSUMO/IMOVEL/OUTRO), `referenciaId`, `status`

Projetos, produção e limites

- **Project (Projeto)**
  - `projectId`, `orgId`, `producerId`, `nome`, `volumeProducao`, `prazoVenda`, `precoAlvo`, `valorInvestimento`, `status` (RASCUNHO/ATIVO/EM_ANDAMENTO/ENCERRADO)
- **CreditLimit (Limite de crédito do projeto)**
  - `limitId`, `orgId`, `projectId`, `limiteVigente`, `limiteUtilizado`, `status`

Compras, estoque, farmácia e lotes

- **Product (Produto/insumo)**
  - `productId`, `orgId`, `nome`, `custo`, `fornecedorId`, `status`
- **InventoryLocation (Local/filial)**
  - `locationId`, `orgId`, `nome`, `cidadeUF`, `status`
- **InventoryBalance (Saldo estoque)**
  - `orgId`, `locationId`, `productId`, `saldoAtual`, `saldoSeguranca`
- **PurchaseRequest (Solicitação de compra)**

    - `purchaseId`, `orgId`, `projectId`, `itens[]`, `status` (SOLICITADO/APROVADO/FATURADO/RECEBIDO/CANCELADO)
- **StockEntry (Entrada de estoque / NF)**
    - `entryId`, `orgId`, `locationId`, `nfNumero`, `itens[]`, `status`, `receivedAt` (regra: "somente com NF")
- **LossEvent (Baixa por perda / descarte)**
    - `lossId`, `orgId`, `locationId`, `productId`, `qtd`, `motivo`, `evidenceRef`, `status` (exige foto; pode exigir laudo)
- **BatchLot (Lote)**
    - `lotId`, `orgId`, `categoria`, `tipo` (COMPRA/VENDA/PRODUCAO), `ativos[]` (inclui animais), `insumosUtilizados[]`, `custoTotal`, `periodoVenda`, `status`
- **PharmacyDispense (Fornecimento/farmácia)**
    - `dispenseId`, `orgId`, `lotId`, `productId`, `qtd`, `datetime`, `operadorId`

Vendas, contratos e marketplace

- **Offer/Listing (Anúncio/oferta)**
    - `offerId`, `orgId`, `tipo` (B2B/B2C/LEILAO), `itens[]` (animais/insumos/imóveis), `preco`, `status`
- **Contract (Contrato)**
    - `contractId`, `orgId`, `partes` (produtor/fornecedor/indústria/investidor), `objeto` (lote/serviço/produção), `valorTotal`, `aPagar`, `aReceber`, `prazo`, `status` (VIGENTE/RENOVAR/SOLICITADO/FECHADO)

Logística e serviços

- **LogisticsRequest (Solicitação de coleta)**
    - `logId`, `orgId`, `tipo` (coleta/entrega), `origem`, `destino`, `status`, `localizacaoAtual`
- **ServiceRequest (Serviços)**
    - `serviceId`, `orgId`, `tipoServico`, `solicitanteId`, `status`, `evidenceRef`

Financeiro: escrow, subcontas, recebíveis, garantias

- **EscrowAccount (Conta "scrow/escrow")**
    - `escrowId`, `orgId`, `saldo`, `regrasLiberacao[]`
- **SubAccount (Subcontas)**
    - `subAccountId`, `orgId`, `tipo` (CLIENTE/PROJETO/FORNECEDOR/INVESTIDOR), `refId`, `saldo`
- **Receivable (Recebível)**
    - `recvId`, `orgId`, `origem` (boleto/cartão/pix/contrato), `valor`, `vencimento`, `status` (A_RECEBER/RECEBIDO/EM_ATRASO)
- **Payment (Pagamento)**
    - `payId`, `orgId`, `destino` (fornecedor/dividendos/serviço), `valor`, `metodo` (PIX/BOLETO/CARTAO/TRANSF), `status` (SOLICITADO/AUTORIZADO/EXECUTADO)

- **Guarantee (Garantia)**
  - `guaranteeId`, `orgId`, `contractId`, `tipo` (hipoteca/outra), `valor`, `status`

Núcleo patenteável: eventos, evidências e auditoria

- **OperationalEvent**
  - `eventId`, `orgId`, `actorId`, `entityType`, `entityId`, `action`, `stateFrom`, `stateTo`, `result`, `createdAt`, `idempotencyKey`
- **Evidence**
  - `evidenceId`, `orgId`, `type` (A/B), `timestamp`, `mediaRef?`, `geo?`, `geofenceOk?`, `deviceAttestation?`, `laudoRef?`
  - Regra: Tipo A para rotineiro; Tipo B para crítico; pode exigir laudo.
- **AuditLog (append-only, hash encadeado)**
  - `auditId`, `orgId`, `eventId`, `eventHash`, `evidenceHash`, `prevHash`, `timestamp`

---

2) Tabela de operações v1 (o que construir primeiro)

Cada operação abaixo deve passar pela cadeia: autenticação → RBAC → filtro orgId → motor de regras → máquina de estados → evidência → auditoria → (se aplicável) liquidação.

Operação (UI/API)	Perfil	Entidade	Regra de bloqueio (motor de regras)	Evidência	Efeito em estados/financeiro
Montar Projeto / Novo Projeto	Produtor/Técnico	Project	exigir campos (volume, prazo, preço alvo, investimento)	Tipo A	cria Project=RASCUNHO→ ATIVO
Solicitar Compra	Produtor	PurchaseRequest	**somente dentro do limite do projeto**	Tipo A	Purchase SOLICITADO; reserva limite
Entrada Estoque	Estoquista/Loja	StockEntry	**somente com NF**	Tipo A (NF/ID + timestamp)	atualiza saldo estoque

Operação (UI/API)	Perfil	Entidade	Regra de bloqueio (motor de regras)	Evidência	Efeito em estados/financeiro
Baixa por Perda (descarte)	Estoquista/Loja	LossEvent	exigir motivo + evidência	**Tipo B (foto)**	reduz saldo; audita motivo
Cadastrar Perda (crítica)	Gestor	LossEvent	exigir laudo quando aplicável	**Tipo B + laudo**	reduz saldo; marca crítico
Farmácia: Controle fornecimento	Técnico/Operador	PharmacyDispense	impedir fornecer sem saldo	Tipo A	baixa por lote e registra data/hora
Cadastro de Animais	Técnico	Animal	validar identificação única	Tipo A (leitor)	cria animal + vínculo propriedade
Escalar venda de Lote	Produtor	BatchLot	exigir custos/insumos até a data	Tipo A	define período venda + preço de contrato
Criar anúncios / Ofertar produtos	Produtor/Loja	Offer	validar estoque/lote disponível	Tipo A	Oferta ATIVA
Solicitação de coleta (logística)	Produtor/Loja	LogisticsRequest	validar origem/destino	**Tipo B (GPS/geofence)**	Logística SOLICITADO→EM_COLETA
Depósito de contrato	Financeiro/Jurídico	Contract/Escrow	exigir contrato válido	Tipo A	aloca valor em escrow
Autorizar liberação de pagamento	Financeiro	Payment	liberar **somente após marco verificado**	Tipo A/B conforme marco	dispara tranche/split
Liquidação de	Financeiro	Receivable	conciliar origem do recebível	Tipo A	RECEBIDO; alimenta subconta

Operação (UI/API)	Perfil	Entidade	Regra de bloqueio (motor de regras)	Evidência	Efeito em estados/financeiro
recebimentos					
Incluir garantia (hipoteca)	Jurídico/Financeiro	Guarantee	exigir documentação	Tipo B (mídia + geoloc se aplicável)	vincula garantia ao contrato
Solicitar retirada de capital	Investidor	SubAccount	saldo disponível	Tipo A	cria solicitação de saque

---

3) Fluxos de liquidação v1 (escrow/tranches/split por marcos verificáveis)

Regra central: valores ficam em **escrow** e só são liberados por **tranches e/ou split** depois de marcos operacionais verificados e auditados.

Fluxo A — Compra/fornecimento (pagamento a fornecedor)

Base no RTF: "pagamento de fornecedores", "autorizar liberação de pagamento", "depósito de contrato", "liquidar operação".
Marcos sugeridos (mínimo viável):

1. **Contrato depositado** → cria trava financeira em escrow (marco 1).
2. **NF anexada e entrada de estoque confirmada** ("somente com NF") → libera **Tranche 1**.
3. Se houver logística/entrega: **coleta/entrega confirmada com GPS** → libera **Tranche 2**.
4. Auditoria registra tudo (append-only).

Fluxo B — Venda de lote (bovinos) com split automático

Base no RTF: "lotes", "escalar venda", "valor pelo preço negociado em contrato", "leilão virtual".
Marcos sugeridos:

1. **Contrato fechado** (preço/lote/período) → escrow recebe o pagamento do comprador.
2. **Saída/embarque confirmados** (Tipo B com GPS/geofence + mídia).
3. **Entrega/recebimento confirmados** (Tipo B; opcional laudo).
4. **Split automático** para: produtor + comissão/plataforma + prestador de serviço/logística (se houver). (split condicionado aos marcos auditados).

Fluxo C — Investimento no projeto e retirada

Base no RTF: “aplicar recurso”, “rentabilidade”, “solicitar retirada de capital”, “fundos de investimentos”.
Marcos sugeridos:

1. Aporte entra em subconta do investidor/projeto (escrow/subconta).
2. Liberação do capital para execução do projeto por tranches (ex.: compra planejada validada + NF + recebimento).
3. Retirada só após: apuração de resultados + disponibilidade em subconta + auditoria (read-only).

Fluxo D — Antecipação de recebíveis (crédito)

Base no RTF: “antecipar recebíveis”, “crédito solicitado”, “em andamento”.
Marcos sugeridos:

1. **Solicitar crédito com projeto** (valida limite, margem/relatório técnico).
2. **Garantia vinculada** (quando aplicável: hipoteca/outro).
3. Liquidação automática conforme recebíveis entram (conciliados) e ficam “em dias / em atraso”.

Saída pronta para build (MVP recomendado)

Se você quiser transformar isso em desenvolvimento rápido, o **MVP bovinos** que casa melhor com seu RTF é:

1. **Cadastro de propriedades + cadastro de animais**
2. **Lotes (ativos/animais) + farmácia (consumo por lote)**
3. **Venda por lote + logística + escrow por marcos**

Se você quiser, no próximo passo eu já te devolvo **(a)** um conjunto de **estados e transições** (máquina de estados) para **Compra**, **Estoque**, **Lote bovinos**, **Venda**, **Logística** e **Liquidação**, e **(b)** uma lista de **regras do motor** (“bloqueia se…”) para cada operação — tudo seguindo o núcleo do pedido (Tipo A/B + auditoria + idempotência/offline).

A seguir está um **modelo completo de sistema Agro (Animais + Culturas)** usando o núcleo do **Ciclo+** (eventos validados em cadeia, evidência Tipo A/B, auditoria imutável, e liquidação condicionada por marcos). É um desenho “setorial” que atende do pequeno ao grande produtor, cooperativa, loja/agropecuária, indústria, frigorífico e prestadores.

Arquitetura do Sistema Agro Ciclo+ (Setor completo)

1) Núcleo obrigatório (igual para tudo)

Toda operação vira **um Evento** e passa pela cadeia:

Autenticação → RBAC → filtro orgId/tenantId → motor de regras → máquina de estados → evidência obrigatória (A/B) → auditoria imutável (append-only + hashes) → liquidação condicionada (escrow/tranches/split, quando aplicável)

Isso garante:

- multi-tenant (sem vazamento entre organizações),
- bloqueio técnico de operações inválidas,
- prova (evidência) proporcional ao risco,
- trilha auditável e resistente a adulteração,
- repasse/pagamento só com marcos verificados.

---

2) Módulos setoriais (Animais + Culturas) sem alterar o núcleo

A) Módulo Cadastros Gerais

- Pessoas: produtor, funcionário, técnico, prestador, fornecedor, comprador, investidor.
- Estruturas: fazenda, talhão, piquete/pasto, curral, armazém/silo, loja/filial, unidade de confinamento.
- Ativos: máquinas, implementos, veículos, balança, leitores (RFID/QR), sensores.

B) Módulo Animais (Bovinos e demais)

Cadastros e gestão

- Animal individual (brinco/RFID) e Lote (banda).
- Categoria: cria/recria/engorda/matriz; confinamento; reprodução.
- Sanidade: calendário, protocolos, medicamentos, carências, laudos.
- Movimentações: piquete, curral, transporte, entrada/saída.
- Pesagens e desempenho: ganho médio diário, conversão, índices.

Eventos típicos

- Nascimento/entrada, identificação, vacinação, medicação, pesagem, apartação, movimentação, mortalidade/descarte, venda, transporte, abate.

C) Módulo Culturas (Grãos, fibras, hortifrúti etc.)

Planejamento e execução

- Safra → Talhão → Cultura → Operações.
- Receituário agronômico e recomendações técnicas.
- Operações agrícolas: preparo, plantio, tratos, pulverização, irrigação, colheita.
- Gestão de insumos por lote (semente, defensivo, fertilizante) com rastreabilidade.
- Monitoramento: pragas/doenças, clima/irrigação, imagens, amostras.

Eventos típicos

- Aplicação de insumo, plantio, pulverização, amostragem, colheita, transporte, recebimento em armazém, classificação, secagem, expedição.

D) Módulo Estoque / Armazém / Farmácia

- Estoque por local (filial/armazém/fazenda).
- Controle por lote e validade (insumos e medicamentos).
- Entrada obrigatória com NF (quando aplicável).
- Baixa por consumo por operação (animal/cultura).
- Perdas com evidência (foto/laudo quando necessário).

E) Módulo Compras

- Solicitação → aprovação → pedido → recebimento → conferência → conciliação fiscal/financeira.
- Regras por **limite de projeto/safra/lote**.

F) Módulo Vendas / Comercial

- Vendas de animais (por lote ou individual), grãos e produtos.
- Propostas, contratos, comissões, marketplace interno (opcional).
- Integração com emissão fiscal (se aplicável).

G) Módulo Logística e Movimentação

- Coleta/entrega, romaneio, rastreio, status.
- Geofence para “entrega/embarque” e operações críticas.

H) Módulo Qualidade, Rastreabilidade e Compliance

- Trilhas por produto/animal/lote.
- Checklists de auditoria (read-only para auditor).
- Relatórios para programas de rastreabilidade e exigências de compradores.

I) Módulo Financeiro (Escrow + Subcontas + Split)

- Subcontas: produtor, projeto/safra, fornecedor, prestador, investidor.
- Escrow por contrato/operação.
- Tranches e split automático por marcos (ex.: entrega confirmada, laudo, pesagem no destino, classificação).

- Antecipação de recebíveis / crédito com garantias (opcional).

J) Analytics / KPIs

- Animais: custo/kg, GMD, mortalidade, taxa de prenhez, custos sanitários.
- Culturas: custo/ha, produtividade, custo por operação, perdas, eficiência de aplicação.
- Financeiro: margem por projeto/safra/lote, inadimplência, ROI de investidor.

Modelo de Dados (essencial) para cobrir setor inteiro

Entidades "mãe"

- **Organization (orgId)**
- **User / Role / Permission**
- **Property (fazenda)**
- **Location** (armazém, curral, silo, loja, talhão, piquete)

Linha Animais

- **Animal** (individual) + **HerdLot** (lote)
- **AnimalMovement** (origem/destino)
- **Weighing** (pesagem)
- **HealthEvent** (vacina/medicação/laudo)
- **Mortality/Discard**

Linha Culturas

- **Season/Safra**
- **Field/Talhão**
- **CropPlan** (planejamento)
- **FieldOperation** (plantio, aplicação, colheita etc.)
- **Sample/Inspection** (amostras, pragas, laudos)
- **HarvestLot** (lote colhido)
- **StorageLot** (lote armazenado/classificado)

Estoque/Compras/Vendas

- **Product/Item**
- **InventoryBalance** / **StockEntry (NF)** / **StockConsumption**
- **PurchaseRequest / PurchaseOrder**
- **Offer / Proposal / Contract**
- **LogisticsRequest / Shipment / DeliveryReceipt**

Núcleo comum

- **OperationalEvent**

- **Evidence (A/B + mídia + geo + timestamp + device attestation + laudo ref)**
- **AuditLog (append-only + hash encadeado)**
- **EscrowAccount / SubAccount / Payment / Receivable / Guarantee**

Máquina de Estados (padrões) para o Agro

Padrão 1 — Compras

SOLICITADO → APROVADO → PEDIDO_EMITIDO → FATURADO → RECEBIDO → CONFERIDO → CONCILIADO → ENCERRADO
Bloqueios típicos: sem limite, sem NF, divergência de quantidade/lote, sem evidência.

Padrão 2 — Operações de Campo (Culturas)

PLANEJADO → EM_EXECUCAO → EXECUTADO → VALIDADO → AUDITADO (opcional)
Marcos: execução + evidência; validação por técnico; auditoria read-only.

Padrão 3 — Lote Animal (bovinos)

ATIVO → EM_MANEJO → EM_MOVIMENTACAO → PRONTO_PARA_VENDA → VENDIDO → EM_TRANSPORTE → ENTREGUE → LIQUIDADO
Marcos: embarque/entrega (GPS+geofence), pesagem no destino (laudo), liberação financeira.

Padrão 4 — Logística

SOLICITADO → A_COLETAR → EM_COLETA → EM_ROTA → ENTREGUE → CONFERIDO → ENCERRADO
Crítico: entrega e conferência (Tipo B obrigatório).

Matriz de Evidências (Tipo A vs Tipo B)

Tipo A (rotineiro – rápido)

- leitura QR/NFC/RFID + timestamp
- confirmação de quantidade/estado
  Usos: pesagens rotineiras, consumo de insumo planejado, checklist simples, movimentação interna controlada.

Tipo B (crítico – prova forte)

- foto/vídeo + GPS + timestamp (+ geofence)
- opcional: laudo, assinatura/atestado do dispositivo
  Usos: mortalidade/descarte, perdas relevantes, embarque/entrega, conferência de

lote, aplicação sensível, colheita/recebimento em armazém, qualquer evento que destrave pagamento.

Motor de Regras (exemplos “setor completo”)

- **Financeiro**
  - bloquear compra se exceder limite do projeto/safra
  - liberar tranche apenas se marco X validado e auditado
- **Estoque**
  - entrada somente com NF (quando aplicável)
  - perda somente com evidência (foto; laudo se crítico)
- **Sanidade**
  - bloquear venda se animal estiver em período de carência
  - exigir laudo para eventos sanitários críticos
- **Culturas**
  - bloquear pulverização sem receita/autorizações exigidas
  - bloquear colheita se operação anterior obrigatória não foi validada (ex.: dessecação quando aplicável)
- **Logística**
  - bloquear entrega se geofence não confere ou GPS ausente

Liquidação Condicionada (escrow/tranches/split) pronta para agro

Caso 1 — Compra de insumos com repasse condicionado

- Escrow recebe valor do projeto
- Tranche 1: NF + recebimento confirmado
- Tranche 2: entrega confirmada (GPS/geofence)
- Split: fornecedor + frete + plataforma (se houver)

Caso 2 — Venda de bovinos por lote com “peso no destino”

- Escrow recebe do comprador
- Marco 1: embarque confirmado (B)
- Marco 2: entrega + pesagem/laudo (B + laudo)
- Split: produtor + comissão + transporte + boitel (se aplicável)

Caso 3 — Grãos (colheita → armazenagem → expedição)

- Escrow/contrato
- Marco 1: recebimento no armazém + classificação (B + laudo)
- Marco 2: expedição/entrega no destino (B)
- Split: produtor + armazenagem + transporte + taxas

MVP recomendado para cobrir “todo o setor” sem ficar gigante

MVP 1 (90% do agro operacional)

1. Cadastros (fazenda/talhão/piquete + pessoas + produtos)
2. Estoque (NF, lote, consumo, perdas)
3. Animais (cadastro, movimentação, sanidade, pesagem)
4. Culturas (safra, talhão, operações, consumo de insumos)
5. Eventos + Evidência + Auditoria imutável

MVP 2 (valor financeiro e diferenciação)

6. Contratos + Escrow + marcos + tranches/split
7. Logística com geofence
8. Portal auditor read-only + relatórios

Se você quiser, eu consigo transformar isso em **documentação de desenvolvimento v1** (pronta para time técnico) em 4 artefatos, todos alinhados ao seu núcleo:

1. **ERD** (tabelas e chaves)
2. **APIs** (endpoints principais por módulo)
3. **Máquinas de estado** detalhadas (transições + bloqueios + evidências)
4. **Catálogo de eventos** (eventType, payload mínimo, tipo de evidência, marcos financeiros)

E eu faço isso já direcionado para **bovinos + culturas** como “primeiros verticais”, sem perder a abrangência do agro inteiro.

Perfeito — dá para modelar o **Agro Ciclo+** para **todos os animais de produção** (bovinos, suínos, aves, ovinos/caprinos, equinos de trabalho, peixes/aquicultura, abelhas, etc.) sem “quebrar” o núcleo. A forma certa é criar uma **camada zootécnica genérica** (AnimalUnit + ProductionGroup + SpeciesProfile) e, por cima, **perfis por espécie** (plugins) que só ajustam regras, evidências e estados.

Abaixo está o modelo já ampliado.

Modelo Agro Ciclo+ para todos os animais de produção

1) Princípio: mesma espinha dorsal, espécies como “perfil”

- **Núcleo**: eventos validados + evidência obrigatória + auditoria imutável + liquidação por marcos.

- **Perfil por espécie**: configura *o que muda* (dados mínimos, estados, regras sanitárias, exigência de evidências, marcos).

---

2) Entidades universais para qualquer animal

A) Espécies e perfis

SpeciesProfile

- `speciesId` (BOVINO, SUINO, AVES, OVINO, CAPRINO, AQUICULTURA, APICULTURA, EQUINO, OUTROS)
- `productionModel` (INDIVIDUAL | LOTE | AMBOS)
- `requiredIdentifiers` (RFID/QR/brinco/anel/caixa/colmeia/tanque)
- `defaultEvidencePolicy` (matriz A/B por operação)
- `healthConstraints` (carências, vacinações, exigências de laudo)
- `stateMachineId` (qual máquina de estados aplicar)

B) Unidade animal (individual ou grupo)

AnimalUnit (genérico)

- `unitId`, `orgId`, `speciesId`, `propertyId`, `locationId`
- `identityType` (INDIVIDUAL | GRUPO)
- `primaryId` (RFID/QR/brinco/anel/nº tanque/nº colmeia)
- `status` (ATIVO/INATIVO/TRANSFERIDO/BAIXADO)
- `birthOrEntryDate`
- `attributes` (JSON: raça/linhagem, sexo, idade, categoria produtiva)

ProductionGroup (quando a unidade é “lote”)

- `groupId`, `orgId`, `speciesId`, `propertyId`, `locationId`
- `count`, `avgWeight?`, `phase` (cria/engorda/term./reprod./crescimento etc.)
- `batchRef` (origem genética/compra)
- `status`

Para aves e suínos, “grupo” é o padrão. Para bovinos e equinos, “individual” é comum. Para aquicultura, “tanque/lote” é padrão. Para apicultura, a unidade pode ser “colmeia”.

C) Localização e ambiente (serve para todas)

AnimalLocation

- `locationId`, `orgId`, `type` (PIQUETE, GALPAO, BAIA, GRANJA, TANQUE, VIVEIRO, APIARIO, CURRAL, CONFINAMENTO)
- `geo` + `geofence`
- `capacity`, `biosecurityLevel`

D) Sanidade e biossegurança (genérico)

HealthProtocol

- `protocolId`, `orgId`, `speciesId`, `rules` (vacinas, manejo, carências, exames)

HealthEvent

- `healthEventId`, `orgId`, `unitId/groupId`, `type` (VACINA, MEDICACAO, EXAME, QUARENTENA, TRATAMENTO)
- `productId` (medicamento), `dose`, `operatorId`
- `carenciaEndAt?`, `laudoRef?`, `evidenceRef`, `status`

E) Alimentação e insumos (genérico)

FeedInputEvent

- `feedEventId`, `orgId`, `unitId/groupId`, `feedType`, `qty`, `timestamp`
- vínculo com estoque/insumo

F) Eventos de produção (genérico)

ProductionEvent

- `prodEventId`, `orgId`, `unitId/groupId`, `type`
    - exemplos: NASCIMENTO/ENTRADA, PESAGEM/BIOMETRIA, MOVIMENTACAO, REPRODUCAO, PARTO, COLETA (leite/ovos/mel), MORTALIDADE, DESCARTE, VENDA/TRANSFERENCIA
- `metrics` (JSON: peso, ganho, produção diária, conversão, mortalidade, etc.)
- `evidenceRef`, `status`

---

3) Máquinas de estados por “modelo produtivo” (3 padrões)

Padrão S1 — Individual (bovinos, equinos, ovinos/caprinos em controle fino)

ATIVO → EM_MANEJO → EM_MOVIMENTACAO → PRONTO_PARA_COMERCIALIZACAO → COMERCIALIZADO → EM_TRANSPORTE → ENTREGUE → LIQUIDADO
Bloqueios: carência sanitária, ausência de evidência crítica, divergência de identificação.

Padrão S2 — Lote/Granja (aves e suínos)

LOTE_CRIADO → EM_CRESCIMENTO → EM_TERM / PRONTO → SAIDA_PROGRAMADA → EMBARCADO → ENTREGUE → LIQUIDADO

Eventos críticos: mortalidade acima do limite, medicação com carência, embarque/entrega, classificação/peso.

Padrão S3 — Ambiente controlado (aquicultura / apicultura)

ATIVO → CICLO_EM_ANDAMENTO → MANEJO_CRITICO (quando ocorre) → COLHEITA/RETIRADA → CLASSIFICACAO → ENTREGA → LIQUIDADO
Crítico: parâmetros ambientais fora do limite, mortalidade em massa, colheita, laudos, destino.

---

4) Matriz de evidência (A/B) ajustada para todas as espécies

Evidência Tipo A (rotineira)

- leitura de ID (RFID/QR/anel/etc.) + timestamp
- confirmação de contagem/estado
  Usos típicos:
- movimentação interna rotineira
- alimentação registrada
- pesagem rotineira / biometrias simples
- checklists de manejo padrão

Evidência Tipo B (crítica)

- foto/vídeo + GPS + timestamp (+ geofence)
- opcional: laudo, atestado do dispositivo
  Usos típicos:
- mortalidade/descarte
- medicação crítica / quarentena / exames
- embarque/entrega e conferência no destino
- colheita (aquicultura) / retirada e classificação
- ocorrências sanitárias ou perdas relevantes

---

5) Regras universais do motor (válidas para qualquer animal)

- **Venda bloqueada** se existir `carenciaEndAt` futura (sanidade).
- **Movimentação bloqueada** se origem/destino não autorizado (biossegurança).
- **Mortalidade** acima de limite do perfil exige Tipo B + laudo (quando configurado).
- **Embarque/entrega** sempre crítico: Tipo B + geofence.
- **Integração com estoque**: medicamento/ração só pode ser baixado se houver saldo e lote válido.

---

6) Liquidação condicionada (escrow) aplicável a qualquer cadeia animal

Caso A — Venda por peso/qualidade (muito comum)

Marcos:

1. Embarque confirmado (B)
2. Entrega confirmada (B)
3. Pesagem/classificação/laudo (B + laudo, quando aplicável)
   → libera tranches e executa split (produtor + comissão + frete + serviço)

Caso B — Prestação de serviço (vacinação, manejo, inseminação)

Marcos:

1. Serviço executado (B ou A conforme criticidade)
2. Evidência validada e auditada
   → libera pagamento do prestador

Caso C — Integração (granja/frigorífico/integradora)

Marcos:

1. Lote atingiu fase/índice mínimo (evento validado)
2. Saída/entrega
3. Classificação (qualidade)
   → split entre integradora, produtor, insumos/adiantamentos

---

"Todos os animais" na prática: como configurar sem reescrever o sistema

Você implementa:

1. **Um conjunto universal de entidades e APIs** (AnimalUnit, ProductionGroup, HealthEvent, ProductionEvent, Movement, Evidence, AuditLog, Escrow)
2. Um **SpeciesProfile** com:
   - quais telas aparecem,
   - estados/transições,
   - matriz A/B,
   - limites e regras sanitárias,
   - marcos de liquidação mais comuns.

MVP recomendado (setor animal inteiro)

1. Cadastros: espécie + propriedade + locais (piquetes/galpões/baías/tanques/colmeias)
2. Unidades: AnimalUnit (individual) + ProductionGroup (lote)
3. Operações: entrada, movimentação, sanidade, mortalidade/descarte, produção (quando aplicável)
4. Evidência A/B + auditoria imutável
5. Venda/transferência + logística (embarque/entrega)
6. Escrow por marcos (embarque/entrega/classificação)

Se você quiser, eu continuo com o próximo nível e te devolvo:

- um **Catálogo de Eventos** (eventType, payload mínimo, evidência A/B, estados afetados) para **todas as espécies**, e
- um **mapa de telas** por perfil (produtor/técnico/loja/frigorífico/auditor) já pronto para produto/MVP.

Posso sim — e dá pra fazer isso de um jeito **coeso** (não "colcha de retalhos"): você mantém o **núcleo Ciclo+** como *motor de eventos + estados + evidência + auditoria + liquidação*, e adiciona 6 grandes camadas: **Marketplace (normal + inverso)**, **Contratos/escrow**, **Acompanhamento técnico**, **Integrações de campo (drones/tratores/IoT/leitores/balanças)**, **Laboratório**, e **IA**. Abaixo vai o desenho completo.

---

1) Marketplace completo: normal + inverso (reverso)

A) Marketplace "normal" (ofertas do vendedor)

- **Vendedor** (produtor, loja, fornecedor) publica **Oferta**: animal/lote, grãos/lote, serviço, insumo, crédito/garantia.
- **Comprador** envia proposta/compra.
- O Ciclo+ amarra: **contrato + escrow + marcos + evidência + auditoria**.

Entidades

- `Listing` (oferta): item, preço, disponibilidade, local, validade, requisitos (laudo, rastreio)
- `Proposal/Bid`: preço, quantidade, prazo, condições
- `Match/Deal`: proposta aceita
- `Contract`: contrato gerado
- `EscrowPlan`: plano de tranches/split por marcos

B) Marketplace inverso / reverso (demanda do comprador)

Aqui o **comprador publica uma demanda**, e **vendedores competem** para atender.

Tipos de demanda

1. **Indústria/frigorífico**: "Preciso de 500 bois padrão X, entrega em Y, janela Z, qualidade W, laudos obrigatórios"
2. **Cooperativa/armazenador**: "Preciso de milho loteado com umidade ≤…"
3. **Fornecedor**: "Compra de grãos/animais com condições"
4. **Produtor**: "Preciso comprar ração/insumo com entrega e financiamento"
5. **Prestador**: "Demanda por serviço (pulverização, manejo, vacinação)"

Entidades

- `Demand` (demanda): specs, volumes, janela, local, requisitos de evidência, score de qualidade, regra de aceitação
- `SupplierResponse` (resposta do fornecedor): proposta com preço, lead time, capacidade, garantias, evidências
- `DemandAuction` (leilão reverso): rodada, ranking, critérios, fechamento automático ou manual
- `PreContract` (pré-contrato): travas e validações antes de virar contrato final

Regras essenciais (motor)

- só entra no ranking se: capacidade verificada + documentação mínima + evidências mínimas (quando exigido)
- "melhor oferta" não é só preço: considera SLA, compliance, histórico auditado, geofence, laudos

---

2) Contratos: assinatura, execução por marcos e risco

Tipos de contrato (templates)

- **Compra/Venda** (animais, grãos, insumos)
- **Prestação de serviço** (campo, manejo, drone)
- **Consignação/custódia**
- **Integração/Parceria** (integradora-produtor)
- **Financiamento/antecipação/garantia**
- **Logística** (coleta/entrega/transportadora)

Execução por "marcos" (milestones)

Cada contrato vira uma **máquina de estados** e cada marco exige evidência.

Exemplo (animais por peso no destino):

1. `CONTRATO_ASSINADO`
2. `EMBARQUE_CONFIRMADO` (Tipo B + geofence + mídia)
3. `ENTREGA_CONFIRMADA` (Tipo B + geofence)
4. `PESAGEM/CLASSIFICACAO` (Tipo B + laudo)
5. `LIQUIDACAO` (escrow libera tranche/split)

6. `ENCERRADO`

Liquidação condicionada (escrow/tranches/split)

- `EscrowHold`: valor entra e fica retido
- `TrancheRule`: libera X% no marco Y
- `SplitRule`: divide para produtor/transportadora/comissão/serviço/impostos
- `DisputeRule`: se marco falhar, retém e abre disputa

Módulo de disputa (muito importante no marketplace)

- `DISPUTE_OPENED` com evidência e motivo
- auditor read-only valida, emite parecer
- resolução libera/estorna/ajusta split (tudo auditado)

---

3) Acompanhamento técnico (assistência, TRT, rastreio e compliance)

"Planos técnicos" por produção (animal e cultura)

- Plano = conjunto de **tarefas/eventos esperados** com janela de execução.
- Ex.: protocolo sanitário, manejo de pasto, cronograma de aplicação, colheita.

Entidades

- `TechPlan` (plano técnico): por propriedade/talhão/lote/unidade animal
- `TechVisit` (visita): checklists + recomendações + evidências
- `Prescription/Recommendation` (receita/recomendação): regras e documentos
- `NonConformity` (não conformidade): abre correção com prazo

O que o Ciclo+ garante aqui

- tarefa crítica não "fecha" sem evidência
- recomendação vira evento auditado
- indústria pode exigir "compliance mínimo" como requisito de demanda/contrato

---

4) Integrações de tecnologia: drones, tratores, IoT, leitores, balanças

A regra é: **tudo que vem do mundo físico entra como evidência e/ou evento**.

A) Drones (mapeamento, NDVI, falhas, contagem, inspeção)

- `DroneMission`: plano de voo + área + operador + equipamento

- `DroneCapture`: imagens/vídeo/ortomosaico + metadados (timestamp, local)
- `DroneFinding`: anomalias (falha de plantio, erosão, invasoras, cerca rompida)

Usos práticos

- comprovar execução de serviço (pulverização/inspeção)
- comprovar marco do contrato ("área inspecionada", "talhão colhido", "pasto avaliado")

Evidência

- geralmente Tipo B (mídia + geo + timestamp), com hash e referência

B) Tratores/implementos (telemetria e operação mecanizada)

- `MachineOperationEvent`: plantio, aplicação, colheita, gradagem etc.
- captura: hora, trilha GPS, velocidade, implemento, taxa de aplicação (quando disponível)

Marcos

- "operação executada" pode ser validada por telemetria (reduz fraude)

C) Leitor identificador animal (RFID/QR/NFC)

- `IdentityScanEvent`: leitura + timestamp + operador + local
- alimenta: movimentação, pesagem, medicação, inventário de lote

Tipo de evidência

- em geral Tipo A (rápido) — vira Tipo B se o evento for crítico (ex.: descarte/mortalidade)

D) Pesagem (balança, tronco, caminhão, frigorífico)

- `WeightEvent`: peso, unidade, identificação, local, balançaId, operadorId
- opcional: `ScaleAttestation` (certificação/calibração / foto do display)

Em contratos por peso

- pesagem + laudo = marco liberador de pagamento (Tipo B + laudo)

E) IoT e sensores (ambiente/água/galpão)

- `SensorReadingEvent`: temperatura, umidade, amônia, pH, oxigênio etc.
- regras: se sair do range, abre evento crítico e obriga evidência extra/ação corretiva

---

5) Análise laboratorial: cadeia de custódia (Chain of Custody)

Para laboratório virar prova forte, você precisa de "cadeia de custódia".

Fluxo

1. `SAMPLE_COLLECTED` (Tipo B: foto + geo + timestamp)
2. `SAMPLE_SEALED` (id do lacre/QR)
3. `SAMPLE_SHIPPED` (logística rastreada)
4. `SAMPLE_RECEIVED_LAB` (confirmação + integridade)
5. `LAB_RESULT_ISSUED` (PDF/resultado assinado digitalmente)
6. `RESULT_LINKED_TO_EVENT/CONTRACT` (marco)

Entidades

- `Sample`, `Seal`, `LabOrder`, `LabResult`, `CoCRecord`

Isso atende animal (sanidade, resíduos) e cultura (solo, folha, grão).

---

6) IA: assistente, validação, detecção de risco e recomendação

IA aqui não pode ser "mágica": ela vira **motor auxiliar**, sempre auditável.

Blocos de IA

1. **Assistente operacional**: "qual a próxima etapa?", "por que bloqueou?", "qual ação corretiva?"
2. **Classificação de criticidade**: sugere A ou B (motor decide)
3. **Detecção de anomalia/fraude**:
   - peso fora do padrão
   - geofence suspeito
   - mídia repetida
   - divergência estoque vs consumo
4. **Visão computacional** (opcional):
   - qualidade de carcaça/animal (se houver fotos)
   - falha de plantio em imagem de drone
5. **Score de fornecedor/produtor** para marketplace inverso:
   - histórico de entrega, compliance, disputas, laudos

Governança

- toda decisão automatizada registra: `aiDecisionId`, input, output, modelo/versão, confiança, e se houve override humano (auditoria).

---

7) Visualização de demandas pela indústria e fornecedores (normal e inverso)

Painéis (Dashboards)

Indústria/Comprador

- demandas abertas, ranking de respostas, SLA, compliance, histórico
- mapa (geo) de origens/rotas
- evidências e laudos por contrato
- status dos marcos e previsão de liquidação

Fornecedores/Produtores

- oportunidades do marketplace inverso (filtros por local, janela, espécie/cultura, certificações)
- "checklist de habilitação" (o que falta para atender uma demanda)
- simulação de split e custos logísticos

Auditor/Compliance (read-only)

- trilhas de eventos e hashes
- validação de evidências e laudos
- pareceres e disputas

---

8) MVP grande, mas realista (para cobrir tudo sem travar)

MVP-1 (Marketplace + Contratos + Evidências)

- Cadastro universal (animais/culturas/itens/locais)
- Marketplace normal + inverso (Demand + Responses)
- Contratos com 2–3 templates
- Escrow + tranches + split
- Evidência Tipo A/B + auditoria imutável

MVP-2 (Tecnologias e prova forte)

- Integração: leitor animal + pesagem
- Logística + geofence
- Coleta de amostras + laudo laboratorial (CoC)
- Drone: anexar missão/captura como evidência (mesmo sem análise automática no início)

MVP-3 (IA e automações)

- assistente de bloqueios/ações corretivas
- scoring de propostas no marketplace inverso

- detecção de anomalias e antifraude

---

9) Regras-chave para “garantir” marketplace inverso de verdade

Para não virar “mural de recados”, você precisa destes mecanismos técnicos:

1. **Demand Spec** padronizado (requisitos mínimos + evidências obrigatórias)
2. **Pré-qualificação** (RBAC + KYC/credenciais + histórico)
3. **Ranking automático** (preço + prazo + compliance + risco)
4. **Fechamento com contrato automático** (gera contrato e escrow)
5. **Execução por marcos** (sem marco, sem pagamento)
6. **Disputa** com auditor read-only (parecer sem alterar histórico)

Se você quiser, eu faço o próximo passo “de engenharia”, já pronto para time dev:

1. **Catálogo de Eventos** (50–80 eventos base) cobrindo animais + culturas + marketplace + contratos + integrações
2. **APIs principais** (endpoints e payload mínimo)
3. **State machines detalhadas** para: Demand, Proposal, Contract, Shipment, Lab CoC, AnimalLot, FieldOperation
4. **Matriz A/B por operação** com gatilhos de criticidade (quando vira Tipo B ou B+laudo)

Só me diga qual prioridade de go-to-market:

- **(A)** indústria comprando (demanda inversa)
- **(B)** produtor comprando insumo/serviço (demanda inversa)
- **(C)** venda de animais/grãos com escrow por marcos (marketplace normal)

Fechado — prioridade **(A) Indústria comprando via demanda inversa**. Abaixo vai o pacote “pronto para build” do **Marketplace Inverso Industrial** no Ciclo+ (com contratos, acompanhamento técnico, integrações e marcos que destravam pagamento).

---

1) Produto: como funciona o Marketplace Inverso Industrial

Perfis

- **Indústria/Comprador** (ex.: frigorífico, laticínio, integradora, cerealista, agroindústria)
- **Fornecedor** (produtor, cooperativa, trading, confinamento, granja, piscicultura, apicultor)
- **Técnico** (assistência técnica/certificador)
- **Transportador**
- **Auditor (read-only)**

Fluxo resumido

1. Indústria cria **Demanda** (especificação + janela + local + requisitos + marcos)
2. Sistema abre **Rodada** (leilão reverso) e notifica fornecedores elegíveis
3. Fornecedores enviam **Respostas** (preço + capacidade + evidências + cronograma)
4. Sistema faz **Ranking** (preço + SLA + compliance + risco)
5. Indústria seleciona → gera **Contrato** + plano de **Escrow/Tranches/Split**
6. Execução por **Marcos verificáveis** (embarque/entrega/classificação/laudos)
7. **Liquidação automática** por marcos; se falhar, retém e registra motivo (disputa)

---

2) Dados e objetos principais (modelo mínimo)

Demand (Demanda Industrial)

Campos mínimos:

- `demandId`, `orgIdBuyer` (indústria), `category` (ANIMAIS/CULTURAS/DERIVADOS/SERVICOS)
- `productSpec` (JSON): espécie/cultura, padrão, volume, unidade
- `qualitySpec`: critérios (ex.: peso/idade/umidade/classificação), tolerâncias
- `deliveryWindow`: início/fim + `deliveryLocation` (geo + geofence)
- `evidencePolicy`: obrigatoriedade A/B por marco
- `labRequirements`: quais laudos e limites (quando aplicável)
- `contractTemplateId`
- `awardRule`: melhor preço / melhor score / multi-winner
- `state`: DRAFT → OPEN → BIDDING → AWARDED → CONTRACTED → CLOSED/CANCELLED

SupplierResponse (Proposta do fornecedor)

- `responseId`, `demandId`, `orgIdSupplier`
- `price`, `quantity`, `leadTime`, `shipmentPlan`
- `complianceDocsRefs` (se houver)
- `evidenceCommitment` (o que ele vai comprovar e como)
- `state`: SUBMITTED → SHORTLISTED → REJECTED → AWARDED

Contract + EscrowPlan

- `contractId`, `demandId`, partes, valor, condições
- `escrowId`, `trancheRules[]`, `splitRules[]`, `disputeRules[]`
- `state`: SIGNED → IN_EXECUTION → COMPLETED → DISPUTED → CLOSED

---

3) Catálogo de eventos (núcleo industrial inverso)

Abaixo estão os **eventTypes** base que você precisa (com indicação de evidência).

Regra: todo evento gera `AuditLog` (append-only + hash encadeado).

A) Demanda e rodada (comprador)

1. `DEMAND_CREATED` (A)
2. `DEMAND_PUBLISHED` (A)
3. `BIDDING_ROUND_OPENED` (A)
4. `BIDDING_ROUND_EXTENDED` (A)
5. `DEMAND_UPDATED` (A) (com versionamento)
6. `DEMAND_CANCELLED` (A)

B) Propostas (fornecedor)

7. `SUPPLIER_RESPONSE_SUBMITTED` (A)
8. `SUPPLIER_RESPONSE_UPDATED` (A)
9. `SUPPLIER_RESPONSE_WITHDRAWN` (A)
10. `SUPPLIER_SHORTLISTED` (A)
11. `SUPPLIER_AWARDED` (A)
12. `SUPPLIER_REJECTED` (A)

C) Contrato e escrow

13. `CONTRACT_GENERATED` (A)
14. `CONTRACT_SIGNED` (A) (assinatura digital/ref)
15. `ESCROW_FUNDED` (A)
16. `SPLIT_RULES_SET` (A)
17. `TRANCHE_RULES_SET` (A)

D) Execução logística (crítico)

18. `SHIPMENT_SCHEDULED` (A)
19. `PICKUP_CONFIRMED` (B: GPS+geofence+mídia)
20. `IN_TRANSIT_UPDATE` (A ou B conforme risco)
21. `DELIVERY_CONFIRMED` (B: GPS+geofence+mídia)
22. `RECEIPT_ACKNOWLEDGED` (A) (recebimento administrativo)

E) Classificação, pesagem e qualidade (crítico)

23. `WEIGHING_RECORDED` (B; pode anexar laudo/atestado)
24. `QUALITY_INSPECTION_RECORDED` (B; mídia/relatório)
25. `LAB_SAMPLE_COLLECTED` (B: foto+geo+timestamp)
26. `LAB_SAMPLE_RECEIVED` (A/B conforme exigência)
27. `LAB_RESULT_ISSUED` (A, com documento assinado ref)

28. `QUALITY_APPROVED` (A)
29. `QUALITY_REJECTED` (A) (abre disputa)

F) Liquidação e disputa

30. `MILESTONE_VERIFIED` (A) (resultado do motor)
31. `TRANCHE_RELEASED` (A)
32. `SPLIT_EXECUTED` (A)
33. `PAYMENT_HELD` (A) (motivo + ação corretiva)
34. `DISPUTE_OPENED` (A/B conforme caso)
35. `AUDITOR_OPINION_ISSUED` (A) (read-only)
36. `DISPUTE_RESOLVED` (A)
37. `CONTRACT_CLOSED` (A)

---

4) Máquinas de estado detalhadas (MVP industrial)

Demand state machine

- DRAFT
  - `DEMAND_CREATED`
- OPEN
  - `DEMAND_PUBLISHED`
- BIDDING
  - `BIDDING_ROUND_OPENED`
- AWARDED
  - `SUPPLIER_AWARDED` (um ou múltiplos)
- CONTRACTED
  - `CONTRACT_SIGNED` + `ESCROW_FUNDED`
- CLOSED
  - `CONTRACT_CLOSED` / `DEMAND_CANCELLED`

Bloqueios:

- publicar sem `deliveryWindow`, `qualitySpec`, `awardRule`
- permitir proposta só para fornecedores elegíveis (RBAC + requisitos)

Contract execution state machine

SIGNED → IN_EXECUTION → (DELIVERED) → QUALITY_VERIFIED → LIQUIDATED → CLOSED
Transições críticas:

- IN_EXECUTION só após escrow funded (se exigido)
- DELIVERED exige `DELIVERY_CONFIRMED` (B)
- QUALITY_VERIFIED exige (pesagem/classificação/laudo conforme demanda)
- LIQUIDATED só após `MILESTONE_VERIFIED` (tudo auditado)

---

5) Regras do motor (o que “garante” o inverso)

Elegibilidade e ranking

- Só aceita `SUPPLIER_RESPONSE_SUBMITTED` se:
  - fornecedor tem perfil e permissões
  - capacidade mínima e janela compatível
  - requisitos documentais mínimos anexados (se demandar)
- Ranking = Score ponderado:
  - preço (peso X)
  - SLA/lead time (peso Y)
  - histórico de entrega/qualidade (peso Z)
  - risco (fraude/anomalia) penaliza

Execução por marcos

- `TRANCHE_RELEASED` somente se:
  - marco correspondente tem evento + evidência válida
  - eventos são consistentes com máquina de estados
  - auditoria está íntegra (hash encadeado ok)
- Se falhar:
  - gerar `PAYMENT_HELD` + motivo + ação corretiva

Disputa

- abre disputa se:
  - qualidade reprovada
  - entrega fora de geofence
  - divergência de peso/classificação
- auditor read-only emite parecer (`AUDITOR_OPINION_ISSUED`)
- resolução libera/ajusta/estorna (tudo auditado)

---

6) Integrações essenciais para Indústria (A)

Leitor animal / Identificação

- Integração via app/câmera/leitor RFID
- Evento: `IDENTITY_SCAN_RECORDED` (Tipo A) ligado ao lote/embarque

Pesagem

- Balança do embarque e do destino
- Evento: `WEIGHING_RECORDED` (Tipo B + opcional atestado/calibração)

Laudos laboratoriais / classificação

- Integração por upload assinado (PDF) ou API do laboratório

- Eventos CoC: `LAB_SAMPLE_COLLECTED` → `LAB_RESULT_ISSUED`

Drones / tratores (principalmente para culturas, mas pode apoiar auditoria)

- Drone: evidência de colheita/condição/área (Tipo B)
- Trator: telemetria de operação (validação de execução)

IA

- Assistente de "bloqueio e ação corretiva" (para operador/fornecedor)
- Anomalias: peso fora do padrão; mídia repetida; geofence suspeito
- Score para ranking do inverso (explicável e auditável)

---

7) Telas/painéis que a Indústria precisa (MVP)

Indústria

- Criar Demanda (template + requisitos)
- Painel de Rodada (propostas em ranking, filtros, shortlist)
- Selecionar vencedor (multi-winner opcional)
- Contrato + plano de marcos (visual)
- Execução: mapa de embarques/entregas + status
- Qualidade: pesagem/classificação/laudos
- Financeiro: escrow, tranches, split, retenções, disputas

Fornecedor

- Demandas abertas (filtros por local/janela)
- Checklist de habilitação (o que falta para participar)
- Enviar proposta
- Execução: embarque/entrega com evidência guiada
- Recebíveis e tranches (visão clara do que falta liberar)

Auditor (read-only)

- trilha de eventos e evidências
- verificação de integridade
- parecer em disputa

---

8) Checklist "garantia" do marketplace inverso (o que não pode faltar)

1. Demanda com **spec + critérios + evidência** (não só texto livre)
2. Ranking automático com critérios auditáveis

3. Contrato gerado ao premiar
4. Escrow + tranches/split
5. Marcos com evidência Tipo B onde importa (embarque/entrega/qualidade)
6. Retenção e disputa com auditor read-only
7. Logs imutáveis (append-only + hash encadeado)

Próximo passo (eu continuo agora se você quiser)

Eu posso detalhar imediatamente (sem depender de mais nada):

1. **APIs REST** principais (endpoints e payload mínimo) para Demand, Response, Contract, Shipment, Weighing, Lab, Escrow
2. **Templates de Demanda Industrial** prontos (ex.: "boi padrão frigorífico", "milho classe X", "soja umidade Y", "serviço de colheita")
3. **Modelo de ranking** (score ponderado) + regras antifraude

Me diga só qual é a indústria alvo primeiro:

- **frigorífico (carne)**, **laticínio**, **cerealista/grãos**, ou **agroindústria de insumos**?

Perfeito — então o sistema precisa suportar **as 3 modalidades de negociação** no mesmo modelo (e até misturar: ex. preço por carcaça + mínimo por peso vivo). A forma "limpa" é: **o contrato escolhe um "método de precificação"** e o Ciclo+ liga **marcos + evidências** a esse método, para que a liquidação (escrow/tranches/split) funcione sem ambiguidade.

Abaixo vai o desenho definitivo do **Frigorífico (produção sob demanda) + módulo leilão separado**, com suporte às 3 formas.

---

1) Produção sob demanda: objetos e fluxo (sem leilão)

1.1 Capacidade (Fornecedor → "eu tenho X para entregar")

CapacityOffer

- `species` (bovino/suíno/aves/…)
- `quantity` (cabeças / kg vivo estimado / kg carcaça estimado)
- `deliveryWindow` (início/fim)
- `pickupGeo` + `geofence`
- `qualityEnvelope` (faixa de peso/idade/acabamento/sexo etc.)
- `sanitaryStatus` + `withdrawalConstraints` (carências)
- `commitmentStatus`: UNCOMMITTED | RESERVED | CONTRACTED

1.2 Demanda do frigorífico (Indústria → "preciso de X com padrão Y")

BuyerDemand

- `requiredVolume` + `window`
- `destinationPlant` (unidade frigorífico) + geofence
- `qualitySpec` (mínimos e tolerâncias)
- `pricingAllowedModes`: {LIVE_WEIGHT, DEST_WEIGHT, CARCASS_GRADE}
- `requiredEvidencePolicy` por marco (A/B)

1.3 Matching + Reserva + Contrato

1. Frigorífico filtra e seleciona fornecedores/capacidades
2. Sistema faz `CAPACITY_RESERVED` (com expiração)
3. Gera `CONTRACT_DRAFTED` via template
4. Partes negociam no chat/contraproposta (auditável)
5. `CONTRACT_SIGNED`
6. Execução por marcos (embarque/entrega/pesagem/classificação)
7. Liquidação conforme método de preço escolhido

---

2) Contrato: o "motor" que suporta as 3 formas de preço

2.1 Campo-chave: método de precificação

No contrato, você grava:
`pricingMethod` = LIVE_WEIGHT | DEST_WEIGHT | CARCASS_GRADE | HYBRID

E cada método "ativa" um conjunto de marcos obrigatórios e evidências.

A) LIVE_WEIGHT (peso vivo na fazenda)

Marcos mínimos

- `PICKUP_CONFIRMED` (B: geofence + mídia)
- `ORIGIN_WEIGHING_RECORDED` (B: pesagem + atesto opcional)
- `DELIVERY_CONFIRMED` (B)
- `QUALITY_INSPECTION` (opcional, conforme contrato)

Liquidação

- Base = peso vivo medido na origem
- Ajustes: mortalidade/transporte, bonificações/penalidades (se pactuadas)
- Pode liberar tranches:
  - tranche 1 após pesagem origem
  - tranche 2 após entrega

B) DEST_WEIGHT (peso no destino)

Marcos mínimos

- `PICKUP_CONFIRMED` (B)
- `DELIVERY_CONFIRMED` (B)
- `DEST_WEIGHING_RECORDED` (B)
- `QUALITY_INSPECTION_RECORDED` (B, se contrato exigir)

Liquidação

- Base = peso aferido no frigorífico
- Ajustes: diferença de peso vs envelope, SLA de entrega, perdas

C) CARCASS_GRADE (carcaça/classificação)

Marcos mínimos

- `PICKUP_CONFIRMED` (B)
- `DELIVERY_CONFIRMED` (B)
- `DEST_WEIGHING_RECORDED` (B) (muitas plantas pesam na entrada)
- `CARCASS_CLASSIFICATION_RECORDED` (B + laudo/relatório)
- (opcional) `LAB_RESULT_ISSUED` (A, com documento assinado) quando exigido

Liquidação

- Base = tabela por tipificação/grade (contrato define)
- Ajustes por rendimento, padrão, não conformidade

D) HYBRID (combinações)

Exemplos comuns:

- preço base por **peso vivo** + ajuste por **classificação**
- mínimo garantido por **peso vivo** e acerto final por **carcaça**
- peso no destino com **gatilhos** de grade

Como implementar

- Contrato define `primaryMethod` + `settlementAdjustments[]`
- Liquidação ocorre em 2 fases:
    1. **liquidação parcial** no marco do primaryMethod
    2. **acerto final** após classificação/laudo

---

3) Evidência e integrações obrigatórias (para dar lastro)

3.1 Embarque e entrega (sempre críticos)

- **Tipo B**: foto/vídeo + GPS + timestamp + geofence ok
- Motivo: é o “marco físico” que destrava pagamento e evita fraude

3.2 Leitor de identificação animal (RFID/QR)

- Evento: `IDENTITY_SCAN_RECORDED` (Tipo A)
- Vincula: lote/animal → embarque → contrato
- Gatilho: se contrato exigir rastreio individual, o sistema bloqueia entrega sem leitura mínima

3.3 Pesagem (origem e/ou destino)

- Evento: `WEIGHING_RECORDED` (Tipo B)
- Anexo opcional: foto do display, arquivo exportado da balança, calibração
- Regra: método de preço define qual pesagem é "oficial"

3.4 Classificação e laudos

- Evento: `QUALITY_INSPECTION_RECORDED` / `CARCASS_CLASSIFICATION_RECORDED` (Tipo B)
- Laboratório: cadeia de custódia (se exigida)
  - `LAB_SAMPLE_COLLECTED` (B) → `LAB_RESULT_ISSUED` (A, assinado)

---

4) Liquidação (escrow/tranches/split) para as 3 formas

Modelo padrão de "tranches"

- **Tranche 1 (opcional):** após embarque confirmado
- **Tranche 2:** após entrega confirmada
- **Tranche 3:** após pesagem "oficial" (origem/destino)
- **Tranche 4 (acerto final):** após classificação/laudo (quando houver)

Split automático

- produtor/fornecedor
- transportadora/frete
- comissão/plataforma (se houver)
- serviços técnicos (se vinculados ao contrato)

Se marco falhar

- sistema gera `PAYMENT_HELD` com motivo e ação corretiva sugerida
- abre `DISPUTE_OPENED` quando necessário
- auditor read-only dá parecer

---

5) Negociação entre as partes (sem virar bagunça)

Você permite negociação livre **mas auditável e versionada**:

5.1 Chat + Propostas versionadas

- `NEGOTIATION_MESSAGE_SENT` (A)
- `CONTRACT_TERM_PROPOSED` (A) (alteração de preço, tolerância, janela, método)
- `CONTRACT_TERM_ACCEPTED/REJECTED` (A)

5.2 Guardrails (para não quebrar o núcleo)

Mesmo com negociação livre:

- contrato só "fecha" se tiver:
  - método de preço escolhido
  - marcos mínimos configurados
  - evidência exigida definida
  - regra de disputa definida

---

6) Módulo de Leilão separado (para ativos livres)

Só entra em leilão se `UNCOMMITTED`

- gerenciado por `ROLE_LEILOEIRO_CREDENCIADO`
- gera contrato no fechamento do leilão
- segue os mesmos marcos e evidências (embarque/entrega/pesagem/classificação)

---

7) Checklist de implementação (MVP frigorífico)

MVP-1 (produção sob demanda com negociação 3 modos)

1. CapacityOffer (fornecedor publica capacidade)
2. BuyerDemand (frigorífico publica demanda e filtra)
3. Matching + Reserve + Contract templates
4. Execução: embarque/entrega + pesagens + (opcional) classificação
5. Liquidação por marcos (sem ou com escrow total)

MVP-2 (rastreio forte)

6. Integração leitor RFID/QR
7. Integração balança (upload/API)
8. Disputa + auditor read-only

MVP-3 (leiloeiro e leilão spot)

9. Módulo de leilão para lotes livres
10. regras antifraude e scoring

Próximo passo (eu continuo já)

Eu posso te entregar agora (em formato bem prático):

1. **Template de contrato frigorífico** em "campos do sistema" (cláusulas parametrizadas)
2. **Lista de telas** para Indústria, Fornecedor, Técnico, Leiloeiro
3. **APIs** (endpoints e payloads) para CapacityOffer, Demand, Contract, Shipment, Weighing, Classification, Escrow

Só me diga: vocês vão usar **escrow obrigatório** (dinheiro retido) ou **liquidação condicionada sem depósito total** (pagamento liberado por marcos, mas sem reter todo o valor antecipado)?

Perfeito. Com **escrow obrigatório** em **produção sob demanda** e também **após lance vencedor no leilão**, o seu fluxo fica extremamente "Ciclo+" (marcos + evidência + auditoria + liberação automática). Abaixo está o modelo final, já com **templates de contrato parametrizados**, **máquinas de estado**, **marcos**, **evidências**, **regras de bloqueio** e **telas/APIs**.

---

1) Fluxos oficiais do sistema (Frigorífico)

Fluxo 1 — Produção sob demanda (negociação direta + escrow obrigatório)

1. Fornecedor publica **Capacidade** (livre)
2. Frigorífico cria **Demanda** e seleciona capacidade
3. Negociam termos (método: peso vivo / destino / carcaça, janela, tolerâncias)
4. Gera **Contrato** e assinam (`CONTRACT_SIGNED`)
5. **Escrow obrigatório**: frigorífico deposita (`ESCROW_FUNDED`)
6. Execução:
   - **Pesagem "oficial"** conforme contrato (`WEIGHING_RECORDED` + evidência)
   - **Embarque** (`PICKUP_CONFIRMED` – Tipo B)
   - **Entrega** (`DELIVERY_CONFIRMED` – Tipo B)
   - **Checagem de evidências** automática (`MILESTONE_VERIFIED`)
7. **Liberação do escrow** (`TRANCHE_RELEASED` / `PAYMENT_RELEASED`)
8. Encerramento (`CONTRACT_CLOSED`)

Regra: **sem escrow, não inicia execução**.

Fluxo 2 — Leilão (livre) + escrow obrigatório após vencedor

1. Leiloeiro credenciado cria leilão e lista lote "spot"
2. Lances livres até fechar

3. Vencedor definido (`AUCTION_CLOSED`, `WINNER_SET`)
4. Gera contrato padrão do leilão e **escrow obrigatório** (`ESCROW_FUNDED`)
5. Depois segue igual: pesagem/embarque/entrega → validação → liberação escrow.

---

2) Contrato parametrizado (template "Frigorífico Carne")

Campos do contrato (para o sistema)

Partes

- `buyerOrgId` (Frigorífico)
- `sellerOrgId` (Fornecedor/Produtor/Cooperativa)
- `optionalBrokerOrgId` (comissão)
- `logisticsOrgId` (se contratado)

Objeto

- `species` (qualquer animal de produção)
- `batchType` (LOTE/INDIVIDUAL)
- `quantityHeads` e/ou `estimatedKgLive` e/ou `estimatedKgCarcass`
- `pickupLocation` (geo + geofence)
- `deliveryPlant` (geo + geofence)

Método de precificação (suporta os 3)

- `pricingMethod`: `LIVE_WEIGHT` | `DEST_WEIGHT` | `CARCASS_GRADE` | `HYBRID`
- `priceTableRef` (tabela acordada)
- `adjustments[]` (bonificação/penalidade por faixa)
- `tolerances`: variação permitida (peso, idade, padrão)

Requisitos e provas

- `evidencePolicy` (A/B por marco)
- `idTraceabilityRequired` (sim/não; individual/lote)
- `mandatoryDocs[]` (GTA, notas, etc. se aplicável)
- `labRequired` (sim/não; quais exames)
- `disputePolicy` (prazos, critérios, auditor)

Financeiro (obrigatório)

- `escrowRequired = true`
- `escrowAmountRule`: 100% do valor estimado (recomendado)
- `trancheRules` (ver abaixo)
- `splitRules` (produtor/frete/comissão/plataforma)

---

3) Marcos + evidência + liberação do escrow

3.1 Regras gerais de evidência

- **Tipo A**: leitor/ID + timestamp (rotineiro)
- **Tipo B**: foto/vídeo + GPS + timestamp + geofence (crítico)
- Pesagem e entrega/embarque = **sempre crítico** (Tipo B), porque destrava dinheiro.

3.2 Marcos padrão com escrow obrigatório

Marco 0 — Contrato assinado

- Evento: `CONTRACT_SIGNED` (A)

Marco 1 — Escrow financiado (obrigatório)

- Evento: `ESCROW_FUNDED` (A)
- **Bloqueio:** não permite `PICKUP_CONFIRMED` se escrow não estiver funded.

Marco 2 — Pesagem "oficial" (conforme método)

- Se `LIVE_WEIGHT`: `ORIGIN_WEIGHING_RECORDED` (B)
- Se `DEST_WEIGHT`: `DEST_WEIGHING_RECORDED` (B)
- Se `CARCASS_GRADE`: `DEST_WEIGHING_RECORDED` (B) + `CARCASS_CLASSIFICATION_RECORDED` (B)

Marco 3 — Embarque confirmado

- Evento: `PICKUP_CONFIRMED` (B: geofence + mídia)

Marco 4 — Entrega confirmada

- Evento: `DELIVERY_CONFIRMED` (B: geofence + mídia)

Marco 5 — Checagem automática de evidências

- Evento: `MILESTONE_VERIFIED` (A)
- O motor valida: geofence OK + evidência presente + estados ok + hashes ok

Marco 6 — Liberação do escrow (tranches)

Você pode usar 2 modos:

Modo simples (você descreveu):

- Liberação única **após entrega + checagem de evidência**
  - Evento: `PAYMENT_RELEASED` (A)

Modo robusto (recomendado, sem atrito):

- **Tranche 1 (opcional):** após embarque confirmado (ex.: 10–20%)
- **Tranche 2:** após entrega confirmada (ex.: 60–80%)
- **Tranche 3 (acerto):** após pesagem/classificação final (restante)

Mas se você quer "padrão simples", usamos liberação única no final.

---

4) Máquina de estados (execução do contrato)

SIGNED → ESCROW_FUNDED → SCHEDULED → PICKUP_CONFIRMED → DELIVERED_CONFIRMED → (WEIGHED/CLASSIFIED se aplicável) → VERIFIED → PAID → CLOSED
Estados de exceção:

- `PAYMENT_HELD` (falha de evidência/marco)
- `DISPUTED` (qualidade/peso divergente)
- `CANCELLED` (expiração de reserva ou acordo)

---

5) Regras do motor (bloqueios que garantem o seu fluxo)

Bloqueios obrigatórios

1. **Sem escrow funded = sem execução**

- Bloqueia: `PICKUP_CONFIRMED`, `SHIPMENT_START`, `DELIVERY_CONFIRMED`

2. **Sem evidência Tipo B = não valida embarque/entrega**

- Bloqueia progressão do estado e liberação de pagamento

3. **Método de precificação define pesagem oficial**

- Se contrato é `DEST_WEIGHT`, peso da origem não libera pagamento (só registra)

4. **Checagem geofence**

- se fora do perímetro, evento crítico vira inválido e abre correção/retém escrow

5. **Rastreabilidade por ID (se exigida)**

- se `idTraceabilityRequired = true`, bloquear fechamento de embarque/entrega sem leituras mínimas

6. **Disputa automática**

- se divergência > tolerância ou qualidade reprovada → `DISPUTE_OPENED` e escrow retido

---

6) Módulo de Leilão (livre) com escrow após vencedor

Papéis

- `LEILOEIRO_CREDENCIADO`: cria leilão, aprova listagens, encerra rodada
- `VENDEDOR`: lista lote livre (UNCOMMITTED)
- `COMPRADOR`: dá lance

Fluxo

- `AUCTION_CREATED` → `LOT_LISTED` (Tipo B mínimo para evitar fraude)
- `BID_PLACED`...
- `AUCTION_CLOSED` → `WINNER_SET`
- `CONTRACT_GENERATED` (A)
- **`ESCROW_FUNDED` obrigatório** pelo vencedor
- segue execução igual.

---

7) Telas (MVP) — direto ao ponto

Frigorífico

- Criar Demanda (com filtros + padrão de qualidade)
- Buscar/filtrar capacidades
- Negociar termos (proposta/contraproposta)
- Assinar contrato
- Depositar escrow
- Acompanhar execução (mapa + status + evidências)
- Validar recebimento/peso/classificação
- Liberar automaticamente (sistema libera, frigorífico acompanha)

Fornecedor

- Publicar capacidade
- Receber demanda e negociar
- Aceitar contrato
- Checklist de execução (pesagem, embarque, docs)
- Enviar evidências (guiado)
- Acompanhar escrow (retido → liberado)

Leiloeiro

- Criar leilão

- Aprovar listagens
- Encerrar e gerar contrato

Auditor (read-only)

- Trilhas e parecer (disputas)

---

8) APIs principais (essenciais)

(Resumo do que existe — eu detalho payload se você quiser)

- `POST /capacity-offers`
- `POST /buyer-demands`
- `POST /matching/reserve`
- `POST /contracts/draft`
- `POST /contracts/{id}/sign`
- `POST /escrow/{id}/fund`
- `POST /events/weighing`
- `POST /events/pickup-confirmed`
- `POST /events/delivery-confirmed`
- `POST /milestones/verify`
- `POST /payments/release` (na prática, disparado pelo motor)

Leilão:

- `POST /auctions`
- `POST /auctions/{id}/lots`
- `POST /auctions/{id}/bids`
- `POST /auctions/{id}/close` → gera contrato → escrow

Próximo passo (para fechar 100% e começar build)

Eu consigo te entregar agora o **"Contrato Frigorífico" em formato de campos + regras** e o **modelo de tolerâncias** (peso/qualidade) para as 3 modalidades, já com:

- como calcular valor estimado para escrow,
- como fazer acerto final (principalmente em CARCASS_GRADE e HYBRID),
- e como tratar divergência de pesagem (origem vs destino).

Só me diga: o escrow será pago via qual trilha (ex.: **PIX**, **boleto**, **transferência**) e se vocês querem **split automático** já no MVP (transportadora/comissão) ou só pagar o fornecedor primeiro e split depois.

Perfeito — isso vira um **Contrato de Produção Sob Demanda com Dupla Garantia (buyer + seller)**, onde:

- o **comprador** dá garantia para "sustentar" o compromisso ao longo de 6+ meses (evita calote/risco de preço),
- o **fornecedor** também dá garantia de performance/entrega (evita "sumir" ou entregar fora do padrão),
- o **pagamento é condicionado por etapas** (milestones),
- e o sistema mantém **duas travas** (custódias lógicas) e só libera as duas ao encerrar a operação.

Abaixo está o desenho técnico completo (Ciclo+) para implementar isso com **escrow + garantias com penhora lógica/execução por gatilho**.

---

1) Estrutura do negócio: 3 "contas" lógicas e 2 travas

A) Escrow de Pagamento (do comprador)

- `PaymentEscrow` recebe o valor estimado do contrato (ou por tranches programadas).
- Serve para garantir que existe dinheiro reservado para pagar conforme marcos.

B) Garantia do Comprador (Buyer Guarantee)

- `BuyerGuaranteeHold` = caução/garantia que pode ser:
  - **depósito adicional** em escrow,
  - **recebível/garantia financeira** (documentada),
  - **garantia real/documental** (tratada como "colateral registrado").
- Objetivo: cobrir multa/indenização se o comprador quebrar o contrato (ex.: desistir sem motivo).

C) Garantia do Fornecedor (Seller Performance Guarantee)

- `SellerGuaranteeHold` = garantia de entrega/qualidade:
  - depósito/caução,
  - retenção de parte do pagamento (holdback),
  - ou colateral documentado.
- Objetivo: cobrir perdas se o fornecedor não cumprir marcos (não entrega / entrega fora do padrão).

No sistema, os três viram "custódias lógicas" (subcontas/holds), auditadas e travadas por estado.

---

2) Contrato por etapas (milestones) para 6+ meses

Você cria um **Plano de Etapas** com:

- `milestoneId`, `description`

- `dueWindow`
- `requiredEvidencePolicy` (A/B e laudo quando aplicável)
- `acceptanceCriteria` (peso/qualidade/tolerâncias, sanidade, docs)
- `paymentTranche` (quanto libera do PaymentEscrow)
- `penaltyRules` (o que acontece se falhar)
- `disputeWindow` (prazo para contestar)

Exemplo de etapas (produção animal)

1. **Contrato assinado + garantias travadas**
2. **Plano técnico aprovado** (assistência técnica / conformidade)
3. **Checkpoint 1** (visita técnica / evidência de evolução)
4. **Checkpoint 2** (sanidade + conformidade + evidência)
5. **Agendamento de embarque**
6. **Embarque confirmado (Tipo B)**
7. **Entrega confirmada (Tipo B)**
8. **Pesagem/classificação/laudos (Tipo B + laudo)**
9. **Liquidação final + encerramento**

Você pode ter milestones "mensais" e os finais (embarque/entrega/peso) como marcos que destravam o grosso do dinheiro.

---

3) "Penhora de garantias" no Ciclo+ (como modelar tecnicamente)

Importante: eu não vou te orientar a "driblar" lei; o que dá para fazer com segurança é **execução contratual automatizada de garantias** com trilha de evidências + auditoria + disputa. Em termos de produto, isso funciona como:

- **penhora lógica/execução de garantia** = "tomar" (transferir) valor da garantia travada para cobrir inadimplemento, **somente se gatilhos objetivos ocorrerem** e com registro auditável.

3.1 Gatilhos objetivos (exemplos)

Falha do comprador

- não deposita/trancha no prazo (`ESCROW_UNFUNDED_AFTER_DEADLINE`)
- cancela sem justificativa contratual
- recusa recebimento sem base em critérios/qualidade definidos

Ação do sistema

- abre `BUYER_DEFAULT_EVENT`
- retém pagamento/valores e pode **executar BuyerGuaranteeHold** conforme contrato:

    - transfere parte/total ao fornecedor (indenização)
    - ou cobre custos logísticos/produção

Falha do fornecedor

- não cumpre marco no prazo (produção não alcança requisito mínimo)
- embarque não ocorre
- entrega fora de tolerância/qualidade, ou com sanidade/documentos inválidos

Ação do sistema

- abre `SELLER_DEFAULT_EVENT`
- executa `SellerGuaranteeHold`:
    - devolução ao comprador (indenização)
    - ou cobertura de custos adicionais (substituição/compra emergencial)

3.2 Disputa e auditor (muito recomendado)

Antes de executar garantia automaticamente em casos "cinza", você define:

- janela de contestação (`disputeWindow`)
- auditor read-only emite parecer (`AUDITOR_OPINION_ISSUED`)
- só então o motor executa.

Isso torna o sistema defensável e reduz litigiosidade.

---

4) Liberação "das duas travas" no final (o que você pediu)

No encerramento, quando:

- todos os marcos obrigatórios estão `VERIFIED`,
- não há disputa aberta,
- e a liquidação final foi feita,

o sistema dispara:

- `RELEASE_BUYER_GUARANTEE` (devolve/libera garantia do comprador)
- `RELEASE_SELLER_GUARANTEE` (devolve/libera garantia do fornecedor)
- `CONTRACT_CLOSED`

Se houver holdbacks (retenções), elas também são liberadas.

---

5) Máquina de estados do contrato com dupla garantia

DRAFT
→ SIGNED
→ **GUARANTEES_LOCKED** (buyer + seller)
→ **EXECUTION_IN_PROGRESS** (milestones ativos)
→ DELIVERY_PHASE (embarque/entrega/pesagem)
→ SETTLEMENT_READY
→ PAID
→ **GUARANTEES_RELEASED**
→ CLOSED

Estados de exceção:

- DISPUTED
- BUYER_DEFAULT / SELLER_DEFAULT
- GUARANTEE_EXECUTED
- TERMINATED

---

6) Evidências e lastro para "etapas" (muito importante em 6 meses)

Para milestones longos, a prova não é só "foto": você usa uma combinação:

- **Visita técnica** (Tipo B: geo + mídia + checklist + assinatura/atestado)
- **Drones** (Tipo B: missão + ortomosaico + metadados)
- **Leitor animal** (Tipo A: inventário do lote ao longo do tempo)
- **Pesagens periódicas** (Tipo B quando marco de pagamento)
- **Laboratório** (CoC: coleta/lacre/resultado assinado)

Tudo encadeado em auditoria imutável.

---

7) Como calcular "quanto" de garantia (regra prática para produto)

Você parametriza no contrato:

- `buyerGuaranteePercent` (ex.: 10–30% do valor estimado, ou custo de oportunidade)
- `sellerGuaranteePercent` (ex.: 5–20%, ou equivalente ao risco do lote)
- `holdbackPercent` (parte do pagamento retida até validação final)

E deixa flexível por perfil de risco (score histórico).

---

8) Implementação técnica (objetos que você precisa adicionar)

Além do que já tínhamos, inclua:

- `GuaranteeHold`:
  - `holdId`, `orgId`, `contractId`, `party` (BUYER/SELLER)
  - `type` (CASH_ESCROW / RECEIVABLE / REAL_ASSET_DOC / HOLD_BACK)
  - `amount`, `status` (LOCKED/RELEASED/EXECUTED)
  - `executionRules[]` (gatilhos e destinos)
- `MilestonePlan`:
  - `milestoneId`, `contractId`, `dueWindow`, `criteria`, `evidencePolicy`, `trancheAmount`, `penaltyRules`
- `DefaultEvent`:
  - `defaultId`, `contractId`, `partyAtFault`, `reasonCode`, `evidenceRef`, `resolution`

Próximo passo (eu já posso entregar o template pronto)

Para deixar isso "pronto para time jurídico + dev", eu consigo te devolver:

1. **Template de contrato** (campos + cláusulas parametrizadas):
   - garantia do comprador, garantia do fornecedor, milestones, disputa, execução de garantia, liberação final
2. **Plano padrão de milestones (6 meses)** com 6–10 etapas e o tipo de evidência de cada uma
3. **Regras do motor** (gatilhos objetivos) e eventos

Só preciso que você escolha o formato de garantia mais comum para cada parte no MVP:

- Comprador: **(a) depósito em dinheiro**, **(b) recebível/garantia financeira**, ou **(c) ambos**
- Fornecedor: **(a) retenção/holdback do pagamento**, **(b) depósito**, **(c) ambos**

Abaixo está um modelo **completo e parametrizável** de **contratos por ciclo de produção** (Ciclo+) para: **bovinos (cria, recria, engorda, reprodução)**, **ovinos/caprinos**, **peixes**, **aves** e **plantações** — incluindo a possibilidade de o comprador **comprar antes de nascer/antes de produzir** e acompanhar o desenvolvimento com **marcos + evidências + auditoria + escrow + dupla garantia**.

A ideia é: **um "Contrato-Mestre" único** + **Anexos de Ciclo** (um por cadeia/produto) + **Plano de Marcos** configurável.

---

1) Contrato-Mestre Ciclo+ (serve para todos os ciclos)

1.1 Partes e papéis

- **Comprador (Indústria/Integrador/Cooperativa)**
- **Produtor/Fornecedor**
- **Técnico/Certificador** (opcional, mas recomendado)
- **Transportador** (opcional)
- **Auditor** (read-only, para disputa/compliance)

1.2 Objeto contratual (parametrizado)

- `cadeia`: ANIMAIS | AQUICULTURA | AVICULTURA | CULTURAS
- `produto`: espécie/cultura + padrão
- `unidade`: cabeça / kg vivo / kg carcaça / kg pescado / dúzia / tonelada / hectare etc.
- `janelaEntrega`
- `destino` (planta/unidade)

1.3 Escrow obrigatório + Dupla Garantia

- **PaymentEscrow (comprador)**: valor retido para pagamento por marcos.
- **BuyerGuaranteeHold**: garantia do comprador (sustenta contrato longo).
- **SellerGuaranteeHold**: garantia de performance do produtor (entrega/qualidade).
- **Liberação final das duas travas** só quando o ciclo encerrar sem pendências.

1.4 Plano de Marcos (MilestonePlan)

Cada contrato define:

- marcos (etapas), prazos e critérios de aceite;
- evidência exigida (Tipo A/B e laudos);
- tranche do escrow a liberar;
- regras de penalidade/execução de garantias;
- janela de disputa + auditor read-only.

---

2) Como permitir “comprar antes de nascer / antes de produzir”

Isso é um **Contrato a Termo / Produção Sob Demanda** com **identificação antecipada do “lote futuro”**:

2.1 Unidade contratada “futura”

- Você cria um **ProductionLotFuture**:
  - `loteFuturoId`
  - `fazenda/unidade`

- `capacidade` (ex.: 200 bezerros esperados; 50k pintos; 30 toneladas tilápia; 100 ha soja)
- `janela de entrega`
- `padrão mínimo`

2.2 Acompanhamento obrigatório por marcos

O comprador "vai pagando" por evolução, e o produtor prova o avanço:

- marcos técnicos (gestação, nascimento, taxa de sobrevivência, sanidade, biometria/peso, manejo, etc.)
- evidências (visitas técnicas, leituras, mídia, laboratório, drone/telemetria em culturas)
- tudo auditado e travado no escrow/garantias

---

3) Modelos por ciclo (Anexos de Ciclo)

Abaixo eu listo para cada cadeia: **(i) marcos**, **(ii) critérios**, **(iii) evidências**, **(iv) tranches sugeridas**, **(v) gatilhos de garantia**.

A) Bovinos – CRIA (compra antes de nascer)

Objetivo: comprar "bezerro futuro" (ou produção de bezerros) com acompanhamento.

Marcos sugeridos

1. **Contrato + garantias travadas + escrow funded**
2. **Comprovação de matrizes e touros/IA** (inventário do rebanho base)
3. **Diagnóstico de prenhez** (confirmado por técnico)
4. **Acompanhamento trimestral** (condição corporal, sanidade, mortalidade)
5. **Nascimento / identificação** (bezerros)
6. **Desmame** (peso mínimo e sanidade)
7. **Entrega/embarque**
8. **Recebimento/checagem**
9. **Liquidação final + liberação das garantias**

Evidências

- Prenhez e marcos sanitários: **Tipo B** (mídia + geo + laudo/atesto técnico)
- Identificação (RFID/QR) e inventários: **Tipo A**
- Embarque/entrega: **Tipo B** (geofence)

Tranches (exemplo)

- 10% após prenhez confirmada
- 20% após nascimento + identificação
- 30% após desmame e peso mínimo

- 40% após entrega/checagem final

Gatilhos de garantia

- Se taxa de prenhez/nascimento ficar abaixo do mínimo acordado → ajuste automático (reposições ou execução parcial da garantia)
- Se mortalidade acima de limite sem justificativa registrada → execução da garantia do produtor

---

B) Bovinos – RECRIA

Objetivo: comprar/financiar recria com entrega em padrão de peso/idade.

Marcos

1. Lote inicial identificado (A) + baseline de peso (B se marco de pagamento)
2. Checkpoints mensais (peso médio, sanidade, manejo)
3. Padrão de "pronto para engorda" (peso/idade)
4. Embarque/entrega
5. Pesagem no destino (se método for DEST_WEIGHT)
6. Liquidação final

Evidências

- Pesagem que libera tranche: **Tipo B**
- Inventários rotineiros: **Tipo A**
- Entrega: **Tipo B**

C) Bovinos – ENGORDA / CONFINAMENTO

Objetivo: comprar "boi futuro pronto" ou contratar engorda para entrega com padrão (peso/carcaça).

Marcos

1. Entrada do lote no confinamento (A + B se necessário)
2. Dieta e protocolos (registros)
3. Pesagens periódicas (B em marcos)
4. Pré-abate (sanidade/carência ok)
5. Entrega ao frigorífico + pesagem/classificação (B)
6. Acerto por carcaça/grade (se aplicável)
7. Liquidação + liberação garantias

Evidências fortes

- Carência sanitária e documentos: obrigatório para "liberar abate"

- Pesagem/classificação: **Tipo B + laudo**

D) Bovinos – REPRODUÇÃO (venda de genética: prenhez/IA/embrião)

Objetivo: comprar resultado reprodutivo (prenhez confirmada / bezerro de genética).

Marcos

1. Protocolo reprodutivo executado (IA/TE)
2. Confirmação de prenhez (laudo)
3. Acompanhamento até parto
4. Nascimento/registro genético (quando aplicável)
5. Entrega (se for venda do animal) ou encerramento (se for prestação do serviço)

Evidências

- Sempre **Tipo B + laudo** para prenhez e genética.

E) Ovinos/Caprinos (lote por fase)

Marcos

1. Lote inicial + sanidade (A + B em marcos críticos)
2. Crescimento (checkpoints)
3. Pronto para abate (peso e condição)
4. Embarque/entrega + pesagem
5. Liquidação

Ajustes do perfil

- mais foco em lote (grupo), mortalidade e sanidade.

F) Peixes (aquicultura – ciclo por tanque/lote)

Objetivo: comprar produção futura (ex.: tilápia) com acompanhamento por biomassa e qualidade.

Marcos

1. Povoamento do tanque (quantidade e origem)
2. Checkpoints de biomassa (amostragem)
3. Parâmetros de água dentro de faixa (sensores/lab)
4. Classificação por tamanho (pré-colheita)
5. Despesca/colheita + pesagem (B)

6. Entrega/recebimento (B)
7. Laudos (se exigidos)
8. Liquidação final

Evidências

- Biomassa/pesagens: **Tipo B**
- Qualidade da água: sensores (A) + laudos (A com documento assinado)
- Colheita/entrega: **Tipo B + geofence**

Gatilhos de garantia

- Mortalidade em massa / quebra de parâmetros sem plano corretivo → execução parcial da garantia do produtor.

G) Aves (lote por ciclo de galpão)

Objetivo: produção sob demanda por lote (frango) ou por produto (ovos).

G.1 Frango de corte

Marcos

1. Alojamento do lote (nº pintos)
2. Checkpoints (peso médio, conversão, mortalidade)
3. Retirada/apanha + carregamento (B)
4. Entrega + pesagem (B)
5. Liquidação

G.2 Ovos

Marcos

1. Entrada em postura / baseline
2. Entregas semanais (marcos recorrentes)
3. Controle de qualidade (amostragens)
4. Liquidação por janela

Evidências

- Lote e checklists: A
- Retirada/entrega/pesagem: B
- Qualidade: laudo/inspeção quando exigido

H) Plantações (culturas – compra antes de colher)

Objetivo: a indústria compra "safra futura" com provas de execução e risco controlado.

Marcos

1. Planejamento de safra (talhões, área, variedade)
2. Plantio confirmado (telemetria trator/dados + evidência)
3. Tratos (aplicações) – checklists + receituário (se aplicável)
4. Monitoramento (drone/NDVI/inspeções)
5. Pré-colheita (estimativa e qualidade)
6. Colheita (evidência + volume)
7. Armazenagem/classificação/umidade (laudos)
8. Entrega/expedição + pesagem
9. Liquidação final

Evidências

- Plantio/aplicação/colheita: **telemetria** + **Tipo B** quando marco financeiro
- Umidade/classificação: **laudo** (A com doc assinado) + fotos (B)
- Entrega: **Tipo B + geofence**

---

4) Regras de "penhora/execução" e liberação dupla no final

4.1 Execução de garantias (gatilhos objetivos)

- **Comprador inadimplente** (não funda escrow ou não cumpre tranche no prazo):
  - executa BuyerGuaranteeHold conforme contrato (indeniza fornecedor/custos)
- **Produtor inadimplente** (não cumpre marcos, qualidade fora de tolerância, mortalidade acima do limite sem justificativa):
  - executa SellerGuaranteeHold (indeniza comprador/custos)

4.2 Liberação final "das duas travas"

Somente quando:

- todos os marcos obrigatórios = `VERIFIED`,
- pagamento final = concluído,
- nenhuma disputa aberta,
- auditoria íntegra,

→ `RELEASE_BUYER_GUARANTEE` e `RELEASE_SELLER_GUARANTEE`.

---

5) O que você ganha com esse modelo

- Dá para **comprar antes de nascer** (cria/reprodução) e **antes de colher** (culturas), sem virar "aposta", porque o risco fica controlado por:
  - escrow por etapas,
  - evidência obrigatória,
  - acompanhamento técnico,
  - garantias bilaterais executáveis por gatilhos.

---

6) Próximo passo prático

Eu posso agora transformar isso em **templates prontos** (modelo de contrato) com campos e anexos para o sistema:

1. **Contrato Ciclo+ – Produção Sob Demanda (Master)**
2. **Anexo Bovinos (Cria/Recria/Engorda/Reprodução)**
3. **Anexo Ovinos/Caprinos**
4. **Anexo Aquicultura**
5. **Anexo Aves (corte + ovos)**
6. **Anexo Culturas (grãos/hortifruti etc.)**
7. **Tabela padrão de tranches** e **tolerâncias** por cadeia (configurável)

Se você me disser quais cadeias vêm primeiro no MVP (ex.: **bovinos + aves + grãos**), eu já priorizo os anexos e deixo as etapas exatamente no seu padrão de operação.

Sim — vou **desenvolver no meu modelo** (Ciclo+) e te entregar uma **estrutura contratual pronta para virar sistema**, com:

1. **Contrato-Mestre Ciclo+ – Produção Sob Demanda (com escrow obrigatório + dupla garantia)**
2. **Anexos por ciclo**: Bovinos (cria, recria, engorda, reprodução), Ovinos/Caprinos, Aves, Peixes, Plantações
3. **Planos de marcos (milestones)** + **evidências (A/B/laudos)** + **tranches do escrow** + **gatilhos de execução de garantias**
4. **Regras operacionais** para o motor (bloqueios e liberações)

---

1) CONTRATO-MESTRE CICLO+ (Produção Sob Demanda)

1.1 Identificação

- **Partes**: Comprador (Indústria) / Fornecedor (Produtor/Cooperativa) / Técnico (opcional) / Transportador (opcional)
- **Objeto**: compra futura / produção sob demanda de "Produto Agro" (animal/cultura) com entrega em janela e padrão definido.
- **Prazo**: ciclo estimado + tolerância (ex.: até 6–12 meses conforme anexo).

1.2 Definições (essenciais para sistema)

- **Unidade Contratada**: individual, lote, safra/talhão, tanque/galpão.
- **Plano de Marcos (MilestonePlan)**: conjunto de etapas com critérios e evidências.
- **Evidência Tipo A**: identificador/leitura + timestamp.
- **Evidência Tipo B**: mídia + GPS/geofence + timestamp (+ laudo quando exigido).
- **Aceite**: validação formal de marco pelo motor e/ou comprador (com janela de contestação).
- **Disputa**: evento formal que congela liberações até resolução.

1.3 Escrow obrigatório (Pagamento)

Cláusula

- O Comprador **é obrigado** a **fundar** o `PaymentEscrow` (integral ou por cronograma) como condição de início do ciclo.
- Sem `ESCROW_FUNDED`, o sistema **bloqueia** execução de marcos que gerem custo/compromisso (ex.: início de fase, agendamento de coleta, embarque).

Regra de cálculo

- `EscrowBase` = valor estimado do contrato (com fórmula do anexo)
- `EscrowFundingSchedule` = integral (no dia 0) **ou** por fases (pré-definidas no anexo).

1.4 Dupla Garantia (Buyer + Seller)

Garantia do Comprador (BuyerGuaranteeHold)

Finalidade: garantir sustentação do contrato longo (6+ meses) e indenizar o fornecedor se houver desistência/ruptura indevida.

Garantia do Fornecedor (SellerGuaranteeHold)

Finalidade: garantir performance/entrega/qualidade e indenizar o comprador em caso de inadimplemento relevante.

Importante (modelo do sistema):

- As garantias ficam em **custódia lógica travada** (`LOCKED`) durante o ciclo.
- Ao final, com o contrato encerrado sem pendência, o sistema libera as duas (`RELEASED`).
- Se ocorrer inadimplemento e gatilho objetivo, o sistema pode **executar** (`EXECUTED`) a garantia conforme regras.

1.5 Preço e modalidades (as 3 formas)

O contrato aceita:

- **LIVE_WEIGHT** (peso vivo na origem)
- **DEST_WEIGHT** (peso no destino)
- **CARCASS_GRADE** (carcaça/classificação)
- **HYBRID** (combinações)

Regra do sistema: o método escolhido define:

- qual pesagem é "oficial",
- quais laudos são obrigatórios,
- quais marcos liberam pagamento.

1.6 Plano de Marcos e Tranches (pagamento por etapas)

- Cada marco tem: `criteria`, `evidencePolicy`, `dueWindow`, `tranchePercent`, `disputeWindow`.
- O motor faz `MILESTONE_VERIFIED`; só então libera tranche.

1.7 Penalidades, execução de garantias e "travas"

Inadimplemento do Comprador (exemplos objetivos)

- não fundar escrow no prazo,
- romper sem justificativa contratual,
- recusar recebimento sem base nos critérios de aceite.

Ação do sistema: `BUYER_DEFAULT_EVENT` → pode executar `BuyerGuaranteeHold` e/ou indenizar custos previstos.

Inadimplemento do Fornecedor (exemplos objetivos)

- falha em marcos críticos (nascimento mínimo, sobrevivência mínima, peso mínimo, sanidade/documentos),
- entrega fora de tolerância sem correção,
- ausência de evidência obrigatória.

Ação do sistema: `SELLER_DEFAULT_EVENT` → pode executar `SellerGuaranteeHold` e/ou reter pagamentos (holdback).

1.8 Disputa e auditor (read-only)

- Abre `DISPUTE_OPENED` quando critérios divergirem ou evidência for contestada.
- Auditor (read-only) emite parecer `AUDITOR_OPINION_ISSUED`.
- Resolução: `DISPUTE_RESOLVED` → libera/ajusta/estorna conforme contrato.

1.9 Encerramento e liberação das duas travas

Condições:

- marcos obrigatórios `VERIFIED`,
- pagamento final liberado,
- sem disputas abertas,
- evidências e auditoria íntegros.

Ações automáticas:

- `RELEASE_BUYER_GUARANTEE`
- `RELEASE_SELLER_GUARANTEE`
- `CONTRACT_CLOSED`

---

2) ANEXOS POR CICLO (planos de marcos prontos)

A seguir, cada anexo já vem com: **marcos**, **evidência**, **tranches sugeridas**, **gatilhos de garantia**.
(Tudo parametrizável no sistema.)

ANEXO A — BOVINOS: CRIA (compra antes de nascer)

Objeto

Compra futura de **bezerros** (lote futuro) com entrega pós-desmame ou em fase definida.

Marcos (modelo padrão)

M0) Contrato assinado + escrow + garantias travadas
M1) Inventário base de matrizes (A) + visita técnica de validação (B)
M2) Prenhez confirmada (B + laudo/atesto)
M3) Checkpoint gestação (B)
M4) Nascimento + identificação (A para IDs, B para validação por amostra)
M5) Desmame + peso mínimo (B)
M6) Embarque (B geofence)
M7) Entrega/recebimento (B geofence)
M8) Liquidação final e encerramento

Tranches sugeridas (exemplo)

- 10% após M2 (prenhez confirmada)
- 20% após M4 (nascimento + identificação)
- 30% após M5 (desmame + peso mínimo)
- 40% após M7 (entrega confirmada)

Gatilhos objetivos

- taxa mínima de prenhez/nascimento/sobrevivência abaixo do pactuado → ajuste (reposições) ou execução parcial de garantias.
- sanidade/carência irregular bloqueia M6/M7.

ANEXO B — BOVINOS: RECRIA

Objeto

Entrega futura de lote “pronto para engorda” (peso/idade).

Marcos

M0 escrow + garantias
M1 lote inicial identificado (A) + baseline de peso (B se tranche)
M2 checkpoints mensais (A para inventário, B para pesagem de marco)
M3 “pronto para engorda” (B – peso mínimo)
M4 embarque (B)
M5 entrega (B)
M6 acerto (se DEST_WEIGHT) e fechamento

Tranches

- 20% M1/M2 inicial
- 30% M3
- 50% M5/M6

ANEXO C — BOVINOS: ENGORDA/CONFINAMENTO

Objeto

Entrega futura de lote para abate com padrão (peso no destino ou carcaça).

Marcos

M0 escrow + garantias
M1 entrada do lote (A + evidência B por amostra)
M2 protocolo alimentar/sanitário (A) + checkpoint (B se tranche)
M3 pré-abate: carência ok + conformidade (B)
M4 embarque (B)
M5 entrega (B)
M6 pesagem destino (B)
M7 classificação carcaça (B + relatório/laudo) quando aplicável
M8 liquidação e liberação de garantias

Tranches (robusto)

- 20% M1
- 20% M3
- 60% M6/M7

ANEXO D — BOVINOS: REPRODUÇÃO (prenhez / genética)

Objeto

Resultado reprodutivo (prenhez confirmada / bezerro de genética / serviço reprodutivo).

Marcos

M0 escrow + garantias
M1 execução do protocolo (B)
M2 confirmação prenhez (B + laudo)
M3 acompanhamento (B)
M4 parto e registro (B + identificação A) quando aplicável
M5 entrega ou encerramento

Tranches

- 30% M1
- 40% M2
- 30% M4/M5

ANEXO E — OVINOS/CAPRINOS (lote)

Marcos

M0 escrow + garantias
M1 lote base + sanidade (B)
M2 checkpoints (A + B em marcos)
M3 pronto para abate (B)
M4 embarque (B)
M5 entrega + pesagem (B)
M6 liquidação

ANEXO F — AVES (corte e ovos)

Frango de corte (lote/galpão)

M0 escrow + garantias
M1 alojamento do lote (A + B por amostra)
M2 checkpoints (mortalidade, peso médio) (A; B quando tranche)
M3 apanha/carregamento (B)
M4 entrega + pesagem (B)
M5 liquidação

Ovos (ciclo recorrente)

Contrato com "janelas semanais":

- marcos recorrentes de entrega (B no recebimento) + laudos periódicos (A doc assinado)
- tranches por janela

ANEXO G — PEIXES (aquicultura: tanque/lote)

Marcos

M0 escrow + garantias
M1 povoamento (A + B)
M2 checkpoints biomassa (B)
M3 qualidade água (A sensor + A laudo quando exigido)
M4 pré-colheita/classificação (B)
M5 despesca/colheita + pesagem (B)
M6 entrega (B)
M7 liquidação

ANEXO H — PLANTAÇÕES (safra futura)

Marcos

M0 escrow + garantias
M1 planejamento (talhões, área, variedade) (A)
M2 plantio confirmado (telemetria trator + B por amostra)
M3 tratos (A logs + B quando tranche)
M4 monitoramento (drone/inspeção) (B)
M5 pré-colheita (B + estimativa)
M6 colheita (B)
M7 classificação/umidade/laudo (A doc assinado + B)
M8 entrega + pesagem (B)
M9 liquidação

---

3) REGRAS DO MOTOR (bloqueios e liberações padrão)

Bloqueios obrigatórios

- **Não inicia ciclo** sem `ESCROW_FUNDED` + garantias travadas.
- **Não libera tranche** sem `MILESTONE_VERIFIED`.
- **Não confirma embarque/entrega** sem evidência Tipo B + geofence.
- **Não valida marco sanitário** sem documento/laudo quando exigido.

Execução de garantias

- gatilhos objetivos + janela de disputa;
- execução automática só para casos "pretos no branco"; casos "cinza" exigem parecer (auditor read-only).

---

4) Como isso vira software (estrutura pronta)

Objetos mínimos no sistema

- `ContractMaster`
- `CycleAnnex` (um por cadeia)
- `MilestonePlan` (instanciado por contrato)
- `PaymentEscrow`
- `GuaranteeHold` (buyer/seller)
- `OperationalEvent` + `Evidence` + `AuditLog`
- `Dispute`

---

5) Próximo passo que eu já faço agora (sem te perguntar mais nada)

Se você quiser, eu continuo e te entrego em seguida:

1. **Templates completos de contrato** (texto + campos) para cada anexo (Bovinos Cria/Recria/Engorda/Reprodução, Aves, Peixes, Plantações)
2. **Tabelas padrão** de:
   - percentuais de escrow e garantias por risco/ciclo,
   - tolerâncias (peso, mortalidade, produtividade, umidade etc.),
   - e janela de disputa por etapa.
3. **Catálogo de eventos** (eventType, payload mínimo, evidência A/B, estado afetado) para todas as cadeias.

Diz só: você quer que o contrato saia com **linguagem mais "jurídica formal"** (para assinar) ou **linguagem "operacional + técnica"** (para embutir no sistema e depois gerar PDF jurídico)?

Perfeito — então vamos para **modelo operacional-técnico**, "contrato como configuração", gerado por **(a) padrões predefinidos** ou **(b) negociação assistida por IA**, com **assinatura digital** e/ou **validação por técnico vinculado**. Abaixo está o blueprint completo para embutir no sistema: **objetos**, **fluxo**, **templates parametrizados**, **motor de negociação/IA**, **assinatura**, **trilhas de auditoria** e **regras de bloqueio**.

---

1) Como o contrato nasce no sistema (geração automática)

1.1 Entradas possíveis

A) Padrão predefinido

- Indústria escolhe um **ContractTemplate** (ex.: "Frigorífico – Abate – Escrow + Dupla Garantia")
- Preenche: volume, janela, qualidade, local, método de preço, garantias, milestones
- Sistema gera um **ContractDraft** com parâmetros.

B) Negociação assistida por IA

- Partes negociam (chat + campos)
- IA sugere: percentuais de garantia, tranches, tolerâncias, método de preço, marcos mínimos, riscos
- Resultado vira um **TermSheet** (folha de termos) versionada.
- Sistema transforma TermSheet em ContractDraft.

C) Técnico vinculado (opcional)

- Técnico revisa ou valida termos críticos (sanidade, carência, laudos, critérios de aceite)
- Ele assina como "validador" (coassinatura) ou atesta campos.

---

2) Objetos (modelagem) para embutir no sistema

2.1 Template e versão

ContractTemplate

- `templateId`, `name`, `category` (ANIMAIS/CULTURAS/…)
- `requiredFields[]` (campos obrigatórios)
- `defaultRulesetId` (regras padrão)
- `defaultMilestonePlanId`
- `defaultPricingConfig`
- `defaultGuaranteeConfig`
- `version`, `effectiveFrom`

ClauseBlock (blocos técnicos)

- ao invés de texto jurídico, você usa blocos como:
  - `ESCROW_REQUIRED`
  - `DUAL_GUARANTEE`
  - `WEIGHING_METHOD`
  - `QUALITY_ACCEPTANCE`
  - `DISPUTE_POLICY`
  - `PENALTY_EXECUTION`
- Cada bloco tem parâmetros e validações.

2.2 TermSheet (resultado da negociação)

TermSheet

- `termSheetId`, `templateId`, `buyerOrgId`, `sellerOrgId`
- `pricingMethod` (LIVE/DEST/CARCASS/HYBRID)
- `volume`, `deliveryWindow`, `pickupGeo`, `deliveryGeo`
- `qualitySpec`, `tolerances`
- `escrowPolicy` (funding schedule)
- `guaranteePolicy` (buyer/seller)
- `milestonePlanOverrides[]`
- `splitRules[]`
- `disputePolicy`
- `techValidationRequired` (true/false)
- `status`: DRAFT → PROPOSED → AGREED → LOCKED

2.3 Contrato executável (o que roda o ciclo)

ExecutableContract

- `contractId`, `termSheetId`, `stateMachineId`
- `escrowAccountId`
- `buyerGuaranteeHoldId`, `sellerGuaranteeHoldId`
- `milestonePlanId` (instância com datas)
- `evidencePolicyId`
- `signatureEnvelopeId`
- `status`: SIGNED → GUARANTEES_LOCKED → ESCROW_FUNDED → IN_EXECUTION → VERIFIED → SETTLED → CLOSED

2.4 Assinatura (digital ou assistida)

SignatureEnvelope

- `envelopeId`, `contractId`
- `signers[]`: buyer, seller, (optional) técnico
- `method`: DIGITAL | ASSISTED (técnico) | BOTH
- `status`: PENDING → SIGNED → REJECTED
- `signedAt`, `signatureRefs[]`

2.5 Eventos, evidências e auditoria (núcleo)

- `OperationalEvent`
- `Evidence` (A/B + mídia + geo + timestamp + laudoRef)
- `AuditLog` (append-only + hash encadeado)
- `MilestoneVerification` (resultado do motor)
- `Escrow` + `Tranche` + `Split`
- `Dispute`

---

3) Fluxo completo (com IA + assinatura + técnico)

Etapa 0 — Início

1. Buyer cria "Intenção" (Demand/Need) ou seleciona "Capacidade" do seller.
2. Sistema seleciona template adequado (por espécie/cultura/ciclo).

Etapa 1 — Negociação (humanos + IA)

- Partes conversam e preenchem campos (não texto livre).
- Cada ajuste gera `TERM_PROPOSED` com diff de parâmetros.

IA (assistente de negociação)

A IA pode:

- sugerir percentuais de garantias por risco,
- sugerir tranches por ciclo,
- sugerir tolerâncias realistas,
- detectar inconsistências (ex.: método carcaça sem marco de classificação),
- sinalizar risco e exigir evidência tipo B / laudo.

Mas: IA nunca "assina"; ela só propõe e registra recomendação auditável:

- `AI_TERM_SUGGESTION` (com modelo/versão + confiança)

Etapa 2 — TermSheet fechado

- Quando ambas as partes aceitam, vira `TermSheet.status = AGREED`
- Sistema valida consistência (motor):
  - escrow obrigatório,
  - dupla garantia configurada,
  - marcos mínimos presentes,
  - evidência por marco definida.

Etapa 3 — Validação técnica (opcional)

Se `techValidationRequired = true`:

- Técnico revisa campos críticos (sanidade, carências, laudos, critérios)
- Técnico atesta (coassina ou "aprova"):
  - `TECH_VALIDATION_APPROVED`

Etapa 4 — Assinatura

- Sistema cria `SignatureEnvelope`
- Buyer e Seller assinam digitalmente
- (se existir) técnico assina como validador
- Ao final: `CONTRACT_SIGNED`

Etapa 5 — Travamento de garantias + escrow

- `GUARANTEES_LOCKED` (buyer + seller)
- `ESCROW_FUNDED` (buyer deposita)
- Sem isso, não entra em execução (bloqueio)

Etapa 6 — Execução por marcos + evidências

- Operador executa marcos, enviando evidência guiada
- Motor valida estado + evidência + geofence + integridade

Etapa 7 — Liquidação automática e encerramento

- `MILESTONE_VERIFIED`
- `TRANCHE_RELEASED` / `PAYMENT_RELEASED`
- No final: libera as duas garantias
- `CONTRACT_CLOSED`

---

4) Templates operacionais (exemplos prontos para o sistema)

Abaixo estão os **templates** como "blocos técnicos" com parâmetros.

Template T1 — Produção sob demanda (animal) com compra antes de nascer

Aplicação: bovinos cria / reprodução; ovinos; aves (lote futuro); peixes (biomassa futura)

Campos

- `productionCycleType`: CRIA | RECRIA | ENGORDA | REPRODUCAO | LOTE_AVES | TANQUE_PEIXE
- `futureLotDefinition`: capacidade, janela, origem
- `milestonesPreset`: M0..Mn (por anexo)
- `escrowFunding`: FULL (obrigatório no início) ou SCHEDULED (por fases)
- `buyerGuaranteePercent`, `sellerGuaranteePercent`

- `disputeWindowDays`

Validações

- exige marco de “confirmação técnica” (prenhez / povoamento / alojamento)
- exige evidência Tipo B em marcos críticos e geofence em embarque/entrega

Template T2 — Abate frigorífico (compra programada)

Aplicação: contratos com embarque/entrega/peso destino/carcaça
Campos

- `pricingMethod`: LIVE/DEST/CARCASS/HYBRID
- `pricingTableRef`
- `qualitySpec` + `tolerances`
- `mandatoryWeighing`: ORIGIN / DEST / BOTH
- `carcassClassificationRequired`: boolean
- `escrow`: FULL
- `releaseMode`: FINAL_ONLY (seu padrão) ou TRANCHE

Validações

- se CARCASS/HYBRID: exige marco de classificação
- se DEST: exige pesagem destino
- sempre exige `PICKUP_CONFIRMED` e `DELIVERY_CONFIRMED` como Tipo B + geofence

Template T3 — Safra futura (plantações)

Campos

- `areaHa`, `cropVariety`, `harvestWindow`
- `opsRequired`: PLANTIO, TRATOS, COLHEITA, CLASSIFICACAO, ENTREGA
- `evidenceSources`: DRONE, TELEMETRIA, LAB
- `escrow`: FULL ou por fase (mas obrigatório antes de iniciar compromissos)
- garantias e disputas

---

5) Regras de bloqueio (motor) — obrigatórias

1. **Contrato só nasce se TermSheet consistente** (sem campos faltando).
2. **Sem escrow funded, bloqueia execução**.
3. **Sem evidência Tipo B em embarque/entrega, bloqueia verificação**.
4. **Sem marco exigido pelo método de preço, bloqueia liquidação**.
5. **Se técnico obrigatório, sem validação técnica bloqueia assinatura**.
6. **Se disputa aberta, congela tranches e garantias**.

---

6) Catálogo mínimo de eventos para negociação/contrato (MVP)

- `TERM_PROPOSED`, `TERM_ACCEPTED`, `TERM_REJECTED`
- `AI_TERM_SUGGESTION`
- `TECH_VALIDATION_REQUESTED`, `TECH_VALIDATION_APPROVED/REJECTED`
- `CONTRACT_DRAFTED`, `CONTRACT_SIGNED`
- `GUARANTEES_LOCKED`, `ESCROW_FUNDED`
- `MILESTONE_SUBMITTED`, `MILESTONE_VERIFIED`, `MILESTONE_REJECTED`
- `PAYMENT_RELEASED`, `GUARANTEE_RELEASED`, `GUARANTEE_EXECUTED`
- `DISPUTE_OPENED`, `AUDITOR_OPINION_ISSUED`, `DISPUTE_RESOLVED`

---

7) Como isso vira UI (bem direto)

Tela "Negociação" (não texto livre)

- Colunas: **Preço**, **Método**, **Qualidade**, **Janela**, **Garantias**, **Milestones**, **Liberação**
- Cada alteração = proposta (diff) + chat contextual
- Botão: "Aceitar termos" → gera TermSheet

Tela "Contrato"

- Mostra TermSheet + milestones + status escrow/garantias
- Botão: "Assinar"

Tela "Execução"

- Checklist do próximo marco
- Captura evidência guiada (A/B)
- Status do escrow (retido / pronto / liberado)

Próximo passo (eu continuo já)

Agora que definimos "contrato operacional gerado por negociação/IA", eu consigo te entregar imediatamente os **templates completos por cadeia** em formato de **JSON de configuração** (para o seu time implementar), por exemplo:

- `template_frigo_abate_v1.json`
- `template_bovinos_cria_v1.json`
- `template_peixes_tanque_v1.json`
- `template_aves_lote_v1.json`
- `template_safra_futura_v1.json`

Cada um com:

- campos,
- validações,
- plano de marcos,
- matriz de evidência,
- tranches,
- regras de garantia e disputa.

Quer que eu escreva esses templates em **JSON** (pronto para dev) ou em **tabela/markdown** (mais fácil de ler e revisar)?

1) `contract_template.schema.json` (Schema do Template)

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id":
"https://cicloplus.local/schemas/contract_template.schema.json",
  "title": "Ciclo+ Contract Template",
  "type": "object",
  "required": ["templateId", "name", "version", "category", "parties",
"termFields", "policies", "milestones", "stateMachine"],
  "additionalProperties": false,
  "properties": {
    "templateId": { "type": "string", "minLength": 3, "maxLength": 80
},
    "name": { "type": "string", "minLength": 3, "maxLength": 160 },
    "version": { "type": "string", "pattern": "^v\\d+(\\.\\d+){0,2}$"
},
    "category": { "type": "string", "enum": ["ANIMAIS", "AVICULTURA",
"AQUICULTURA", "CULTURAS", "SERVICOS", "LEILAO"] },
    "description": { "type": "string", "maxLength": 2000 },
    "effectiveFrom": { "type": "string", "format": "date" },
    "defaults": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "pricingMethod": { "type": "string", "enum": ["LIVE_WEIGHT",
"DEST_WEIGHT", "CARCASS_GRADE", "HYBRID", "FIXED"] },
        "escrowFundingMode": { "type": "string", "enum":
["FULL_UPFRONT", "SCHEDULED"] },
        "releaseMode": { "type": "string", "enum": ["FINAL_ONLY",
"TRANCHE"] }
      }
    },
    "parties": {
      "type": "array",
      "minItems": 2,
      "items": {
        "type": "object",
        "required": ["role", "required"],
        "additionalProperties": false,
        "properties": {
          "role": { "type": "string", "enum": ["BUYER", "SELLER",
"TECHNICIAN", "CARRIER", "AUCTIONEER", "AUDITOR"] },
          "required": { "type": "boolean" }
        }
      }
    },
    "termFields": {
      "type": "array",
      "minItems": 1,
```

```
      "items": {
        "type": "object",
        "required": ["key", "type", "required"],
        "additionalProperties": false,
        "properties": {
          "key": { "type": "string", "minLength": 2, "maxLength": 80
},
          "label": { "type": "string", "maxLength": 160 },
          "type": {
            "type": "string",
            "enum": ["string", "number", "integer", "boolean", "enum",
"object", "array", "geo", "date", "daterange"]
          },
          "required": { "type": "boolean" },
          "enumValues": { "type": "array", "items": { "type": "string"
} },
          "constraints": {
            "type": "object",
            "additionalProperties": false,
            "properties": {
              "min": { "type": "number" },
              "max": { "type": "number" },
              "pattern": { "type": "string" }
            }
          },
          "aiSuggestible": { "type": "boolean", "default": true }
        }
      }
    },
    "policies": {
      "type": "object",
      "required": ["escrow", "guarantees", "dispute", "evidence"],
      "additionalProperties": false,
      "properties": {
        "escrow": {
          "type": "object",
          "required": ["required", "currency", "amountRule",
"fundingMode"],
          "additionalProperties": false,
          "properties": {
            "required": { "type": "boolean", "const": true },
            "currency": { "type": "string", "minLength": 3,
"maxLength": 3 },
            "amountRule": {
              "type": "object",
              "required": ["basis", "percent"],
              "additionalProperties": false,
              "properties": {
                "basis": { "type": "string", "enum":
["ESTIMATED_CONTRACT_VALUE", "FIXED_AMOUNT"] },
                "percent": { "type": "number", "minimum": 0,
"maximum": 200 },
                "fixedAmount": { "type": "number", "minimum": 0 }
              },
              "allOf": [
                {
                  "if": { "properties": { "basis": { "const":
"FIXED_AMOUNT" } } },
                  "then": { "required": ["fixedAmount"] }
                }
              ]
```

```
            },
            "fundingMode": { "type": "string", "enum":
["FULL_UPFRONT", "SCHEDULED"] },
            "fundingSchedule": {
              "type": "array",
              "items": {
                "type": "object",
                "required": ["milestoneKey", "percent"],
                "additionalProperties": false,
                "properties": {
                  "milestoneKey": { "type": "string", "minLength": 2,
"maxLength": 80 },
                  "percent": { "type": "number", "minimum": 0,
"maximum": 100 }
                }
              }
            }
          }
        },
        "guarantees": {
          "type": "object",
          "required": ["buyer", "seller", "execution"],
          "additionalProperties": false,
          "properties": {
            "buyer": {
              "type": "object",
              "required": ["required", "type", "percent"],
              "additionalProperties": false,
              "properties": {
                "required": { "type": "boolean" },
                "type": { "type": "string", "enum": ["CASH_HOLD",
"DOCUMENTED_COLLATERAL", "MIXED"] },
                "percent": { "type": "number", "minimum": 0,
"maximum": 100 }
              }
            },
            "seller": {
              "type": "object",
              "required": ["required", "type", "percent"],
              "additionalProperties": false,
              "properties": {
                "required": { "type": "boolean" },
                "type": { "type": "string", "enum": ["CASH_HOLD",
"HOLDBACK", "DOCUMENTED_COLLATERAL", "MIXED"] },
                "percent": { "type": "number", "minimum": 0,
"maximum": 100 }
              }
            },
            "execution": {
              "type": "object",
              "required": ["autoExecuteAllowed",
"requiresAuditorForAmbiguousCases"],
              "additionalProperties": false,
              "properties": {
                "autoExecuteAllowed": { "type": "boolean" },
                "requiresAuditorForAmbiguousCases": { "type":
"boolean" },
                "defaultCurePeriodDays": { "type": "integer",
"minimum": 0, "maximum": 365 }
              }
            }
```

```
          }
        },
        "dispute": {
          "type": "object",
          "required": ["enabled", "disputeWindowDays", "auditorRole"],
          "additionalProperties": false,
          "properties": {
            "enabled": { "type": "boolean" },
            "disputeWindowDays": { "type": "integer", "minimum": 0,
"maximum": 365 },
            "auditorRole": { "type": "string", "enum": ["AUDITOR"] },
            "freezeEscrowOnDispute": { "type": "boolean", "default":
true }
          }
        },
        "evidence": {
          "type": "object",
          "required": ["types", "rules"],
          "additionalProperties": false,
          "properties": {
            "types": {
              "type": "array",
              "items": { "type": "string", "enum": ["A", "B", "B_LAB"]
},
              "minItems": 1
            },
            "rules": {
              "type": "array",
              "minItems": 1,
              "items": {
                "type": "object",
                "required": ["scope", "evidenceType"],
                "additionalProperties": false,
                "properties": {
                  "scope": { "type": "string", "minLength": 2,
"maxLength": 80 },
                  "evidenceType": { "type": "string", "enum": ["A",
"B", "B_LAB"] },
                  "requiresGeofence": { "type": "boolean" },
                  "requiresMedia": { "type": "boolean" },
                  "requiresLabDoc": { "type": "boolean" },
                  "requiresIdScan": { "type": "boolean" }
                }
              }
            }
          }
        }
      }
    },
    "milestones": {
      "type": "array",
      "minItems": 2,
      "items": {
        "type": "object",
        "required": ["key", "label", "critical", "verification",
"tranche"],
        "additionalProperties": false,
        "properties": {
          "key": { "type": "string", "minLength": 2, "maxLength": 80
},
          "label": { "type": "string", "maxLength": 200 },
```

```
          "critical": { "type": "boolean" },
          "dueWindowDays": { "type": "integer", "minimum": 0,
"maximum": 730 },
          "verification": {
            "type": "object",
            "required": ["mode"],
            "additionalProperties": false,
            "properties": {
              "mode": { "type": "string", "enum": ["AUTO",
"BUYER_ACCEPT", "TECHNICIAN_ACCEPT", "AUDITOR_ACCEPT"] },
              "acceptanceCriteria": { "type": "object" }
            }
          },
          "tranche": {
            "type": "object",
            "required": ["releasePercent", "releaseOnVerified"],
            "additionalProperties": false,
            "properties": {
              "releasePercent": { "type": "number", "minimum": 0,
"maximum": 100 },
              "releaseOnVerified": { "type": "boolean" }
            }
          },
          "failure": {
            "type": "object",
            "additionalProperties": false,
            "properties": {
              "allowCure": { "type": "boolean", "default": true },
              "curePeriodDays": { "type": "integer", "minimum": 0,
"maximum": 365 },
              "defaultEventParty": { "type": "string", "enum":
["BUYER", "SELLER"] },
              "executeGuaranteePercent": { "type": "number",
"minimum": 0, "maximum": 100 }
            }
          }
        }
      }
    },
    "stateMachine": {
      "type": "object",
      "required": ["states", "transitions"],
      "additionalProperties": false,
      "properties": {
        "states": {
          "type": "array",
          "minItems": 3,
          "items": { "type": "string", "minLength": 2, "maxLength": 80
}
        },
        "transitions": {
          "type": "array",
          "minItems": 2,
          "items": {
            "type": "object",
            "required": ["from", "event", "to"],
            "additionalProperties": false,
            "properties": {
              "from": { "type": "string" },
              "event": { "type": "string", "minLength": 2,
"maxLength": 120 },
```

```
              "to": { "type": "string" },
              "requires": {
                "type": "object",
                "additionalProperties": false,
                "properties": {
                  "escrowFunded": { "type": "boolean" },
                  "buyerGuaranteeLocked": { "type": "boolean" },
                  "sellerGuaranteeLocked": { "type": "boolean" },
                  "milestoneVerified": { "type": "string",
"minLength": 2, "maxLength": 80 }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

`termsheet.schema.json` (Schema do TermSheet gerado por negociação/IA)

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://cicloplus.local/schemas/termsheet.schema.json",
  "title": "Ciclo+ TermSheet",
  "type": "object",
  "required": ["termSheetId", "templateId", "buyerOrgId",
"sellerOrgId", "terms", "status"],
  "additionalProperties": false,
  "properties": {
    "termSheetId": { "type": "string", "minLength": 6, "maxLength": 80
},
    "templateId": { "type": "string", "minLength": 3, "maxLength": 80
},
    "buyerOrgId": { "type": "string", "minLength": 2, "maxLength": 80
},
    "sellerOrgId": { "type": "string", "minLength": 2, "maxLength": 80
},
    "technicianOrgId": { "type": "string" },
    "terms": {
      "type": "object",
      "minProperties": 1,
      "additionalProperties": true
    },
    "aiAssistance": {
      "type": "object",
      "additionalProperties": false,
      "properties": {
        "enabled": { "type": "boolean" },
        "suggestions": {
          "type": "array",
          "items": {
            "type": "object",
            "required": ["key", "suggestedValue", "confidence"],
            "additionalProperties": false,
            "properties": {
              "key": { "type": "string" },
              "suggestedValue": {},
```

```
              "confidence": { "type": "number", "minimum": 0,
"maximum": 1 },
              "rationale": { "type": "string", "maxLength": 2000 }
            }
          }
        }
      }
    },
    "status": { "type": "string", "enum": ["DRAFT", "PROPOSED",
"AGREED", "LOCKED", "REJECTED"] }
  }
}
```

template_frigo_abate_v1.json (Frigorífico – compra programada / abate)

```
{
  "templateId": "TEMPLATE_FRIGO_ABATE",
  "name": "Frigorifico Abate - Escrow + Dupla Garantia",
  "version": "v1.0",
  "category": "ANIMAIS",
  "description": "Compra programada para abate com negociacao direta,
escrow obrigatorio, liberacao por evidencia e marcos.",
  "defaults": {
    "pricingMethod": "DEST_WEIGHT",
    "escrowFundingMode": "FULL_UPFRONT",
    "releaseMode": "FINAL_ONLY"
  },
  "parties": [
    { "role": "BUYER", "required": true },
    { "role": "SELLER", "required": true },
    { "role": "TECHNICIAN", "required": false },
    { "role": "CARRIER", "required": false },
    { "role": "AUDITOR", "required": true }
  ],
  "termFields": [
    { "key": "species", "label": "Especie", "type": "enum",
"required": true, "enumValues": ["BOVINO", "SUINO", "OVINO",
"CAPRINO", "AVES", "PEIXES"], "aiSuggestible": true },
    { "key": "volumeHeads", "label": "Quantidade (cabecas)", "type":
"integer", "required": false, "constraints": { "min": 1 },
"aiSuggestible": true },
    { "key": "estimatedKg", "label": "Peso estimado total (kg)",
"type": "number", "required": true, "constraints": { "min": 1 },
"aiSuggestible": true },
    { "key": "deliveryWindow", "label": "Janela de entrega", "type":
"daterange", "required": true, "aiSuggestible": true },
    { "key": "pickupGeo", "label": "Local de embarque", "type": "geo",
"required": true, "aiSuggestible": true },
    { "key": "deliveryGeo", "label": "Unidade frigorifico (destino)",
"type": "geo", "required": true, "aiSuggestible": true },
    { "key": "pricingMethod", "label": "Metodo de precificacao",
"type": "enum", "required": true, "enumValues": ["LIVE_WEIGHT",
"DEST_WEIGHT", "CARCASS_GRADE", "HYBRID"], "aiSuggestible": true },
    { "key": "priceTableRef", "label": "Tabela de preco/grade",
"type": "string", "required": true, "aiSuggestible": true },
    { "key": "qualitySpec", "label": "Criterios de qualidade", "type":
"object", "required": true, "aiSuggestible": true },
    { "key": "tolerances", "label": "Tolerancias (peso, padrao)",
"type": "object", "required": true, "aiSuggestible": true },
    { "key": "idTraceabilityRequired", "label": "Rastreio por ID",
"type": "enum", "required": true, "enumValues": ["NONE", "LOT_ONLY",
"INDIVIDUAL"], "aiSuggestible": true },
```

```
    { "key": "labRequired", "label": "Laudos laboratoriais", "type":
"boolean", "required": true, "aiSuggestible": true }
  ],
  "policies": {
    "escrow": {
      "required": true,
      "currency": "BRL",
      "amountRule": { "basis": "ESTIMATED_CONTRACT_VALUE", "percent":
100 },
      "fundingMode": "FULL_UPFRONT"
    },
    "guarantees": {
      "buyer": { "required": true, "type": "CASH_HOLD", "percent": 15
},
      "seller": { "required": true, "type": "HOLDBACK", "percent": 10
},
      "execution": { "autoExecuteAllowed": true,
"requiresAuditorForAmbiguousCases": true, "defaultCurePeriodDays": 7 }
    },
    "dispute": {
      "enabled": true,
      "disputeWindowDays": 7,
      "auditorRole": "AUDITOR",
      "freezeEscrowOnDispute": true
    },
    "evidence": {
      "types": ["A", "B", "B_LAB"],
      "rules": [
        { "scope": "PICKUP", "evidenceType": "B", "requiresGeofence":
true, "requiresMedia": true, "requiresIdScan": false,
"requiresLabDoc": false },
        { "scope": "DELIVERY", "evidenceType": "B",
"requiresGeofence": true, "requiresMedia": true, "requiresIdScan":
false, "requiresLabDoc": false },
        { "scope": "WEIGHING_OFFICIAL", "evidenceType": "B",
"requiresGeofence": false, "requiresMedia": true, "requiresIdScan":
false, "requiresLabDoc": false },
        { "scope": "ID_SCAN", "evidenceType": "A", "requiresGeofence":
false, "requiresMedia": false, "requiresIdScan": true,
"requiresLabDoc": false },
        { "scope": "LAB", "evidenceType": "B_LAB", "requiresGeofence":
false, "requiresMedia": false, "requiresIdScan": false,
"requiresLabDoc": true }
      ]
    }
  },
  "milestones": [
    {
      "key": "M0_SIGNED",
      "label": "Contrato assinado",
      "critical": true,
      "verification": { "mode": "AUTO", "acceptanceCriteria": {} },
      "tranche": { "releasePercent": 0, "releaseOnVerified": false }
    },
    {
      "key": "M1_GUARANTEES_LOCKED",
      "label": "Garantias travadas (buyer + seller)",
      "critical": true,
      "verification": { "mode": "AUTO", "acceptanceCriteria": {} },
      "tranche": { "releasePercent": 0, "releaseOnVerified": false }
    },
```

```
    {
      "key": "M2_ESCROW_FUNDED",
      "label": "Escrow fundado pelo comprador",
      "critical": true,
      "verification": { "mode": "AUTO", "acceptanceCriteria": {} },
      "tranche": { "releasePercent": 0, "releaseOnVerified": false },
      "failure": { "allowCure": true, "curePeriodDays": 3,
"defaultEventParty": "BUYER", "executeGuaranteePercent": 25 }
    },
    {
      "key": "M3_PICKUP_CONFIRMED",
      "label": "Embarque confirmado (geofence + midia)",
      "critical": true,
      "verification": { "mode": "AUTO", "acceptanceCriteria": {
"evidenceScope": "PICKUP" } },
      "tranche": { "releasePercent": 0, "releaseOnVerified": false }
    },
    {
      "key": "M4_DELIVERY_CONFIRMED",
      "label": "Entrega confirmada (geofence + midia)",
      "critical": true,
      "verification": { "mode": "AUTO", "acceptanceCriteria": {
"evidenceScope": "DELIVERY" } },
      "tranche": { "releasePercent": 0, "releaseOnVerified": false }
    },
    {
      "key": "M5_WEIGHING_AND_QUALITY",
      "label": "Pesagem oficial e checagem de qualidade",
      "critical": true,
      "verification": {
        "mode": "BUYER_ACCEPT",
        "acceptanceCriteria": { "requiresOfficialWeighing": true,
"qualityWithinTolerance": true }
      },
      "tranche": { "releasePercent": 0, "releaseOnVerified": false }
    },
    {
      "key": "M6_RELEASE_ESCROW_FINAL",
      "label": "Liberar pagamento do escrow (final)",
      "critical": true,
      "verification": { "mode": "AUTO", "acceptanceCriteria": {
"requiresMilestones": ["M2_ESCROW_FUNDED", "M3_PICKUP_CONFIRMED",
"M4_DELIVERY_CONFIRMED", "M5_WEIGHING_AND_QUALITY"] } },
      "tranche": { "releasePercent": 100, "releaseOnVerified": true }
    },
    {
      "key": "M7_RELEASE_GUARANTEES",
      "label": "Liberar garantias (buyer + seller) e encerrar",
      "critical": true,
      "verification": { "mode": "AUTO", "acceptanceCriteria": {
"noOpenDisputes": true } },
      "tranche": { "releasePercent": 0, "releaseOnVerified": false }
    }
  ],
  "stateMachine": {
    "states": ["DRAFT", "SIGNED", "GUARANTEES_LOCKED",
"ESCROW_FUNDED", "IN_EXECUTION", "DELIVERED", "VERIFIED", "PAID",
"CLOSED", "DISPUTED"],
    "transitions": [
      { "from": "DRAFT", "event": "CONTRACT_SIGNED", "to": "SIGNED" },
```

```
      { "from": "SIGNED", "event": "GUARANTEES_LOCKED", "to":
"GUARANTEES_LOCKED", "requires": { "buyerGuaranteeLocked": true,
"sellerGuaranteeLocked": true } },
      { "from": "GUARANTEES_LOCKED", "event": "ESCROW_FUNDED", "to":
"ESCROW_FUNDED", "requires": { "escrowFunded": true } },
      { "from": "ESCROW_FUNDED", "event": "PICKUP_CONFIRMED", "to":
"IN_EXECUTION", "requires": { "escrowFunded": true,
"milestoneVerified": "M2_ESCROW_FUNDED" } },
      { "from": "IN_EXECUTION", "event": "DELIVERY_CONFIRMED", "to":
"DELIVERED", "requires": { "escrowFunded": true } },
      { "from": "DELIVERED", "event": "QUALITY_ACCEPTED", "to":
"VERIFIED" },
      { "from": "VERIFIED", "event": "ESCROW_RELEASED", "to": "PAID",
"requires": { "milestoneVerified": "M6_RELEASE_ESCROW_FINAL" } },
      { "from": "PAID", "event": "GUARANTEES_RELEASED", "to": "CLOSED"
},
      { "from": "DELIVERED", "event": "DISPUTE_OPENED", "to":
"DISPUTED" },
      { "from": "DISPUTED", "event": "DISPUTE_RESOLVED", "to":
"VERIFIED" }
    ]
  }
}
```

`template_bovinos_cria_v1.json` (Bovinos – Cria / compra antes de nascer)

```
{
  "templateId": "TEMPLATE_BOVINOS_CRIA",
  "name": "Bovinos Cria - Compra antes de nascer (Escrow + Dupla
Garantia)",
  "version": "v1.0",
  "category": "ANIMAIS",
  "description": "Contrato por ciclo de cria permitindo compra futura
(prenhez -> nascimento -> desmame -> entrega) com acompanhamento e
tranches.",
  "defaults": { "pricingMethod": "FIXED", "escrowFundingMode":
"FULL_UPFRONT", "releaseMode": "TRANCHE" },
  "parties": [
    { "role": "BUYER", "required": true },
    { "role": "SELLER", "required": true },
    { "role": "TECHNICIAN", "required": true },
    { "role": "AUDITOR", "required": true }
  ],
  "termFields": [
    { "key": "species", "label": "Especie", "type": "enum",
"required": true, "enumValues": ["BOVINO"], "aiSuggestible": false },
    { "key": "futureLotCount", "label": "Quantidade futura
(bezerros)", "type": "integer", "required": true, "constraints": {
"min": 1 }, "aiSuggestible": true },
    { "key": "deliveryWindow", "label": "Janela de entrega (pos-
desmame)", "type": "daterange", "required": true, "aiSuggestible":
true },
    { "key": "originGeo", "label": "Fazenda origem", "type": "geo",
"required": true, "aiSuggestible": true },
    { "key": "qualitySpec", "label": "Padrao minimo (peso desmame,
sanidade)", "type": "object", "required": true, "aiSuggestible": true
},
    { "key": "minPregnancyRate", "label": "Taxa minima de prenhez
(%)", "type": "number", "required": true, "constraints": { "min": 0,
"max": 100 }, "aiSuggestible": true },
```

```
    { "key": "minSurvivalRate", "label": "Taxa minima de sobrevivencia
(%)", "type": "number", "required": true, "constraints": { "min": 0,
"max": 100 }, "aiSuggestible": true },
    { "key": "pricePerHead", "label": "Preco por cabeca (base)",
"type": "number", "required": true, "constraints": { "min": 0 },
"aiSuggestible": true }
  ],
  "policies": {
    "escrow": {
      "required": true,
      "currency": "BRL",
      "amountRule": { "basis": "ESTIMATED_CONTRACT_VALUE", "percent":
100 },
      "fundingMode": "FULL_UPFRONT"
    },
    "guarantees": {
      "buyer": { "required": true, "type": "CASH_HOLD", "percent": 20
},
      "seller": { "required": true, "type": "DOCUMENTED_COLLATERAL",
"percent": 15 },
      "execution": { "autoExecuteAllowed": true,
"requiresAuditorForAmbiguousCases": true, "defaultCurePeriodDays": 10
}
    },
    "dispute": { "enabled": true, "disputeWindowDays": 10,
"auditorRole": "AUDITOR", "freezeEscrowOnDispute": true },
    "evidence": {
      "types": ["A", "B", "B_LAB"],
      "rules": [
        { "scope": "INVENTORY_BASE", "evidenceType": "B",
"requiresGeofence": true, "requiresMedia": true, "requiresIdScan":
true, "requiresLabDoc": false },
        { "scope": "PREGNANCY_CONFIRM", "evidenceType": "B_LAB",
"requiresGeofence": false, "requiresMedia": false, "requiresIdScan":
false, "requiresLabDoc": true },
        { "scope": "BIRTH_AND_ID", "evidenceType": "A",
"requiresGeofence": false, "requiresMedia": false, "requiresIdScan":
true, "requiresLabDoc": false },
        { "scope": "WEANING_WEIGHT", "evidenceType": "B",
"requiresGeofence": false, "requiresMedia": true, "requiresIdScan":
true, "requiresLabDoc": false },
        { "scope": "PICKUP_DELIVERY", "evidenceType": "B",
"requiresGeofence": true, "requiresMedia": true, "requiresIdScan":
false, "requiresLabDoc": false }
      ]
    }
  },
  "milestones": [
    { "key": "M0_SIGNED", "label": "Contrato assinado", "critical":
true, "verification": { "mode": "AUTO", "acceptanceCriteria": {} },
"tranche": { "releasePercent": 0, "releaseOnVerified": false } },
    { "key": "M1_GUARANTEES_LOCKED", "label": "Garantias travadas",
"critical": true, "verification": { "mode": "AUTO",
"acceptanceCriteria": {} }, "tranche": { "releasePercent": 0,
"releaseOnVerified": false } },
    {
      "key": "M2_ESCROW_FUNDED",
      "label": "Escrow fundado (inicio do ciclo)",
      "critical": true,
      "verification": { "mode": "AUTO", "acceptanceCriteria": {} },
      "tranche": { "releasePercent": 0, "releaseOnVerified": false },
```

```
      "failure": { "allowCure": true, "curePeriodDays": 3,
"defaultEventParty": "BUYER", "executeGuaranteePercent": 30 }
    },
    {
      "key": "M3_INVENTORY_BASE_VERIFIED",
      "label": "Inventario base validado por tecnico (matrizes) +
evidencia",
      "critical": true,
      "verification": { "mode": "TECHNICIAN_ACCEPT",
"acceptanceCriteria": { "evidenceScope": "INVENTORY_BASE" } },
      "tranche": { "releasePercent": 0, "releaseOnVerified": false }
    },
    {
      "key": "M4_PREGNANCY_CONFIRMED",
      "label": "Prenhez confirmada (laudo/atesto)",
      "critical": true,
      "verification": { "mode": "TECHNICIAN_ACCEPT",
"acceptanceCriteria": { "evidenceScope": "PREGNANCY_CONFIRM",
"minRateField": "minPregnancyRate" } },
      "tranche": { "releasePercent": 10, "releaseOnVerified": true }
    },
    {
      "key": "M5_BIRTH_AND_ID",
      "label": "Nascimento + identificacao (lote/individual)",
      "critical": true,
      "verification": { "mode": "AUTO", "acceptanceCriteria": {
"evidenceScope": "BIRTH_AND_ID" } },
      "tranche": { "releasePercent": 20, "releaseOnVerified": true }
    },
    {
      "key": "M6_WEANING_WEIGHT_OK",
      "label": "Desmame + peso minimo e sanidade",
      "critical": true,
      "verification": { "mode": "TECHNICIAN_ACCEPT",
"acceptanceCriteria": { "evidenceScope": "WEANING_WEIGHT",
"qualityWithinTolerance": true } },
      "tranche": { "releasePercent": 30, "releaseOnVerified": true },
      "failure": { "allowCure": true, "curePeriodDays": 15,
"defaultEventParty": "SELLER", "executeGuaranteePercent": 25 }
    },
    {
      "key": "M7_DELIVERY_CONFIRMED",
      "label": "Entrega confirmada (geofence + midia)",
      "critical": true,
      "verification": { "mode": "BUYER_ACCEPT", "acceptanceCriteria":
{ "evidenceScope": "PICKUP_DELIVERY" } },
      "tranche": { "releasePercent": 40, "releaseOnVerified": true }
    },
    {
      "key": "M8_RELEASE_GUARANTEES_CLOSE",
      "label": "Liberar garantias e encerrar",
      "critical": true,
      "verification": { "mode": "AUTO", "acceptanceCriteria": {
"noOpenDisputes": true } },
      "tranche": { "releasePercent": 0, "releaseOnVerified": false }
    }
  ],
  "stateMachine": {
    "states": ["DRAFT", "SIGNED", "GUARANTEES_LOCKED",
"ESCROW_FUNDED", "IN_CYCLE", "DELIVERED", "PAID", "CLOSED",
"DISPUTED"],
```

```
    "transitions": [
      { "from": "DRAFT", "event": "CONTRACT_SIGNED", "to": "SIGNED" },
      { "from": "SIGNED", "event": "GUARANTEES_LOCKED", "to":
"GUARANTEES_LOCKED", "requires": { "buyerGuaranteeLocked": true,
"sellerGuaranteeLocked": true } },
      { "from": "GUARANTEES_LOCKED", "event": "ESCROW_FUNDED", "to":
"ESCROW_FUNDED", "requires": { "escrowFunded": true } },
      { "from": "ESCROW_FUNDED", "event": "CYCLE_STARTED", "to":
"IN_CYCLE", "requires": { "milestoneVerified": "M2_ESCROW_FUNDED" } },
      { "from": "IN_CYCLE", "event": "DELIVERY_CONFIRMED", "to":
"DELIVERED" },
      { "from": "DELIVERED", "event": "ESCROW_TRANCHE_RELEASED", "to":
"PAID" },
      { "from": "PAID", "event": "GUARANTEES_RELEASED", "to": "CLOSED"
},
      { "from": "IN_CYCLE", "event": "DISPUTE_OPENED", "to":
"DISPUTED" },
      { "from": "DISPUTED", "event": "DISPUTE_RESOLVED", "to":
"IN_CYCLE" }
    ]
  }
}
```

`template_aves_lote_v1.json` (Aves – lote futuro / ciclo de galpão)

```
{
  "templateId": "TEMPLATE_AVES_LOTE",
  "name": "Aves - Lote futuro (Escrow + Dupla Garantia)",
  "version": "v1.0",
  "category": "AVICULTURA",
  "description": "Contrato por ciclo de galpao com alojamento,
checkpoints, retirada e entrega. Escrow obrigatorio e liberacao por
evidencia.",
  "defaults": { "pricingMethod": "DEST_WEIGHT", "escrowFundingMode":
"FULL_UPFRONT", "releaseMode": "TRANCHE" },
  "parties": [
    { "role": "BUYER", "required": true },
    { "role": "SELLER", "required": true },
    { "role": "TECHNICIAN", "required": false },
    { "role": "AUDITOR", "required": true }
  ],
  "termFields": [
    { "key": "species", "label": "Especie", "type": "enum",
"required": true, "enumValues": ["AVES"], "aiSuggestible": false },
    { "key": "lotSize", "label": "Tamanho do lote (cabecas)", "type":
"integer", "required": true, "constraints": { "min": 100 },
"aiSuggestible": true },
    { "key": "cycleDays", "label": "Duracao do ciclo (dias)", "type":
"integer", "required": true, "constraints": { "min": 20, "max": 90 },
"aiSuggestible": true },
    { "key": "deliveryWindow", "label": "Janela de retirada/entrega",
"type": "daterange", "required": true, "aiSuggestible": true },
    { "key": "originGeo", "label": "Origem (granja)", "type": "geo",
"required": true, "aiSuggestible": true },
    { "key": "deliveryGeo", "label": "Destino (planta)", "type":
"geo", "required": true, "aiSuggestible": true },
    { "key": "mortalityMaxPercent", "label": "Mortalidade maxima (%)",
"type": "number", "required": true, "constraints": { "min": 0, "max":
100 }, "aiSuggestible": true },
```

```
    { "key": "weightTargetKg", "label": "Peso alvo medio (kg)",
"type": "number", "required": true, "constraints": { "min": 0.5 },
"aiSuggestible": true },
    { "key": "priceTableRef", "label": "Tabela de preco", "type":
"string", "required": true, "aiSuggestible": true }
  ],
  "policies": {
    "escrow": { "required": true, "currency": "BRL", "amountRule": {
"basis": "ESTIMATED_CONTRACT_VALUE", "percent": 100 }, "fundingMode":
"FULL_UPFRONT" },
    "guarantees": {
      "buyer": { "required": true, "type": "CASH_HOLD", "percent": 10
},
      "seller": { "required": true, "type": "HOLDBACK", "percent": 10
},
      "execution": { "autoExecuteAllowed": true,
"requiresAuditorForAmbiguousCases": true, "defaultCurePeriodDays": 5 }
    },
    "dispute": { "enabled": true, "disputeWindowDays": 5,
"auditorRole": "AUDITOR", "freezeEscrowOnDispute": true },
    "evidence": {
      "types": ["A", "B"],
      "rules": [
        { "scope": "ALLOCATION", "evidenceType": "A",
"requiresGeofence": false, "requiresMedia": false, "requiresIdScan":
false, "requiresLabDoc": false },
        { "scope": "CHECKPOINT", "evidenceType": "A",
"requiresGeofence": false, "requiresMedia": false, "requiresIdScan":
false, "requiresLabDoc": false },
        { "scope": "PICKUP", "evidenceType": "B", "requiresGeofence":
true, "requiresMedia": true, "requiresIdScan": false,
"requiresLabDoc": false },
        { "scope": "DELIVERY", "evidenceType": "B",
"requiresGeofence": true, "requiresMedia": true, "requiresIdScan":
false, "requiresLabDoc": false },
        { "scope": "WEIGHING_OFFICIAL", "evidenceType": "B",
"requiresGeofence": false, "requiresMedia": true, "requiresIdScan":
false, "requiresLabDoc": false }
      ]
    }
  },
  "milestones": [
    { "key": "M0_SIGNED", "label": "Contrato assinado", "critical":
true, "verification": { "mode": "AUTO", "acceptanceCriteria": {} },
"tranche": { "releasePercent": 0, "releaseOnVerified": false } },
    { "key": "M1_GUARANTEES_LOCKED", "label": "Garantias travadas",
"critical": true, "verification": { "mode": "AUTO",
"acceptanceCriteria": {} }, "tranche": { "releasePercent": 0,
"releaseOnVerified": false } },
    { "key": "M2_ESCROW_FUNDED", "label": "Escrow fundado",
"critical": true, "verification": { "mode": "AUTO",
"acceptanceCriteria": {} }, "tranche": { "releasePercent": 0,
"releaseOnVerified": false } },
    {
      "key": "M3_LOT_ALOJADO",
      "label": "Alojamento do lote",
      "critical": true,
      "verification": { "mode": "AUTO", "acceptanceCriteria": {
"evidenceScope": "ALLOCATION" } },
      "tranche": { "releasePercent": 20, "releaseOnVerified": true }
    },
```

```
    {
      "key": "M4_CHECKPOINTS_OK",
      "label": "Checkpoints (peso/mortalidade) dentro do acordado",
      "critical": true,
      "verification": { "mode": "AUTO", "acceptanceCriteria": {
"evidenceScope": "CHECKPOINT", "mortalityWithinTolerance": true } },
      "tranche": { "releasePercent": 20, "releaseOnVerified": true }
    },
    { "key": "M5_PICKUP_CONFIRMED", "label": "Retirada confirmada
(B)", "critical": true, "verification": { "mode": "AUTO",
"acceptanceCriteria": { "evidenceScope": "PICKUP" } }, "tranche": {
"releasePercent": 0, "releaseOnVerified": false } },
    { "key": "M6_DELIVERY_CONFIRMED", "label": "Entrega confirmada
(B)", "critical": true, "verification": { "mode": "AUTO",
"acceptanceCriteria": { "evidenceScope": "DELIVERY" } }, "tranche": {
"releasePercent": 40, "releaseOnVerified": true } },
    { "key": "M7_RELEASE_GUARANTEES_CLOSE", "label": "Liberar
garantias e encerrar", "critical": true, "verification": { "mode":
"AUTO", "acceptanceCriteria": { "noOpenDisputes": true } }, "tranche":
{ "releasePercent": 0, "releaseOnVerified": false } }
  ],
  "stateMachine": {
    "states": ["DRAFT", "SIGNED", "GUARANTEES_LOCKED",
"ESCROW_FUNDED", "IN_CYCLE", "DELIVERED", "PAID", "CLOSED",
"DISPUTED"],
    "transitions": [
      { "from": "DRAFT", "event": "CONTRACT_SIGNED", "to": "SIGNED" },
      { "from": "SIGNED", "event": "GUARANTEES_LOCKED", "to":
"GUARANTEES_LOCKED" },
      { "from": "GUARANTEES_LOCKED", "event": "ESCROW_FUNDED", "to":
"ESCROW_FUNDED", "requires": { "escrowFunded": true } },
      { "from": "ESCROW_FUNDED", "event": "LOT_ALOJADO", "to":
"IN_CYCLE" },
      { "from": "IN_CYCLE", "event": "DELIVERY_CONFIRMED", "to":
"DELIVERED" },
      { "from": "DELIVERED", "event": "ESCROW_TRANCHE_RELEASED", "to":
"PAID" },
      { "from": "PAID", "event": "GUARANTEES_RELEASED", "to": "CLOSED"
},
      { "from": "IN_CYCLE", "event": "DISPUTE_OPENED", "to":
"DISPUTED" },
      { "from": "DISPUTED", "event": "DISPUTE_RESOLVED", "to":
"IN_CYCLE" }
    ]
  }
}
```


`template_peixes_tanque_v1.json` (Peixes – ciclo por tanque / biomassa futura)


```
{
  "templateId": "TEMPLATE_PEIXES_TANQUE",
  "name": "Peixes - Tanque/Lote futuro (Escrow + Dupla Garantia)",
  "version": "v1.0",
  "category": "AQUICULTURA",
  "description": "Contrato por ciclo de aquicultura com povoamento,
checkpoints de biomassa, qualidade de agua, colheita e entrega.",
  "defaults": { "pricingMethod": "DEST_WEIGHT", "escrowFundingMode":
"FULL_UPFRONT", "releaseMode": "TRANCHE" },
  "parties": [
    { "role": "BUYER", "required": true },
    { "role": "SELLER", "required": true },
```

```
    { "role": "TECHNICIAN", "required": false },
    { "role": "AUDITOR", "required": true }
  ],
  "termFields": [
    { "key": "species", "label": "Especie", "type": "enum",
"required": true, "enumValues": ["PEIXES"], "aiSuggestible": false },
    { "key": "tankCount", "label": "Numero de tanques", "type":
"integer", "required": true, "constraints": { "min": 1 },
"aiSuggestible": true },
    { "key": "estimatedKg", "label": "Biomassa estimada (kg)", "type":
"number", "required": true, "constraints": { "min": 10 },
"aiSuggestible": true },
    { "key": "cycleDays", "label": "Duracao do ciclo (dias)", "type":
"integer", "required": true, "constraints": { "min": 30, "max": 365 },
"aiSuggestible": true },
    { "key": "originGeo", "label": "Origem (viveiro)", "type": "geo",
"required": true, "aiSuggestible": true },
    { "key": "deliveryGeo", "label": "Destino (planta)", "type":
"geo", "required": true, "aiSuggestible": true },
    { "key": "waterQualitySpec", "label": "Parametros de agua
(faixas)", "type": "object", "required": true, "aiSuggestible": true
},
    { "key": "labRequired", "label": "Laudos de qualidade", "type":
"boolean", "required": true, "aiSuggestible": true },
    { "key": "priceTableRef", "label": "Tabela de preco", "type":
"string", "required": true, "aiSuggestible": true }
  ],
  "policies": {
    "escrow": { "required": true, "currency": "BRL", "amountRule": {
"basis": "ESTIMATED_CONTRACT_VALUE", "percent": 100 }, "fundingMode":
"FULL_UPFRONT" },
    "guarantees": {
      "buyer": { "required": true, "type": "CASH_HOLD", "percent": 12
},
      "seller": { "required": true, "type": "DOCUMENTED_COLLATERAL",
"percent": 12 },
      "execution": { "autoExecuteAllowed": true,
"requiresAuditorForAmbiguousCases": true, "defaultCurePeriodDays": 7 }
    },
    "dispute": { "enabled": true, "disputeWindowDays": 7,
"auditorRole": "AUDITOR", "freezeEscrowOnDispute": true },
    "evidence": {
      "types": ["A", "B", "B_LAB"],
      "rules": [
        { "scope": "STOCKING", "evidenceType": "B",
"requiresGeofence": true, "requiresMedia": true, "requiresIdScan":
false, "requiresLabDoc": false },
        { "scope": "BIOMASS_CHECK", "evidenceType": "B",
"requiresGeofence": false, "requiresMedia": true, "requiresIdScan":
false, "requiresLabDoc": false },
        { "scope": "WATER_LAB", "evidenceType": "B_LAB",
"requiresGeofence": false, "requiresMedia": false, "requiresIdScan":
false, "requiresLabDoc": true },
        { "scope": "HARVEST", "evidenceType": "B", "requiresGeofence":
true, "requiresMedia": true, "requiresIdScan": false,
"requiresLabDoc": false },
        { "scope": "DELIVERY", "evidenceType": "B",
"requiresGeofence": true, "requiresMedia": true, "requiresIdScan":
false, "requiresLabDoc": false }
      ]
    }
```

```
  },
  "milestones": [
    { "key": "M0_SIGNED", "label": "Contrato assinado", "critical":
true, "verification": { "mode": "AUTO", "acceptanceCriteria": {} },
"tranche": { "releasePercent": 0, "releaseOnVerified": false } },
    { "key": "M1_GUARANTEES_LOCKED", "label": "Garantias travadas",
"critical": true, "verification": { "mode": "AUTO",
"acceptanceCriteria": {} }, "tranche": { "releasePercent": 0,
"releaseOnVerified": false } },
    { "key": "M2_ESCROW_FUNDED", "label": "Escrow fundado",
"critical": true, "verification": { "mode": "AUTO",
"acceptanceCriteria": {} }, "tranche": { "releasePercent": 0,
"releaseOnVerified": false } },
    { "key": "M3_STOCKING_DONE", "label": "Povoamento realizado (B)",
"critical": true, "verification": { "mode": "AUTO",
"acceptanceCriteria": { "evidenceScope": "STOCKING" } }, "tranche": {
"releasePercent": 15, "releaseOnVerified": true } },
    { "key": "M4_BIOMASS_OK", "label": "Checkpoint biomassa (B)",
"critical": true, "verification": { "mode": "AUTO",
"acceptanceCriteria": { "evidenceScope": "BIOMASS_CHECK" } },
"tranche": { "releasePercent": 25, "releaseOnVerified": true } },
    { "key": "M5_WATER_LAB_OK", "label": "Laudo agua/qualidade OK",
"critical": false, "verification": { "mode": "AUTO",
"acceptanceCriteria": { "evidenceScope": "WATER_LAB" } }, "tranche": {
"releasePercent": 0, "releaseOnVerified": false } },
    { "key": "M6_HARVEST_DONE", "label": "Colheita/despesca (B)",
"critical": true, "verification": { "mode": "AUTO",
"acceptanceCriteria": { "evidenceScope": "HARVEST" } }, "tranche": {
"releasePercent": 20, "releaseOnVerified": true } },
    { "key": "M7_DELIVERY_CONFIRMED", "label": "Entrega confirmada
(B)", "critical": true, "verification": { "mode": "BUYER_ACCEPT",
"acceptanceCriteria": { "evidenceScope": "DELIVERY" } }, "tranche": {
"releasePercent": 40, "releaseOnVerified": true } },
    { "key": "M8_RELEASE_GUARANTEES_CLOSE", "label": "Liberar
garantias e encerrar", "critical": true, "verification": { "mode":
"AUTO", "acceptanceCriteria": { "noOpenDisputes": true } }, "tranche":
{ "releasePercent": 0, "releaseOnVerified": false } }
  ],
  "stateMachine": {
    "states": ["DRAFT", "SIGNED", "GUARANTEES_LOCKED",
"ESCROW_FUNDED", "IN_CYCLE", "DELIVERED", "PAID", "CLOSED",
"DISPUTED"],
    "transitions": [
      { "from": "DRAFT", "event": "CONTRACT_SIGNED", "to": "SIGNED" },
      { "from": "SIGNED", "event": "GUARANTEES_LOCKED", "to":
"GUARANTEES_LOCKED" },
      { "from": "GUARANTEES_LOCKED", "event": "ESCROW_FUNDED", "to":
"ESCROW_FUNDED", "requires": { "escrowFunded": true } },
      { "from": "ESCROW_FUNDED", "event": "STOCKING_DONE", "to":
"IN_CYCLE" },
      { "from": "IN_CYCLE", "event": "DELIVERY_CONFIRMED", "to":
"DELIVERED" },
      { "from": "DELIVERED", "event": "ESCROW_TRANCHE_RELEASED", "to":
"PAID" },
      { "from": "PAID", "event": "GUARANTEES_RELEASED", "to": "CLOSED"
},
      { "from": "IN_CYCLE", "event": "DISPUTE_OPENED", "to":
"DISPUTED" },
      { "from": "DISPUTED", "event": "DISPUTE_RESOLVED", "to":
"IN_CYCLE" }
    ]
```

```
  }
}
```

`template_safra_futura_v1.json` (Safra futura – plantações)

```
{
  "templateId": "TEMPLATE_SAFRA_FUTURA",
  "name": "Safra Futura - Plantacoes (Escrow + Dupla Garantia)",
  "version": "v1.0",
  "category": "CULTURAS",
  "description": "Contrato para compra futura de safra com marcos de
plantio, tratos, monitoramento, colheita, classificacao e entrega.",
  "defaults": { "pricingMethod": "FIXED", "escrowFundingMode":
"FULL_UPFRONT", "releaseMode": "TRANCHE" },
  "parties": [
    { "role": "BUYER", "required": true },
    { "role": "SELLER", "required": true },
    { "role": "TECHNICIAN", "required": false },
    { "role": "AUDITOR", "required": true }
  ],
  "termFields": [
    { "key": "crop", "label": "Cultura", "type": "enum", "required":
true, "enumValues": ["SOJA", "MILHO", "ALGODAO", "TRIGO", "FEIJAO",
"ARROZ", "OUTRA"], "aiSuggestible": true },
    { "key": "areaHa", "label": "Area (ha)", "type": "number",
"required": true, "constraints": { "min": 1 }, "aiSuggestible": true
},
    { "key": "estimatedTons", "label": "Producao estimada (t)",
"type": "number", "required": true, "constraints": { "min": 1 },
"aiSuggestible": true },
    { "key": "fieldGeo", "label": "Talhoes/area (geo)", "type": "geo",
"required": true, "aiSuggestible": true },
    { "key": "deliveryWindow", "label": "Janela de entrega", "type":
"daterange", "required": true, "aiSuggestible": true },
    { "key": "qualitySpec", "label": "Especificacao (umidade,
impurezas, classe)", "type": "object", "required": true,
"aiSuggestible": true },
    { "key": "labRequired", "label": "Laudo de classificacao/umidade",
"type": "boolean", "required": true, "aiSuggestible": true },
    { "key": "pricePerTon", "label": "Preco por tonelada", "type":
"number", "required": true, "constraints": { "min": 0 },
"aiSuggestible": true }
  ],
  "policies": {
    "escrow": { "required": true, "currency": "BRL", "amountRule": {
"basis": "ESTIMATED_CONTRACT_VALUE", "percent": 100 }, "fundingMode":
"FULL_UPFRONT" },
    "guarantees": {
      "buyer": { "required": true, "type": "CASH_HOLD", "percent": 12
},
      "seller": { "required": true, "type": "DOCUMENTED_COLLATERAL",
"percent": 12 },
      "execution": { "autoExecuteAllowed": true,
"requiresAuditorForAmbiguousCases": true, "defaultCurePeriodDays": 10
}
    },
    "dispute": { "enabled": true, "disputeWindowDays": 10,
"auditorRole": "AUDITOR", "freezeEscrowOnDispute": true },
    "evidence": {
      "types": ["A", "B", "B_LAB"],
      "rules": [
```

```
        { "scope": "PLANTING", "evidenceType": "B",
"requiresGeofence": true, "requiresMedia": true, "requiresIdScan":
false, "requiresLabDoc": false },
        { "scope": "CROP_MONITORING", "evidenceType": "B",
"requiresGeofence": false, "requiresMedia": true, "requiresIdScan":
false, "requiresLabDoc": false },
        { "scope": "HARVEST", "evidenceType": "B", "requiresGeofence":
true, "requiresMedia": true, "requiresIdScan": false,
"requiresLabDoc": false },
        { "scope": "LAB_CLASSIFICATION", "evidenceType": "B_LAB",
"requiresGeofence": false, "requiresMedia": false, "requiresIdScan":
false, "requiresLabDoc": true },
        { "scope": "DELIVERY", "evidenceType": "B",
"requiresGeofence": true, "requiresMedia": true, "requiresIdScan":
false, "requiresLabDoc": false }
      ]
    }
  },
  "milestones": [
    { "key": "M0_SIGNED", "label": "Contrato assinado", "critical":
true, "verification": { "mode": "AUTO", "acceptanceCriteria": {} },
"tranche": { "releasePercent": 0, "releaseOnVerified": false } },
    { "key": "M1_GUARANTEES_LOCKED", "label": "Garantias travadas",
"critical": true, "verification": { "mode": "AUTO",
"acceptanceCriteria": {} }, "tranche": { "releasePercent": 0,
"releaseOnVerified": false } },
    { "key": "M2_ESCROW_FUNDED", "label": "Escrow fundado",
"critical": true, "verification": { "mode": "AUTO",
"acceptanceCriteria": {} }, "tranche": { "releasePercent": 0,
"releaseOnVerified": false } },
    { "key": "M3_PLANTING_CONFIRMED", "label": "Plantio confirmado
(B)", "critical": true, "verification": { "mode": "AUTO",
"acceptanceCriteria": { "evidenceScope": "PLANTING" } }, "tranche": {
"releasePercent": 20, "releaseOnVerified": true } },
    { "key": "M4_MONITORING_OK", "label": "Monitoramento
(drone/inspecao) OK", "critical": false, "verification": { "mode":
"AUTO", "acceptanceCriteria": { "evidenceScope": "CROP_MONITORING" }
}, "tranche": { "releasePercent": 20, "releaseOnVerified": true } },
    { "key": "M5_HARVEST_DONE", "label": "Colheita realizada (B)",
"critical": true, "verification": { "mode": "AUTO",
"acceptanceCriteria": { "evidenceScope": "HARVEST" } }, "tranche": {
"releasePercent": 20, "releaseOnVerified": true } },
    { "key": "M6_LAB_CLASS_OK", "label": "Classificacao/umidade
(laudo)", "critical": true, "verification": { "mode": "BUYER_ACCEPT",
"acceptanceCriteria": { "evidenceScope": "LAB_CLASSIFICATION",
"qualityWithinTolerance": true } }, "tranche": { "releasePercent": 0,
"releaseOnVerified": false } },
    { "key": "M7_DELIVERY_CONFIRMED", "label": "Entrega confirmada
(B)", "critical": true, "verification": { "mode": "BUYER_ACCEPT",
"acceptanceCriteria": { "evidenceScope": "DELIVERY" } }, "tranche": {
"releasePercent": 40, "releaseOnVerified": true } },
    { "key": "M8_RELEASE_GUARANTEES_CLOSE", "label": "Liberar
garantias e encerrar", "critical": true, "verification": { "mode":
"AUTO", "acceptanceCriteria": { "noOpenDisputes": true } }, "tranche":
{ "releasePercent": 0, "releaseOnVerified": false } }
  ],
  "stateMachine": {
    "states": ["DRAFT", "SIGNED", "GUARANTEES_LOCKED",
"ESCROW_FUNDED", "IN_CYCLE", "DELIVERED", "PAID", "CLOSED",
"DISPUTED"],
    "transitions": [
```

```
      { "from": "DRAFT", "event": "CONTRACT_SIGNED", "to": "SIGNED" },
      { "from": "SIGNED", "event": "GUARANTEES_LOCKED", "to":
"GUARANTEES_LOCKED" },
      { "from": "GUARANTEES_LOCKED", "event": "ESCROW_FUNDED", "to":
"ESCROW_FUNDED", "requires": { "escrowFunded": true } },
      { "from": "ESCROW_FUNDED", "event": "PLANTING_CONFIRMED", "to":
"IN_CYCLE" },
      { "from": "IN_CYCLE", "event": "DELIVERY_CONFIRMED", "to":
"DELIVERED" },
      { "from": "DELIVERED", "event": "ESCROW_TRANCHE_RELEASED", "to":
"PAID" },
      { "from": "PAID", "event": "GUARANTEES_RELEASED", "to": "CLOSED"
},
      { "from": "IN_CYCLE", "event": "DISPUTE_OPENED", "to":
"DISPUTED" },
      { "from": "DISPUTED", "event": "DISPUTE_RESOLVED", "to":
"IN_CYCLE" }
    ]
  }
}
```

Sim — dá para transformar isso em **regra técnica do motor (pré-execução)** com **bloqueio por propriedade** usando **CAR** e/ou **Inscrição Estadual do imóvel**, e ainda adicionar **limites ambientais** como condição de elegibilidade para operar.

A forma "boa para dev" é modelar como **Compliance Gate** (um módulo de compliance que publica status e restrições) + **Policy Engine** (motor de regras) que bloqueia qualquer evento comercial/contratual quando o gate estiver violado.

1) Conceito: "Bloqueio por Propriedade" (CAR/IE)

Identificadores do imóvel

- `propertyId` (interno)
- `carCode` (CAR)
- `stateRegistration` (Inscrição estadual do imóvel, quando aplicável)
- opcional: `cib/incra`, `matricula` (se quiser expandir)

Estado de compliance do imóvel

- `complianceStatus`: `ACTIVE | WATCHLIST | BLOCKED`
- `blockReasons[]`: lista padronizada (ex.: `DEFAULT_CONTRACT`, `ENV_LIMIT`, `FRAUD_SUSPECT`, `REGULATORY_HOLD`)
- `blockedScopes[]`: o que está bloqueado (ex.: `MARKETPLACE_SELL`, `MARKETPLACE_BUY`, `CONTRACT_SIGN`, `ESCROW_RELEASE`, `PROJECT_CREATE`, `SHIPMENT`)
- `blockUntil` (data) ou `conditionsToUnblock[]`

Regra prática

Se **qualquer operação** envolver um `propertyId` que esteja `BLOCKED`, o sistema:

- **nega antes de executar** (pré-execução)
- gera evento auditável `OPERATION_BLOCKED`
- registra motivo + base do bloqueio (CAR/IE + contrato relacionado) em auditoria imutável

Isso atende sua exigência: "tornando ele impossibilitado de praticar comercialização".

---

2) Bloqueio por descumprimento de contrato (com efeito no imóvel)

Quando dispara

- `milestone overdue` (marco não cumprido no prazo)
- `evidence invalid` (evidência não válida para marco crítico)
- `dispute resolved against party` (decisão do auditor)
- `guarantee executed` (execução de garantia)
- `fraud flag` (tentativa repetida de transição inválida, manipulação etc.)

Como vira bloqueio do imóvel

Ao confirmar inadimplência/descumprimento, cria-se um evento:

- `PROPERTY_BLOCKED_FOR_DEFAULT`
  - vincula: `contractId`, `propertyId`, `carCode/stateRegistration`, `partyOrgId`
  - define: `blockedScopes` (por exemplo, vender/firmar novos contratos)
  - define: condição de desbloqueio (ex.: pagar multa, acordo aceito, auditor liberar)

Esse bloqueio passa a ser consultado pelo motor de regras sempre que:

- criar contrato
- publicar oferta (produção/insumos)
- iniciar projeto
- liberar escrow/tranches
- registrar entrega/embarque

---

3) Limites ambientais como motivo de bloqueio

Aqui entra o "projeto/atividade respeita limite ambiental":

Modelo de "limites ambientais"

Crie um objeto `EnvironmentalConstraints` por propriedade:

- `allowedActivities[]` (pecuária, piscicultura, agricultura, confinamento, etc.)
- `maxHeads` / `maxDensity` (quando fizer sentido)
- `maxAreaHaByCrop`
- `restrictedZones` (polígonos onde não pode operar)
- `waterUseLimit` (se você quiser expandir)
- `documents[]` (licenças, autorizações) com evidência `B_LAB`/documental
- `status`: `OK | EXCEEDS_LIMIT | MISSING_DOCS | EXPIRED_DOCS`

Regras do motor (exemplos objetivos)

- **Criar projeto** (`PROJECT_CREATE`) bloqueia se:
  - atividade não estiver em `allowedActivities`
  - ultrapassar `maxHeads` ou `maxAreaHaByCrop`
  - operar em `restrictedZones`
  - documentos obrigatórios ausentes/expirados
- **Registrar eventos do ciclo** (`STOCK_APPLY`, `WEIGHING`, `DELIVERY`) bloqueia se:
  - projeto já estiver `EXCEEDS_LIMIT`
  - geofence estiver fora do perímetro autorizado (CAR + zonas)

Importante: o motor de regras não precisa "interpretar lei". Ele só verifica **parâmetros** cadastrados (por documentos e validação técnica/auditor).

---

4) Escopos de bloqueio (para ficar claro no sistema)

Eu recomendo padronizar scopes:

Comercialização

- `MARKETPLACE_SELL` / `MARKETPLACE_BUY`
- `AUCTION_SELL` / `AUCTION_BID`

Contratos

- `CONTRACT_CREATE` / `CONTRACT_SIGN`
- `CONTRACT_MILESTONE_VERIFY`

Financeiro

- `ESCROW_FUND` / `ESCROW_RELEASE`

Operacional

- `PROJECT_CREATE` / `PROJECT_UPDATE`
- `SHIPMENT_CREATE` / `DELIVERY_CONFIRM`

Assim, o bloqueio pode ser "cirúrgico" (só comercialização) ou "total" (tudo).

---

5) Auditoria e “devido processo”

Como isso é sensível, faça com trilha auditável:

- todo bloqueio gera evento append-only com hash
- todo desbloqueio também (`PROPERTY_UNBLOCKED`)
- interface **read-only para auditor/fiscal** + “parecer” (evento de decisão)

---

6) Implementação simples para DEV

Tabelas/coleções mínimas

- `properties` (com CAR/IE, geometry)
- `property_compliance` (status, scopes, reasons, until)
- `environmental_constraints` (parâmetros + docs)
- `events_audit` (append-only encadeado)

Função central do motor

Antes de executar qualquer operação:

1. `resolvePropertyContext(op)` → acha `propertyId`, `carCode`, `stateRegistration`
2. `checkComplianceGate(propertyId, op.scope)`
3. `checkEnvironmentalLimits(propertyId, op.activity, op.quantities, op.geometry)`
4. se falhar: **bloqueia** + `OPERATION_BLOCKED` (com motivo padronizado)

Se você quiser, eu incorporo isso no pacote que te entreguei adicionando:

- `property.schema.json`, `property_compliance.schema.json`, `environmental_constraints.schema.json`
- Regras do Policy Engine (JSON) com `blockedScopes` e motivos
- Telas mobile: **Cadastro Propriedade (CAR/IE)** + **Status Compliance** + **Operação bloqueada (motivo + ação corretiva)**

Cadastro da propriedade com CAR + Inscrição Estadual

- Tela e fluxo **“Cadastrar Propriedade (CAR/IE)”**
- Schema `property.schema.json`
- Evidência anexada (foto/PDF) com hash e auditoria

2) Bloqueios por descumprimento de contrato vinculados ao imóvel (CAR/IE)

- **Compliance Gate por propriedade** (ACTIVE/WATCHLIST/BLOCKED)
- Bloqueio **pré-execução** por escopo (venda/compra/contrato/projeto/escrow/entrega)
- Evento `OPERATION_BLOCKED` registrado em auditoria imutável
- Policy pronta em `policy_compliance_gate_v1.json`

3) Limites ambientais como motivo de bloqueio

- Schema `environmental_constraints.schema.json`
- Regras de bloqueio por: limite excedido / docs ausentes ou vencidos / geofence fora do CAR / zona restrita

4) Lançamento por foto de relatórios manuais (OCR assistido)

- Tela **"Lançar por Foto (OCR)"** com extração + revisão + confirmação humana
- Integrado ao modelo de evidência e auditoria

Para o **Ciclo+ Agro** ficar "fechado" (pronto para virar produto e operar no mundo real), o que falta agora não é mais conceito — é completar alguns **blocos operacionais**, **governança**, **integrações**, e **robustez**. Aqui vai o checklist objetivo:

1) Cadastros e base operacional mínima

- **Cadastro completo de atores**: produtor, indústria, técnico, transportador, leiloeiro credenciado, laboratório, auditor.
- **Cadastro de propriedades** (já temos CAR/IE) + **unidades internas**: talhões, currais, tanques, galpões, silos.
- **Cadastro de ativos**:
  - animais (individual e lote), culturas (talhão), insumos (SKU/lote/validade), máquinas/equipamentos (tratores/drones).

2) Marketplace e negociação (do jeito que você definiu)

- **Marketplace de insumos** (compra/venda) com:
  - estoque, frete, impostos, NF-e (se aplicável), regras de entrega.
- **Produção sob demanda por contrato**:
  - publicar capacidade, receber demanda, negociar termos e gerar contrato.
- **Leilão (módulo separado)**:
  - só para ativos "não comprometidos"
  - papel "leiloeiro" e regras de credenciamento
  - pós-lance: **escrow obrigatório** + evidência + liberação.

3) Contratos e ciclo produtivo (multi-espécie e multi-cultura)

- Expandir templates além de bovinos:
  - **ovinos/caprinos, aves, suínos, peixes**, e **culturas** (soja, milho, algodão, etc.)
- Para cada um: ciclos, marcos, evidências, tolerâncias, janelas de disputa e tranches.

4) Motor de regras + compliance (falta “parametrizador”)

Você já tem as regras; falta a **tela/portal para parametrizar**:

- limites ambientais por propriedade (atividades permitidas, zonas, docs)
- tolerâncias (peso, mortalidade, rendimento)
- políticas de escrow/garantias por produto e por risco
- lista de bloqueios, desbloqueios e “condições para desbloquear”.

5) Financeiro real (escrow/split de verdade)

Hoje está modelado. Falta conectar com “mundo bancário”:

- integração com **PSP/gateway** para escrow, split, TED/PIX
- conciliação, estornos, chargeback (quando aplicável)
- ledger interno (razão de eventos financeiros) + auditoria.

6) Logística e entrega (cadeia física)

- módulo de **frete/embarque**:
    - emissão de ordem de coleta
    - rastreio, check-in/out com evidência B (GPS+foto)
    - recebimento e divergência com disputa.

7) Módulo técnico (assistência + laudos)

- agenda + aceite do técnico
- checklist por espécie/cultura
- anexos (laudos laboratoriais) com validade e impacto em compliance.

8) IA (do jeito “bom para dev e seguro”)

Não é “IA solta”: é IA **assistida**.

- IA para sugerir preço, risco, contrato e marcos
- IA para extrair campos de voz/OCR
- sempre com **confirmação humana** e log auditável do que foi sugerido vs aceito.

9) Segurança, governança e auditoria (produção)

- RBAC completo (papéis bem definidos)
- trilha imutável (já previsto) + verificador de integridade
- LGPD: retenção, anonimização, consentimentos, trilha de acesso
- antifraude (alertas por padrão anômalo, tentativas inválidas, GPS incoerente etc.)

10) Offline-first “de verdade”

Você já tem o conceito. Falta implementar:

- fila offline por dispositivo
- sincronização, deduplicação (idempotência)
- resolução de conflitos
- telemetria (quando trava, por quê, onde).

Se eu tivesse que escolher as 3 lacunas mais críticas agora

1. **Portal Admin de parametrização** (regras, limites ambientais, políticas, desbloqueio).
2. **Financeiro real** (escrow/split/conciliação).
3. **Templates completos por espécie/cultura** (para cobrir todo agro e virar produto).

Se você disser **qual é o primeiro mercado-alvo** (ex.: bovinos + frigorífico / aves / peixe / soja), eu fecho um **MVP final 1.0** com: módulos exatos, telas faltantes e a ordem de desenvolvimento (sprint a sprint).

Vou escolher com base em **viabilidade de implantação**, **valor imediato** e **aderência ao que você descreveu** (indústria comprando produção sob demanda + escrow + evidência + compliance).

Minha escolha de foco para o 1.0

Bovinos (cria/recria/engorda) + Frigorífico/Indústria + Insumos

Por quê:

- É onde **escrow por marcos** faz mais sentido (ciclo longo, risco alto, necessidade de garantia).
- Evidências **pesagem + visita técnica + entrega** são naturais e "fáceis" de validar.
- Compliance por **CAR/IE + limites ambientais** tem grande valor (reduz risco regulatório e reputacional).
- Gera "efeito rede": produtor + técnico + transportador + indústria + fornecedores de insumos.

Leilão entra como **módulo opcional** do 1.0 (para ativos não comprometidos), mas **não é o coração** do começo.

O que falta para o sistema virar "produto fechado" (na prática)

Vou listar em ordem de prioridade (o que mais trava go-live).

1) Portal Admin para parametrizar regras (CRÍTICO)

Hoje você tem regras excelentes, mas precisa do “painel de controle” para operar no mundo real:

Faltam telas/configurações para:

- Cadastro e validação de **limites ambientais** por propriedade (atividades permitidas, zonas, docs e validade).
- Parametrização de **políticas por produto** (bovinos, aves, etc):
  - escrow obrigatório? % garantia? prazos? tolerâncias?
- Configurar **scopes de bloqueio** (o que bloqueia comercialização vs contrato vs escrow).
- Fluxo de **desbloqueio condicionado** (ex.: acordo assinado + documento renovado + evento do auditor).

✅ Sem isso, o sistema fica “hardcoded” e difícil de escalar para vários clientes/regiões.

2) Financeiro real (escrow/split/conciliação) (CRÍTICO)

Modelamos o escrow, mas falta “virar dinheiro de verdade”:

- integração com PSP/gateway (escrow/split/PIX)
- conciliação (pagou? caiu? estornou?)
- ledger interno simples (razão) baseado em eventos
- regras de liberação por tranche e bloqueio automático em disputa

✅ Sem isso, vira só “gestão” e perde o diferencial principal.

3) Operação de campo completa (offline + evidência) (CRÍTICO)

Você já tem offline, voz e OCR como conceito. Falta transformar em “industrial”:

- fila offline por dispositivo (PENDING/SYNCED/REJECTED)
- resolução de conflitos
- idempotência real
- verificação de GPS + geofence por CAR
- modo “operador” simplificado (modo fácil que fizemos)

✅ Sem isso, o operador trava e o lastro não é confiável.

4) Templates completos de ciclos (para cobrir o agro todo)

Para virar “setor inteiro”, precisamos de biblioteca de templates:

- bovinos (já iniciado) ✅
- aves, suínos, peixes, ovinos/caprinos

- culturas (soja/milho etc.) com marcos: plantio, tratos, colheita, laudos

✅ Isso é o que expande o produto. Não impede o go-live com bovinos, mas impede escalar "para todos".

5) Logística e entrega (bem simples, mas essencial)

- ordem de embarque/coleta
- check-in/out com evidência B
- recebimento + divergência (disputa)

6) Governança e papéis (RBAC completo)

- perfis prontos: produtor, indústria, técnico, laboratório, transportador, auditor, leiloeiro
- trilhas read-only para fiscal/auditor
- trilha de acesso (quem viu o quê)

7) IA do jeito certo (assistida, não "mágica")

- IA para sugerir contrato/marcos/risco
- IA para voz/OCR
- sempre com confirmação humana + registro do que foi aceito

Meu plano de desenvolvimento (sequência ideal)

Sprint 1 — Go-live mínimo que funciona

1. Propriedade CAR/IE + geofence
2. Contrato bovinos + escrow por marcos
3. Evidência B para pesagem/entrega + auditoria imutável
4. Estoque de insumos (baixa) + vínculo ao contrato

Sprint 2 — Compliance e bloqueios operacionais (o diferencial)

5. Compliance gate por imóvel (bloqueio por default e ambiental)
6. Portal Admin para regras, docs e limites ambientais
7. Tela de bloqueio com "o que fazer" + solicitação de auditor

Sprint 3 — Financeiro real

8. Integração PSP + conciliação
9. Liberação tranche automática + disputa congela escrow

Sprint 4 — Escala e cobertura do setor

10. Templates para aves/suínos/peixes/culturas
11. Marketplace insumos completo + logística básica
12. Leilão como módulo (ativos não comprometidos)

Decisão final (minha escolha)

O sistema já está "bem desenhado". O que falta para ficar completo é:

1. **Portal Admin de parametrização** (regras/limites/docs/desbloqueio)
2. **Financeiro real integrado** (escrow/split/conciliação)
3. **Operação offline + evidência "industrial"** (fila, idempotência, geofence CAR)
4. **Templates multi-espécie/cultura** para escalar

Se você quiser, eu posso agora:

- definir exatamente o **Módulo Admin** (telas + campos + regras) e
- gerar mais um pacote com **as telas do portal web** (acessível também) + schemas/policies atualizados.

Fechado — eu segui o caminho **mais "completador" do sistema**: criei o **Portal Admin (Web)** para parametrizar regras, compliance, limites ambientais, documentos, RBAC e credenciamentos. Isso é o que mais faltava para o Ciclo+ virar **produto operável** (sem ficar tudo travado em código).

✅ **Baixar o pacote v4 (Portal Admin + telas web + schemas + políticas + backlog):**
Download CicloPlus_Portal_Admin_v4.zip

O que tem dentro

🖥️ Telas Web (mockups) em `images/web/`

- `01_admin_dashboard.png` – visão geral + alertas + fila de auditor
- `02_properties_list.png` – lista de propriedades (CAR/IE) + status + ações
- `03_property_detail_compliance.png` – detalhe do imóvel + escopos bloqueados + docs
- `04_environmental_limits.png` – limites ambientais + simulação rápida (BLOCK/ALLOW)
- `05_policy_editor.png` – editor de políticas (formulário) + versionamento
- `06_policy_simulator.png` – simulador de regras (teste controlado)
- `07_rbac_roles.png` – papéis e permissões (RBAC)
- `08_credentialing.png` – credenciamento de técnico/lab/leiloeiro

📄 Documentação em `docs/`

- `ADMIN_FEATURES.md` – tudo que o portal faz (compliance, regras, RBAC, auditoria)

- `DEV_BACKLOG_SPRINTS.md` – ordem ideal de desenvolvimento (sprint a sprint)
- `README.md` – visão geral do pacote

🧩 Schemas e políticas em `code/`

- `schemas/` (policy definition, RBAC role, credentialing, environmental docs)
- `policies/policy_compliance_gate_v2.json` (exemplo publicado)
- `ui/admin_screens.json` (mapa de rotas)

Resultado prático

Agora o sistema fica **"completo para operar"** porque:

- você consegue **definir e alterar regras** sem mexer no código,
- consegue **bloquear/desbloquear imóveis** (CAR/IE) com motivo, escopo e condição,
- consegue **controlar limites ambientais e validade de docs**,
- e tem **governança** (RBAC + credenciamento) para ambiente real.

Se você quiser, o próximo melhor passo (e eu já faço) é gerar um pacote **v5 Financeiro** com:

- ledger (razão) por eventos,
- conciliação,
- tranches automáticas,
- e fluxo de disputa congelando escrow + decisão do auditor.

Falta sim — mas agora falta **módulos "de produção"**, não mais o núcleo. O núcleo (eventos + evidência + auditoria + máquina de estados + escrow + compliance + portal admin) está bem fechado. O que falta são blocos para **operar 100% do dia a dia** e **escalar para todo o agro**.

Vou listar por **prioridade prática** (o que mais faz falta para rodar de verdade).

Módulos que ainda faltam (prioridade alta)

1) Financeiro real integrado (Escrow/Split/PIX + conciliação)

Hoje está modelado, mas falta:

- integração com PSP/bancos (PIX, split, repasse)
- conciliação (pagou/confirmou/estorno)
- ledger interno (razão) para auditoria financeira
- reembolso/chargeback/retenções

✅ Esse é o "coração" do seu diferencial.

2) Logística / Transporte / Entrega completa

Falta um módulo para:

- ordem de coleta/embarque
- rastreio (check-in/out com evidência B)
- divergência no recebimento (abre disputa automaticamente)
- integração com transportadoras ou fluxo interno

3) Emissão fiscal e documentos comerciais (se aplicável)

Dependendo do modelo:

- NF-e/NFS-e, CT-e, MDF-e (quando necessário)
- contratos anexos e comprovantes
- padronização por estado/município

4) Gestão de ativos e produção "completos"

Você já cobre bovinos no template, mas falta "biblioteca" completa:

- aves, suínos, peixes, ovinos/caprinos
- culturas (soja/milho etc.) com marcos e evidências próprias
- rastreabilidade por lote e eventos sanitários

Módulos importantes (prioridade média)

5) CRM/Relacionamento + cadastro avançado de participantes

- onboarding de produtor/indústria/técnico/lab
- KYC/validação (documentos, credenciamento)
- reputação/score operacional (não financeiro)

6) Analytics e relatórios

- performance por ciclo (KPIs, perdas, ganhos)
- compliance dashboard (docs vencendo, bloqueios)
- auditoria com exportação (relatórios)

7) Notificações e alertas

- vencimento de marcos, documentos, janela de disputa
- alertas de risco (fraude, GPS inconsistente, repetição de tentativas inválidas)

Módulos opcionais (mas valiosos)

8) Integrações com máquinas e IoT

- balança Bluetooth / leitores RFID
- tratores/telemetria
- drones (mapas, evidências, inspeção)
- sensores (temperatura, umidade, tanque, silos)

9) Motor de IA (assistido)

- sugestão de contrato e parâmetros
- OCR/voz mais avançado
- previsão de risco e "próximo melhor passo"
  (sempre com confirmação humana + auditoria do que foi sugerido/aceito)

10) Leilão completo

- regras do leiloeiro, lances, antifraude
- escrow automático pós-lance vencido
- disputa específica de leilão

Minha análise final

✅ **O sistema já tem a "espinha dorsal" completa.**
📌 Para ficar "completo como produto", faltam principalmente **3 módulos**:

1. **Financeiro real integrado** (escrow/split/PIX + conciliação + ledger)
2. **Logística/entrega completa** (coleta, rastreio, divergência)
3. **Biblioteca de ciclos/templates multi-espécie e culturas** (para cobrir todo agro)

Se você disser "quero fechar o produto para go-live agora", eu recomendaria começar pelo **Financeiro**. Se disser "quero expandir o agro todo", começo pelos **templates** (espécies/culturas) junto com logística básica.

Beleza — para "agro todo" a estratégia vencedora é **transformar o Ciclo+ em uma fábrica de templates** (espécies/culturas) + um "meta-modelo" único de ciclo produtivo. Assim você não cria um sistema diferente para cada cadeia; você só **pluga um template**.

Estrutura que eu recomendo para escalar para TODO o agro

1) Um "Meta-Modelo" único (serve para qualquer cadeia)

Todo produto agro (animal ou cultura) vira:

- **Ciclo** (fases) → **Marcos** (checkpoints) → **Evidências** (A/B/B_LAB) → **Regras** → **Tranches** (escrow) → **Auditoria**

E cada template define:

- fases (ex.: plantio → tratos → colheita) ou (cria → recria → engorda)
- quais marcos liberam quais percentuais do pagamento
- quais evidências são obrigatórias
- tolerâncias e limites (peso, mortalidade, produtividade, prazos)
- quais laudos/documentos podem ser exigidos

✅ Isso permite atender "tudo" sem reescrever o core.

Biblioteca de Templates (Agro todo) — pacote recomendado

A) Animais de produção

Bovinos (carne e leite)

- Cria / Recria / Engorda (carne)
- Reprodução (matrizes/IA)
- Leite (produção mensal + qualidade)

Evidências típicas

- RFID/Brinco + pesagens (B)
- visita técnica (B)
- GTA/NF/romaneio (B_LAB/documental)
- entrega (B com geofence)

Aves (corte e postura)

- Lote 42 dias (corte): alojamento → vacinação → ganho → abate
- Postura: início → pico → manutenção → descarte

Evidências

- contagem/entrada de pintos (A+foto)
- mortalidade diária (A)
- laudo sanitário (B_LAB)
- entrega (B)

Suínos

- creche → crescimento → terminação
- marcos por pesagens e conversão alimentar

Peixes (piscicultura)

- povoamento → engorda → biometria → despesca
- qualidade de água (sensores/laudos)
- mortalidade e conversão alimentar

Ovinos/Caprinos

- reprodução → crescimento → terminação
- laudos e manejo sanitário

B) Culturas (grãos, fibras e perenes)

Soja / Milho / Algodão / Arroz / Feijão

- planejamento → plantio → tratos → colheita → armazenagem → entrega

Evidências

- nota/registro de compra de insumo (A)
- aplicação (A ou B conforme risco/valor)
- imagens/drones (B)
- laudo (B_LAB: pragas, solo, umidade, micotoxinas)
- colheita (B com geofence)
- armazenagem (B: silo, umidade)
- entrega (B)

Cana / Café / Citrus (perenes)

- manejo e colheita com marcos por safra (ciclos anuais)

Marketplace "Agro todo"

Para escalar, o marketplace precisa de **3 sub-marketplaces**:

1. **Insumos** (compra/venda)

- sementes, ração, fertilizantes, defensivos, medicamentos, equipamentos
- baixa de estoque integrada + evidência

2. **Produção sob demanda (contratos)**

- produtor publica **capacidade** (por ciclo)
- indústria publica **demanda**
- sistema sugere contrato por template + IA assistida

3. **Leilão (opcional, módulo separado)**

- apenas ativos não comprometidos
- leiloeiro credenciado
- pós-lance: escrow obrigatório

O que precisa ser desenvolvido para "Agro todo" (roadmap objetivo)

Fase 1 — Templates + Cadastros universais (base para tudo)

- "Cadastro universal" de produto:
  - `species/cropType`, unidade, lote/identificação, ciclo padrão
- Biblioteca inicial de 10–12 templates:
  - bovinos, aves, suínos, peixes, ovinos/caprinos
  - soja, milho, algodão, arroz/feijão (escolher)
- Portal Admin para criar/editar templates (igual políticas)

Fase 2 — Evidências especializadas por cadeia

- RFID/brinco animal + pesagem (integração simples)
- modo de contagem lote (aves)
- biometria (peixes)
- tratos culturais com evidência (aplicação)
- importação por OCR/voz (papel) para qualquer template

Fase 3 — Logística e armazenagem multi-cadeia

- embarque/coleta/recebimento com divergência
- armazenagem (silo/câmara fria) com medição e evidência

Fase 4 — Financeiro real e escala

- escrow/split/PIX + conciliação
- liberação tranche automática por marco
- antifraude + auditoria avançada

Minha escolha de "pacote Agro todo" para você começar agora

Se você quer expansão máxima, eu recomendo iniciar com **6 templates** que cobrem a maior parte do agro e geram tração:

1. **Bovinos carne** (cria/recria/engorda)
2. **Aves corte** (lote 42 dias)
3. **Suínos** (creche/terminação)
4. **Peixes** (povoamento/biometria/despesca)
5. **Soja** (plantio/tratos/colheita/armazenagem)

6. **Milho** (plantio/tratos/colheita/armazenagem)

Com isso você atende: proteína animal + grãos (o grosso do Brasil).

Se você quiser, eu faço o próximo passo **agora** no mesmo padrão dos pacotes anteriores:

- gerar um **ZIP "Templates Agro"** com:
  - JSON de cada template (fases, marcos, tranches, evidências, tolerâncias)
  - máquinas de estado Mermaid
  - telas mobile mínimas por template (modo fácil)
  - regras do motor específicas (ex.: mortalidade, produtividade, laudo obrigatório)

E eu já deixo "plugável" no Portal Admin (como catálogo de templates).

Incluído ✅ (com silvicultura também)

📦 **Baixar o pacote atualizado (v2 Expandido):**
[Download CicloPlus_Templates_Agro_Package_v2_Expandido.zip]

O que entrou neste v2

Além dos 6 do v1 (bovinos, aves, suínos, peixes, soja, milho), adicionei:

Animais

- `TPL_OVINOS_CAPRINOS_V1` — ovinos/caprinos (reprodução/crescimento/terminação)
- `TPL_LEITE_V1` — leite (coletas recorrentes + laudo/qualidade)

Culturas

- `TPL_ARROZ_V1` — arroz
- `TPL_FEIJAO_V1` — feijão
- `TPL_ALGODAO_V1` — algodão (inclui beneficiamento)
- `TPL_CANA_V1` — cana (inclui qualidade/Brix)
- `TPL_CAFE_V1` — café (manejo/colheita/pós-colheita)

Silvicultura (inclui palmeiras/açaí e "chacharu")

- `TPL_SILVICULTURA_GENERIC_V1` — **Silvicultura (ciclo longo)**: implantação → pegamento → manejo anual → colheita → entrega
  - Evidência forte (Tipo B) com **geofence + mídia** e laudos quando necessário.

O que vem junto (igual ao padrão)

- JSON dos templates em `code/templates/`
- Mermaid das máquinas de estado em `mermaid/`
- Políticas de tolerância em `code/policies/`
- Telas Mobile "Modo Fácil" para cada template em `images/mobile/`
- Documentação específica de silvicultura em `docs/SILVICULTURA_NOTAS.md`

Se você me disser **quais espécies de silvicultura você quer priorizar** (ex.: açaí, dendê/palma, eucalipto, teca etc.), eu gero um **v3 com templates específicos por espécie**, com marcos e tolerâncias próprios (ex.: produção recorrente vs corte único, janela de colheita, evidências por safra/ano).

Perfeito — são **dois problemas clássicos** no agro, e dá pra resolver dentro do seu núcleo (contrato por ciclo + regras + evidência + escrow + auditoria), sem "inventar moda".

Vou propor a solução já no formato **operacional para embutir no sistema**.

---

1) Leite: acabar com a "surpresa" do preço no recebimento

Causa do problema

No leite, o preço final costuma depender de:

- **índices de qualidade** (gordura, proteína, CCS/CBT, crioscopia etc.)
- **bonificações/penalidades**
- às vezes **frete**, perdas, tabela do laticínio e "média do mês"
- e o produtor só descobre tudo no fechamento.

Solução Ciclo+: Preço Travado + Fórmula Transparente + Escrow de Qualidade

Crie um "Contrato de Leite" com **duas camadas de preço**:

A) Preço base mínimo garantido (Piso)

- A indústria deposita no **escrow** um valor mínimo por litro (ou por faixa), *antes* da coleta.
- Isso garante: produtor **nunca recebe abaixo do piso**.

B) Preço final = piso + ajuste automático por fórmula auditável

O contrato inclui uma **fórmula pública** (tabela) que determina:

- faixas de gordura/proteína
- CCS/CBT (penaliza ou bonifica)

- bônus por volume, constância, sazonalidade
- desconto por frete (se aplicável)

✅ A regra fica no **motor de regras** e é simulável no Portal Admin.

Como fica o fluxo mensal (sem surpresa)

Para cada coleta (ou lote semanal):

1. **Coleta**: evidência B (GPS + foto/vídeo do medidor/volumetria + assinatura digital do coletor)
2. **Amostra**: cria evento `SAMPLE_SEALED` com:
   - QR do frasco + lacre + foto
3. **Laboratório**: evento `LAB_RESULT_POSTED` (B_LAB):
   - laudo anexado + hash + assinatura
4. **Cálculo automático do preço**:
   - o sistema calcula o preço final pela fórmula e mostra:
     - "Você vai receber: R$ X"
     - "Base: R$ piso + bônus – penalidades"
5. **Liberação do escrow em 2 partes (tranches)**:
   - **Tranche 1 (imediata)**: libera o *piso* após coleta confirmada
   - **Tranche 2 (após laudo)**: libera bônus/ajustes quando o laboratório lançar resultado

📌 Se o laudo atrasar, o produtor já recebeu o piso.
📌 Se der disputa, congela só a tranche 2.

Componentes para garantir "não manipulação" do preço

- **Contrato bloqueia alterações da fórmula** dentro do ciclo (só vale para o próximo ciclo).
- Se a indústria quiser mudar tabela, tem que:
  - publicar versão futura no Admin (`PUBLISHED_NEXT_CYCLE`)
  - produtor assina eletronicamente a nova versão.
- O produtor pode simular preço "no dia" com base no piso e qualidade esperada.

✅ Resultado: o produtor sabe o piso e entende exatamente como o preço final será formado.

---

2) Boitel / terminação em parceria: segurança total do dono do ativo

Aqui o medo do dono do gado é:

- troca/substituição de animais
- perda de peso / manejo ruim
- mortalidade e sumiço
- uso indevido do animal como garantia de terceiros
- venda sem autorização
- “confusão” de titularidade no boitel.

Solução Ciclo+: Custódia Contratual + Cadeia de Evidências + Escrow bilateral + Trava de Titularidade

Implementar um “Contrato de Custódia/Terminação (Boitel)” com 4 travas técnicas:

1) Titularidade imutável + trava de alienação

- Animal (ID RFID/Brinco) fica com `ownerOrgId` (dono).
- Boitel recebe apenas `custodianOrgId` (custodiante).
- O sistema bloqueia qualquer:
  - “venda”, “transferência”, “leilão”, “penhora”, “saída” sem evento `OWNER_AUTHORIZATION_SIGNED`.

✅ Isso é RBAC + máquina de estados + filtro orgId funcionando forte.

2) Check-in de entrada com prova forte

Evento `ASSET_CHECKIN` (Tipo B):

- vídeo/fotos do animal + brinco legível
- peso inicial
- geofence do boitel
- assinatura do responsável do boitel e do transportador (ou do próprio dono)

Isso cria o “marco zero” incontestável.

3) Manejo como eventos com evidência (sem buraco)

- alimentação (estoque + baixa)
- medicação
- pesagens periódicas
- mortalidade (se ocorrer) com laudo/foto e registro imediato
  Tudo vira evento auditável.

4) Escrow bilateral com liberação por marcos

Para o boitel:

- recebe por **tranches** conforme:
  - animal mantido + ganho de peso mínimo + evidência de pesagem
    Para o dono do animal:
- o sistema retém pagamentos/bonificações até entrega final com evidência

E se houver divergência:

- abre `DISPUTE_OPENED`, congela o escrow, auditor emite parecer read-only.

Regras de proteção máxima do dono do ativo (o "modo blindado")

- **Proibição técnica** de:
  - "substituir animal" (só pode com evento `REPLACEMENT_REQUEST` + assinatura do dono)
  - "misturar lote sem rastreio" (cada entrada gera lote com IDs fixos)
- **Alerta antifraude**:
  - brinco repetido, pesagem incoerente, GPS fora, tentativa de evento inválido
- **Seguro opcional**:
  - se acionado, evento e payout também passam pelo motor.

✅ Resultado: o boitel opera, mas **não consegue alienar nem "sumir" com ativo** sem deixar rastro e sem bloqueio automático.

O que eu faço agora no seu pacote (para ficar implementável)

Eu consigo gerar um **v3 dos templates** com:

1. **TPL_LEITE_V2** com:
   - piso garantido + fórmula auditável + tranches
   - eventos `SAMPLE_SEALED` e `LAB_RESULT_POSTED`
2. **TPL_BOITEL_CUSTODIA_V1** com:
   - trava de titularidade + check-in/out + pesagens + escrow por ganho

E criar as **políticas**:

- `POLICY_MILK_PRICE_FORMULA_LOCK_V1`
- `POLICY_BOITEL_OWNER_PROTECTION_V1`

Se você quiser uma decisão automática do sistema, me diga apenas:

- o leite vai ser **pago por coleta** (diário/semanal) ou **fechamento mensal**?
  Se não quiser decidir agora, eu implemento no template como **mensal + opção semanal** (configurável no Admin).

que eu recomendo implementar no sistema (objetivo)

1. **Novo tipo de contrato**: `LEITE_GARANTIA_ROLANTE`
2. Campos:
   - `rollingWindowDays` (7–15)

    - `minEscrowBalanceFormula`
    - `payoutSchedule` (D+1 a D+3)
    - `defaultTriggers` (saldo < mínimo, atraso > X dias)
3. Regras:
    - `BLOCK_COLLECTION_IF_ESCROW_BELOW_MIN`
    - `AUTO_PAYOUT_BASE_AFTER_COLLECTION`
    - `FREEZE_ADJUSTMENT_ON_LAB_DISPUTE`

Camada 1 — Bloqueio operacional total dentro do Ciclo+

"Asset Lock" (trava de alienação)

Para qualquer ativo do comprador (ex.: gado, grãos, máquinas, recebíveis), você cria um status:

- `FREE` (livre)
- `LOCKED_AS_GUARANTEE` (travado como garantia)
- `IN_DISPUTE` (congelado)
- `DEFAULT_EXECUTION` (execução de garantia)
- `RELEASED` (liberado)

Quando estiver `LOCKED_AS_GUARANTEE`, o núcleo bloqueia automaticamente:

- listar no marketplace/leilão
- transferir titularidade
- criar sub-garantias
- retirar da custódia (boitel, armazém, silo)
- qualquer evento crítico (máquina de estados + motor de regras)

Tudo isso vira evento auditável (hash + evidência).

✅ Isso impede o "golpe" **dentro da operação do Ciclo+**.

Camada 2 — Força jurídica fora do sistema (recomendado)

Aqui é onde você realmente impede "posterior venda/transferência/penhora" **na prática**, dependendo do tipo de ativo.

O Ciclo+ pode gerar e exigir (como condição do contrato) uma **garantia formal** e registrar/lastrear:

Opções típicas (depende do ativo)

- **Recebíveis**: cessão fiduciária / trava de recebíveis (com comprovação e fluxo)
- **Animais / produção**: penhor/garantia sobre lote + rastreio e custódia (quando aplicável)
- **Grãos armazenados**: warrant/depósito/controle de armazém (custódia + comprovação)

- **Máquinas/veículos**: alienação/gravame em instrumentos adequados (quando aplicável)
- **Seguro garantia / fiança / carta de crédito**: alternativa quando travar ativos é difícil

✅ O sistema não "faz a justiça", mas **obriga a indústria a entregar uma garantia executável** e **não deixa operar sem isso**.

Sobre "impedir penhora"

Penhora é ato judicial — **nenhum software consegue impedir uma decisão judicial**.

O que o Ciclo+ pode fazer (e isso é valioso):

- exigir garantias que **reduzam a chance** de penhora atrapalhar o produtor
- registrar prioridade/ônus (quando aplicável)
- monitorar inadimplência e **bloquear cedo**
- manter prova técnica forte para execução contratual rápida
- congelar recursos (escrow) e travar operações para evitar "dilapidação"

Como eu desenharia isso no Ciclo+ (bem implementável)

Módulo: Garantias & Travas

1. Cadastro de ativos do comprador (com provas):
    - tipo, identificador, localização/custódia, documentos, valor
2. Regra: **Contrato só assina se garantias mínimas estiverem travadas**
3. Engine de bloqueio:
    - `BLOCK_SELL_IF_LOCKED`
    - `BLOCK_TRANSFER_IF_LOCKED`
    - `BLOCK_PLEDGE_IF_LOCKED`
4. Execução por inadimplência:
    - entra em `DEFAULT_EXECUTION`
    - aplica consequências automáticas: bloqueio de CAR/IE no seu sistema + reputação + congelamento + acionamento de garantias (workflow)
5. Evidência e auditoria:
    - toda trava e liberação gera evento com hash e assinatura

---

Resposta direta

Sim, o Ciclo+ pode bloquear ativos do comprador **para impedir venda/transferência/"penhora interna" dentro da plataforma**.
Para impedir **fora da plataforma**, o ideal é o sistema exigir **garantias formalizáveis** (recebíveis, custódia, seguro garantia, instrumentos jurídicos) e **não permitir operação sem isso**.

Se você me disser **quais ativos do comprador você quer travar** (recebíveis? máquinas? gado? imóveis? estoque?), eu ajusto o modelo ideal e eu atualizo os templates/policies para:

- "Garantia mínima por demanda"
- "Asset Lock automático"
- "Execução por default" (com bloqueio operacional total)

A seguir está o **mapa completo dos módulos** (arquitetura funcional) para o Ciclo+ virar um sistema **agro 360** (animais, culturas e silvicultura), com **mercado sob demanda**, **boitel**, **leite anti-calote**, **travas de ativos/garantias**, **compliance CAR/ambiental**, **auditoria imutável**, **voz/OCR** e **escrow**.

Vou organizar por camadas e também indicar quais são **obrigatórios** vs **opcionais**, e como eles se conectam ao seu núcleo patenteável.

---

0) Núcleo patenteável (já definido)

CORE (imutável)

- Autenticação
- RBAC / permissões
- Filtro orgId/tenantId
- Motor de regras (pré-execução)
- Máquina de estados
- Evidência obrigatória (A/B/B_LAB)
- Auditoria imutável (append-only + hashes encadeados)
- Liquidação condicionada (escrow/tranches/split)
- Interface read-only de auditoria/fiscalização

Esse núcleo é "o motor" e sustenta tudo abaixo.

---

1) Cadastros mestres e identidade agro (obrigatório)

1.1 Cadastro de participantes

- Produtor / Propriedade rural
- Indústria (frigorífico, laticínio, cerealista, usina)
- Técnico credenciado
- Laboratório
- Transportador
- Boitel / confinamento / armazém
- Leiloeiro (quando existir)
- Auditor/fiscal (read-only)

1.2 Cadastro de propriedade e ambiente

- Vínculo **CAR**
- Inscrição estadual / estadualização
- Geofence por perímetro do CAR e unidades internas (talhão, curral, tanque, galpão, silo)
- Documentos ambientais e validade (licenças, autorizações)
- Limites ambientais paramétricos (atividades permitidas, áreas, max heads, etc.)

---

2) Gestão de ativos agro (obrigatório)

2.1 Ativos animais

- Individual (RFID/brinco) e por lote
- Pesagens, eventos sanitários, mortalidade, reprodução
- Custódia (quem é dono vs custodiante)
- Controle de saída/transferência

2.2 Ativos de culturas

- Talhão/lote de produção
- Operações: plantio, tratos, colheita, armazenagem, entrega

2.3 Silvicultura (ciclo longo)

- Implantação, pegamento, manejo anual, colheita
- Produção recorrente (ex.: palmeiras/açaí) e corte único (madeira/biomassa)

2.4 Ativos de insumos

- SKU/lote/validade
- Estoque por local (fazenda, boitel, armazém)

2.5 Ativos de máquinas/equipamentos (opcional, recomendado)

- tratores, implementos, drones
- vínculo com telemetria/OS

---

3) Templates de ciclos (obrigatório para “agro todo”)

Catálogo de ciclos plugável (o que você já começou):

- bovinos, leite, aves, suínos, peixes, ovinos/caprinos
- soja, milho, arroz, feijão, algodão, cana, café
- silvicultura (palmeiras/açaí/chacharu e outras)

Cada template define:

- fases → marcos → evidências → tranches → tolerâncias → disputa

---

4) Contratos e negociação (obrigatório)

4.1 Produção sob demanda por contrato (principal)

- Produtor publica capacidade (por ciclo/template)
- Indústria publica demanda
- Negociação direta (chat, termos, anexos)
- Contrato gerado por:
  - padrão pré-definido (recomendado)
  - IA assistida (opcional, com confirmação)
  - técnico vinculado (opcional)

4.2 Contrato de boitel / custódia (obrigatório se operar boitel)

- separa **ownerOrgId** (dono) de **custodianOrgId** (boitel)
- check-in/out com evidência B
- pesagens por marco
- pagamento por ganho/etapas
- bloqueios de alienação sem autorização do dono

4.3 Contrato leite anti-calote (obrigatório se operar leite)

- garantia rolante (escrow mínimo)
- pagamento de piso por coleta (D+1 a D+3)
- ajuste por laudo (tranche 2)
- bloqueio automático de coleta se garantia insuficiente

---

5) Garantias e travas de ativos (obrigatório para segurança jurídica)

Módulo "Garantias & Travas"

- Cadastro de garantias do comprador (recebíveis, estoque, contrato, seguro garantia, etc.)
- Status de trava: FREE / LOCKED_AS_GUARANTEE / IN_DISPUTE / DEFAULT_EXECUTION / RELEASED
- Bloqueios operacionais:
  - impedir venda/transferência/listagem/novas garantias dentro do Ciclo+
- Workflow de execução:
  - default → congela escrow → aciona garantias → auditor decide

📌 Observação jurídica: fora do sistema, você ainda precisa do instrumento formal (ex.: seguro garantia, trava de recebíveis, custódia), mas o Ciclo+ garante que **ninguém opera sem a garantia ativa**.

---

6) Financeiro e liquidação (obrigatório)

- Escrow por contrato
- Split automático
- Tranches por marcos
- Congelamento em disputa
- Ledger (razão) interno por eventos
- Conciliação (PIX/transferências/estornos)
- Cobrança e cura de inadimplência

---

7) Marketplace (obrigatório)

7.1 Marketplace de insumos (principal)

- compra/venda
- estoque integrado (baixa por evento)
- cotação e pedidos
- entrega/recebimento com evidência

7.2 Marketplace de produção sob demanda (contratos)

- catálogo de ciclos e capacidades
- matching indústria ↔ produtor
- propostas e assinatura

7.3 Leilão (opcional, módulo separado)

- somente ativos não comprometidos
- leiloeiro credenciado
- pós-lance: escrow obrigatório

---

8) Operação em campo (obrigatório)

8.1 App Mobile “Modo Fácil” (acessível)

- fontes grandes, botões grandes, fluxo passo a passo

8.2 Voz e OCR de papel

- comando por voz com confirmação humana
- lançamento por foto de relatório (OCR assistido)
- evidência sempre anexada com hash

8.3 Offline-first real

- fila local, sincronização, idempotência, conflitos

---

9) Logística, armazenagem e entrega (obrigatório)

- ordem de coleta/embarque
- check-in/out (GPS + foto) por geofence
- recebimento com divergência e disputa
- armazenagem (silo/câmara) com eventos e evidência

---

10) Qualidade, laboratório e rastreabilidade (obrigatório)

- amostra lacrada (QR + evidência)
- laudos B_LAB (assinados)
- regras automáticas: bloqueio/penalidade/bonificação
- histórico rastreável do lote

---

11) Compliance, ambiental e bloqueios por imóvel (obrigatório)

- compliance gate por CAR/IE
- limites ambientais por atividade
- documentos vencendo → alertas → bloqueio
- bloqueios por inadimplência (efeito “anti-calote”)

---

12) Auditoria e fiscalização read-only (obrigatório)

- timeline de eventos
- verificador de integridade (hash chain)
- parecer do auditor (evento)
- exportação de relatórios

---

13) Portal Admin (obrigatório)

Você já tem a base:

- parametrizar políticas e publicar versões
- parametrizar templates (próxima evolução)
- RBAC e credenciamentos
- compliance por imóvel
- simulador de regras

---

14) Inteligência/IA assistida (opcional, recomendado)

- sugestão de contrato e parâmetros por template
- previsão de risco (default, sanidade, produtividade)
- recomendação de ação corretiva
- OCR/voz mais avançado
  (sempre com confirmação humana + registro auditável)

Lista final: "Módulos necessários" (checklist rápido)

Obrigatórios para agro todo:

1. Cadastros e propriedade (CAR/IE + geofence)
2. Gestão de ativos (animais, culturas, silvicultura, insumos)
3. Templates de ciclos (catálogo)
4. Contratos sob demanda + assinatura
5. Garantias & travas de ativos (anti-calote/boitel)
6. Financeiro (escrow/tranches/split + ledger + conciliação)
7. Marketplace de insumos + produção sob demanda
8. Operação em campo (modo fácil + voz + OCR + offline)
9. Logística/armazenagem/entrega + disputa
10. Laboratório/qualidade (B_LAB)
11. Compliance ambiental + bloqueios por imóvel
12. Auditoria read-only + relatórios
13. Portal Admin (políticas, limites, RBAC, credenciamento)

Opcionais (mas fortes):

- Leilão (com leiloeiro)
- Integração IoT/drones/telemetria
- IA assistida

Se você quiser, eu preparo o **próximo ZIP "Arquitetura Completa"** com:

- diagrama master (Mermaid) de todos os módulos
- tabela de entidades (schemas) por módulo
- lista de eventos (event catalog) por cadeia
- backlog sprint a sprint (MVP → escala)
- telas principais (web admin + mobile operador + auditor)

E deixo tudo organizado do jeito que você vem pedindo (pastas por módulo + docs + regras + imagens).

Sim — dá para "blindar" isso no Ciclo+ com um **trilho técnico obrigatório** que impede a propriedade/custodiante de emitir NF (ou qualquer documento de saída) e de mover o ativo **enquanto não houver**: (1) verificação do dono, (2) escrow + split preparado, (3) liberação formal do dono. Isso mata a fraude clássica de "vender ativo de terceiros como se fosse próprio".

Abaixo está o desenho **operacional + regras** para embutir no sistema.

Módulo novo: Custódia com Trava Fiscal e Liquidação Obrigatória

Ideia central

Quando um ativo de terceiro entra em custódia na propriedade (boitel, armazém, fazenda, silo etc.), ele fica com status **"CUSTODY_LOCK"**.
Enquanto estiver assim, o sistema:

1. **bloqueia a emissão de NF/romaneio/saída** pela propriedade/custodiante;
2. só permite "NF de saída" se o fluxo **Escrow+Split+Autorização do dono** estiver concluído;
3. só permite "movimentação física" (check-out/expedição) após a **NF liberada** e a **liquidação do split**.

---

1) Estados obrigatórios do ativo (máquina de estados)

Para cada ativo/lote em custódia:

- `INBOUND_PENDING` (entrada iniciada)
- `IN_CUSTODY_LOCKED` (custódia travada – não vende/transferir/emitir NF)
- `SALE_INTENT_CREATED` (intenção de venda criada)
- `ESCROW_READY` (escrow depositado / garantias ok)
- `OWNER_RELEASE_SIGNED` (dono autorizou saída)
- `NF_AUTHORIZED` (NF/saída autorizada pelo sistema)
- `CHECKOUT_CONFIRMED` (saída confirmada com evidência)
- `SPLIT_SETTLED` (split liquidado e registrado)
- `CLOSED`

Regra: nenhum salto de estado acontece sem evidência e auditoria.

---

2) Fluxo anti-fraude na prática (passo a passo)

A) Entrada em custódia

Evento: `ASSET_CHECKIN` (Tipo B)

- foto/vídeo + GPS geofence + IDs (RFID/brinco/lote) + peso/quantidade → estado vira `IN_CUSTODY_LOCKED`

✅ Aqui o sistema "prende" o ativo no custodiante, mas com dono definido (`ownerOrgId`).

B) Intenção de venda / saída

Quando a propriedade quiser vender ou expedir:

Evento: `SALE_INTENT_CREATED`
Obrigatório informar:

- comprador
- preço/condições
- tipo de documento (NF-e, romaneio etc.)
- regras de split (percentuais + destino)

A partir daqui:

- a NF ainda está **BLOQUEADA**
- o ativo ainda não pode ser movido

C) Escrow + Split armados antes da NF

Evento: `ESCROW_FUNDED` (ou `PAYMENT_GUARANTEE_LOCKED`)
O sistema exige:

- valor mínimo em escrow >= valor do contrato (ou piso)
- regras de split configuradas (quem recebe o quê)

✅ Sem isso, **não libera nota**.

D) Autorização do dono (anti-“venda sem permissão”)

Evento: `OWNER_RELEASE_SIGNED` (assinatura digital do dono)
O dono aprova:

- preço final
- comprador
- split
- data e local de saída

✅ Sem assinatura do dono, não tem NF e não tem saída.

E) Liberação de NF e saída

Evento: `NF_AUTHORIZED`
O sistema só emite/autoriza NF se:

- escrow ok
- split pronto
- dono assinou
- compliance ok (CAR/IE, geofence, documentos)

Saída física: `ASSET_CHECKOUT` (Tipo B)

- foto/vídeo + GPS + conferência + assinatura do responsável

F) Liquidação do Split (obrigatória)

Evento: `SPLIT_SETTLED`

- assim que o pagamento entra (ou é confirmado), o sistema executa:
  - split automático: dono recebe sua parte direto
  - custodiante recebe sua parte (diária/serviço/boitel)
  - impostos/taxas se houver
- tudo auditável (hash + timestamps)

✅ Se não liquidou split, o sistema:

- pode bloquear baixa definitiva,
- abrir disputa,
- e travar novas saídas do mesmo custodiante.

---

3) Regras do motor (policies) que você quer

Estas são as regras “de ferro” (técnicas) — elas impedem a fraude na origem:

Regra 1 — Bloquear emissão de NF se ativo estiver em custódia de terceiro

`BLOCK_INVOICE_IF_THIRD_PARTY_CUSTODY == true`
Condição:

- `asset.ownerOrgId != custodianOrgId`
- `asset.state in [IN_CUSTODY_LOCKED, SALE_INTENT_CREATED, ESCROW_READY]`
  Ação: `BLOCK` com motivo `THIRD_PARTY_ASSET_LOCKED`

Regra 2 — Liberar NF só com escrow e split prontos

`ALLOW_INVOICE_ONLY_IF_ESCROW_AND_SPLIT_READY`
Condição:

- `escrow.balance >= requiredAmount`
- `splitPlan.status == READY`
  Ação: `ALLOW` → senão `BLOCK`

Regra 3 — Liberar NF só com assinatura do dono

`ALLOW_INVOICE_ONLY_IF_OWNER_RELEASE_SIGNED`
Condição:

- evento `OWNER_RELEASE_SIGNED` válido para aquele `saleIntentId`
  Ação: `ALLOW` → senão `BLOCK`

Regra 4 — Bloquear movimentação física sem NF autorizada

`BLOCK_CHECKOUT_IF_NF_NOT_AUTHORIZED`
Condição:

- tentativa de `ASSET_CHECKOUT`
- e estado != `NF_AUTHORIZED`
  Ação: `BLOCK`

Regra 5 — Congelar operação se tentativa suspeita

`FREEZE_AND_ALERT_ON_INVALID_ATTEMPTS`
Condição:

- tentativas repetidas de emitir NF/checkout bloqueadas
  Ação:
- `FREEZE_CUSTODIAN` (watchlist/bloqueio temporário)
- notificar auditor

---

4) Split + “liberar nota” do jeito que você descreveu

Você pediu: “ao receber e gerar a nota, o escrow split deve funcionar; podendo mover o ativo após o dono liquidar split; liberar nota…”

O desenho mais seguro é:

- **NF só é autorizada** após escrow+split+assinatura do dono.
- **Movimentação física só ocorre** após `NF_AUTHORIZED`.
- **Split é liquidado** no ato da confirmação do pagamento (`PAYMENT_CONFIRMED`) e registrado no `SPLIT_SETTLED`.
- Se você quiser ainda mais duro: permitir `NF_AUTHORIZED` somente quando o pagamento já estiver “capturado” (escrow abastecido). Assim não existe “nota sem dinheiro”.

---

5) Como isso evita o prejuízo do dono

- A propriedade não consegue “vender por fora” **dentro do Ciclo+** porque:
  - sem assinatura do dono → bloqueio
  - sem escrow/split → bloqueio
  - sem NF autorizada → não sai do pátio (check-out bloqueado)
- E se tentar “burlar” com registro manual:
  - fica evidência de tentativas inválidas na auditoria
  - auditor pode bloquear o custodiante (e até o imóvel CAR/IE no seu sistema)

Observação importante (jurídico/prático)

Isso garante o bloqueio **operacional dentro do sistema**.
Para reforçar fora do sistema (cartórios/SEFAZ/terceiros), o Ciclo+ pode gerar e anexar no contrato:

- cláusula de custódia e trava de alienação
- condição de emissão de NF vinculada a escrow/split
- prova (auditoria imutável) de entrada, custódia, autorização e liquidação

---

Se você quiser, eu transformo isso em um **template pronto** no padrão dos pacotes:

- `TPL_CUSTODIA_TRAVA_FISCAL_V1.json`
- políticas `POLICY_BLOCK_NF_THIRD_PARTY_V1` e `POLICY_BLOCK_CHECKOUT_NO_NF_V1`

- e uma tela “Modo Fácil” para o operador:
  “Ativo de terceiro: NOTA BLOQUEADA — faltam: [escrow] [assinatura do dono]”.

Entendido — você quer cobrir **4 cenários de rastreio e divisão** (animais individualizáveis, animais com genealogia, ativos fungíveis/sem identidade, mercadoria unitária no PDV) e fazer o **split automático correto** sem permitir fraude. Abaixo vai o modelo técnico-operacional para embutir no Ciclo+.

---

1) Bovinos misturados: rastrear por brinco (ID) e dono

Regra base

Cada bovino tem um identificador único:

- `animalId` (interno)
- `earTagId` (brinco/RFID)
- `ownerOrgId` (dono)
- `custodianOrgId` (boitel/fazenda)
- `lotId` (opcional)

Operações obrigatórias

- **Entrada/Check-in**: lê brinco + foto/vídeo + peso + geofence
- **Movimentações**: qualquer saída/venda exige lista dos earTagIds
- **Pesagens e eventos sanitários**: sempre vinculados ao earTagId

✅ Se estiver “misturado fisicamente”, não tem problema: o sistema separa **digitalmente** pelo brinco.

---

2) Bezerro/filho: “RG da mãe” (genealogia obrigatória)

Você quer: filho seguir padrão RG da mãe. Dá para fazer assim:

Estrutura

- `damEarTagId` (brinco da mãe)
- `birthEventId`
- `calfEarTagId` (novo brinco gerado/atribuído)
- `calfIdentityRule = DERIVED_FROM_DAM`

Regra técnica

Quando registrar nascimento:

1. o app exige:
    - leitura do brinco da mãe (`damEarTagId`)
    - evidência B (foto/vídeo + geofence)
2. o sistema gera a identidade do bezerro:
    - `calfEarTagId = damEarTagId + "-" + sequencial + ano`
    - ou mantém um campo "RG" derivado (ex.: `RGLineageId`)
3. o **ownerOrgId do bezerro**:
    - padrão: herda do dono da mãe
    - exceção: se contrato definir copropriedade, aplica regra de quotas (abaixo)

✅ Isso cria rastreabilidade forte e reduz fraude (troca de cria).

---

3) Ativos impossíveis de separar por identidade (fungíveis): dividir por quotas

Exemplos: lote de ração misturado, silagem, grãos no mesmo silo sem segregação, biomassa, tanques mistos, etc.

Modelo: "Pool" (piscina) + "Unidades de Participação"

Crie um objeto:

- `poolId`
- `assetType` (ex.: MILHO_SILO, SILAGEM, BIOMASSA)
- `totalQuantity`
- `unitOfMeasure`
- `participants[]` com:
    - `ownerOrgId`
    - `quotaPct` **ou** `quotaUnits`

Regras duras

- Toda entrada no pool gera `poolContributionEvent` com evidência
- Toda saída/venda gera `poolRedemptionEvent`
- O split na venda é:
    - **proporcional às quotas vigentes na data/hora do evento**
    - registrado em auditoria (hash + snapshot das quotas)

✅ Se não dá para separar fisicamente, separa financeiramente e juridicamente por **participação**.

Anti-fraude

- quotas só mudam via evento assinado (donos envolvidos)
- auditoria guarda "cap table" (tabela de quotas) versionada

---

4) Mercadoria unitária (PDV): identificar item e separar fornecedor no split

Ex.: marketplace de insumos / loja agro / revenda.

Modelo: item serializável ou lote/sku

Para cada unidade vendida no PDV:

- `sku`
- `batchId` (lote) e validade (se aplicável)
- `supplierOrgId`
- `costOwnerOrgId` (quem tem o direito do recebível)
- `barcode/qr` (quando possível)

No PDV

Quando o caixa registra a venda:

1. o sistema monta a "cesta" com itens e fornecedores
2. calcula automaticamente:
    - valor do item
    - comissão do operador/loja
    - imposto/taxa
    - **repasse do fornecedor**
3. executa split:
    - fornecedor recebe sua parte
    - loja recebe comissão
    - taxas/impostos segregados

✅ Isso resolve "recebeu no caixa e repassa automaticamente".

---

5) Se não for identificável unitário: split proporcional à NF

Quando não dá para identificar item-a-item (ex.: NF com itens agregados, venda em volume sem rastreio por unidade, "misc"):

Modelo: Split por Nota (NF Allocation)

- ao gerar/importar a NF (ou cupom):
    - `nfId`
    - `totalValue`
    - `allocationRule`:
        - por fornecedor (percentual)

    - ou por pool/quota
    - ou por contrato (ex.: 70/30)

Regras

- NF só é autorizada se existir `allocationRule` válido
- o sistema executa o split no recebimento com base nessa regra
- mantém snapshot da regra no evento (auditável)

✅ Se o documento é "a unidade de verdade", a divisão segue a **NF**.

---

6) Regras de bloqueio e segurança (o que impede prejuízo)

Estas regras amarram tudo:

A) Bloquear venda/saída sem rastreio adequado

- Se `assetType = BOVINO` e não tiver lista de `earTagIds` → **BLOCK**
- Se `assetType = POOL` e não tiver `poolId + allocation snapshot` → **BLOCK**
- Se `assetType = PDV_ITEM` e não tiver `supplierOrgId` → **BLOCK**

B) Congelar e auditar divergência

- Se tentar vender quantidade > saldo do dono/pool → **BLOCK + ALERTA**
- Se houver tentativa repetida → **FREEZE custodian/store**

C) NF e checkout condicionados (como você já pediu antes)

- sem escrow/split/assinatura → **não emite NF**
- sem NF autorizada → **não move ativo**

---

7) Como eu recomendaria implementar como "módulos" dentro do sistema

1. **Identity Registry**

- animais (earTag), genealogia (mãe/filho), lotes

2. **Custody Ledger**

- quem tem o quê, onde está, travas

3. **Pool Engine (fungíveis)**

- poolId + cap table + entradas/saídas

4. **Split Engine**

- itemizado (PDV), por earTag, por pool, por NF

5. **Invoice/Dispatch Gate**

- trava de NF + trava de checkout

6. **Audit & Dispute**

- divergências, congelamento, parecer read-only

Se você quiser, eu já transformo isso em pacotes prontos (ZIP) com:

- schemas JSON (AnimalIdentity, BirthEvent, Pool, CapTableVersion, NFAllocation, SplitPlan)
- políticas do motor (bloqueios)
- templates (Bovino Custódia + Pool Fungível + PDV Split)
- telas mobile "modo fácil" para:
  - registrar nascimento
  - registrar entrada no pool
  - PDV com split por fornecedor
  - emissão de NF bloqueada com checklist do que falta

E eu já deixo "configurável" no Portal Admin para escolher:

- "por brinco" vs "por pool" vs "por NF" conforme tipo de ativo.

Perfeito — isso é exatamente o que "fecha" o PDV e a cadeia inteira: **a origem do direito de recebimento (beneficiário do split)** tem que nascer lá atrás, na **NF de entrada** e "viajar junto" com o item/lote em todas as movimentações até a venda final.

Abaixo está o desenho **implementável** (dados + fluxo + regras) para o Ciclo+.

---

1) Regra de ouro: "Sem NF de entrada, sem split válido"

Todo produto/mercadoria que circular no Ciclo+ (estoque → transferência → venda → devolução) precisa carregar um **Provenance Key** derivado da **NF-e de entrada**.

Provenance Key (chave de rastreio)

- `inboundNfKey` (chave da NF-e de entrada)
- `inboundItemSeq` (número do item na NF)
- `supplierOrgId` (emitente/fornecedor)

- `lotId` (lote interno do Ciclo+ gerado a partir da NF)
- opcional: `batchId`, `validade`, `ncm`, `cfop`, etc.

✅ Isso garante que, ao vender no balcão, o sistema sabe **quem é o fornecedor de cada item** e como split deve ser feito.

---

2) Cadastro de NF de entrada → gera "Lotes rastreáveis" no estoque

Workflow (obrigatório)

1. **Importar NF-e (XML/SEFAZ/arquivo)**
2. Para cada item da NF:
    - criar `StockLot` (lote de estoque)
    - gravar:
        - `supplierOrgId`
        - `inboundNfKey`
        - `inboundItemSeq`
        - custo, tributos, quantidades
3. **Baixa/Movimentação** só pode ocorrer por lote (FIFO/FEFO configurável)

Resultado

Todo estoque passa a ser composto por "pedaços rastreáveis" (`StockLot`), cada um ligado a uma NF de entrada.

---

3) PDV Balcão com cesta de vários fornecedores: split automático por item

No PDV (venda balcão)

Ao escanear um produto, o sistema **não adiciona "o SKU genérico"**; ele adiciona uma **reserva de lote**:

- `saleLine.sku`
- `saleLine.qty`
- `saleLine.stockLotId` (escolhido automaticamente por FIFO/FEFO)
- herdado do lote:
    - `supplierOrgId`
    - `inboundNfKey/inboundItemSeq`

✅ Assim, uma mesma cesta pode ter 10 itens e 6 fornecedores — o split sai certo.

No pagamento

O Ciclo+ cria um `SplitPlan` agrupando por fornecedor:

- Para cada fornecedor: soma `subtotal itens daquele fornecedor`

- Aplica regras:
  - comissão da loja
  - taxas/MEIOS de pagamento
  - frete (se tiver)
  - impostos (se você quiser segregar em linha separada)

E executa:

- **Split fornecedor A**
- **Split fornecedor B**
- **Split loja**
- etc.

---

4) “Vale para toda a cadeia de suprimentos” (não só PDV)

Isso vira uma regra geral: **qualquer movimentação que gere valor** (venda, transferência com cobrança, consignação liquidada, marketplace, leilão, etc.) precisa carregar referência de origem:

Eventos e campos mínimos

- `movementId`
- `movementType` (TRANSFER, SALE, RETURN, CONSIGNMENT, etc.)
- `lines[]` com:
  - `stockLotId` **ou** `provenanceKey`
  - `qty`
  - `supplierOrgId` (herdado)
  - `inboundNfKey/inboundItemSeq` (herdado)
- `splitPlanId` (quando houver liquidação)

✅ Isso garante que o mesmo mecanismo de split funcione na loja, no armazém, no marketplace e em qualquer “cadeia”.

---

5) Segregação “observando a NF de entrada” (o que você pediu)

No cadastro/importação da NF de entrada, você precisa capturar também:

- se o item é **consignado** / “de terceiro”
- se tem **regra de repasse** (percentual fixo, tabela, contrato)
- se o fornecedor é **beneficiário direto** (recebe automático) ou via intermediário

Campo recomendado no lote

- `settlementModel`:
  - `DIRECT_TO_SUPPLIER` (split direto)
  - `CONSIGNMENT_ESCROW` (escrow + liberação por evidência)

    - `HOLD_UNTIL_QUALITY` (segura parte até laudo)
- `defaultSplitRuleId` (regra padrão do fornecedor)

Assim, na venda, não precisa "adivinhar" nada: o lote já diz como liquidar.

---

6) Regras de bloqueio (anti-fraude)

Estas regras garantem que ninguém burle:

Regra A — produto sem origem bloqueia

Se `saleLine.stockLotId` vazio ou lote sem `inboundNfKey`:

- **BLOCK**: `MISSING_INBOUND_NF_PROVENANCE`

Regra B — proibir "mudar fornecedor" no PDV

`supplierOrgId` vem do lote e é **read-only** na venda.

Regra C — venda multi-fornecedor só finaliza com split plan pronto

Se existir mais de 1 fornecedor na cesta e não houver `SplitPlan READY`:

- **BLOCK**: `SPLITPLAN_REQUIRED`

Regra D — devolução estorna o mesmo fornecedor (trilha reversa)

Devolução (`RETURN`) deve referenciar:

- `saleId` + `saleLineId` + `stockLotId`
  e gerar split reverso.

---

7) Casos difíceis (já deixo coberto)

Mesmo SKU com lotes de fornecedores diferentes

- FIFO/FEFO escolhe lote automaticamente
- operador pode trocar lote **apenas com permissão** e evento auditável

Produto "sem lote" (commodity)

- vira "Pool" (quotas) ou "NF Allocation"
- nunca entra como item unitário

Compra/transferência entre unidades internas

- lote mantém `inboundNfKey` original
- movimentação cria “sub-lotes” mas preserva origem

Implementação mínima (o que o DEV precisa criar)

Tabelas/Entidades

- `InboundNF` + `InboundNFItem`
- `StockLot` (gerado por item da NF)
- `InventoryMovement` + `MovementLine(stockLotId, qty)`
- `PDVSale` + `PDVSaleLine(stockLotId, qty, price)`
- `SplitPlan` + `SplitLine(beneficiaryOrgId, amount)`
- `Return` referenciando `PDVSaleLine`

Rotas do app/portal

- Importar NF → gera lotes
- Estoque → mostra lotes e origem
- PDV → adiciona item escolhendo lote
- Pagamento → split automático por fornecedor
- Devolução → estorna por lote/origem

Se você quiser, eu gero um **ZIP v2** do pacote anterior adicionando exatamente isso:

- schema `InboundNF`, `InboundNFItem`, `StockLot`, `MovementLine` e `PDVSaleLine` com `provenanceKey`
- policies novas:
  - `BLOCK_SALE_WITHOUT_INBOUND_NF`
  - `BLOCK_SPLIT_IF_SUPPLIER_MISSING`
  - `RETURN_MUST_REFERENCE_ORIGINAL_SALELINE`
- template novo:
  - `TPL_SUPPLYCHAIN_NF_PROVENANCE_V1`
- e 2 telas “Modo Fácil”:
  - **Importar NF → gerar lotes**
  - **PDV multi-fornecedor → split automático**

Se eu seguir assim, você quer que a loja opere com **FIFO** (padrão) ou **FEFO por validade** (para insumos com vencimento)?

Perfeito — então a regra não é “só NF”. É **Identidade obrigatória universal** para qualquer produto/mercadoria, com **rastreio por qualquer rastro existente** (NF, código, QR, RFID, lote, série, contrato, pool). Isso vira um **pilar do núcleo**: *sem identidade não circula*.

Abaixo vai o modelo completo (bem prático) para embutir no Ciclo+.

---

1) Regra de Ouro: “Sem Identidade, sem operação”

Todo item cadastrado no Ciclo+ (insumo, peça, medicamento, semente, produto final, mercadoria) deve ter:

Identidade mínima obrigatória (Product Identity)

- `productId` (UUID interno)
- `sku` (código interno do produto)
- **um identificador rastreável obrigatório** (pelo menos 1):
  - `gtin/ean` (código de barras) **ou**
  - `qrCode` **ou**
  - `rfidTag` **ou**
  - `internalCode` (código do Ciclo+) **ou**
  - `serialNumber` (se serializável)
- `identityMode`:
  - `UNIT` (unitário/serializável)
  - `LOT` (lote/batch)
  - `POOL` (fungível sem segregação)
- `ownerOrgId` (titular do direito do estoque/recebível)
- `supplierOrgId` (origem comercial quando existir)

✅ Isso garante que qualquer produto tem “RG digital” dentro do sistema.

---

2) “Rastro” não é só NF: é um conjunto de evidências/âncoras

Você quer “qualquer rastro existente”. Então definimos um objeto padrão:

Provenance (âncoras de origem)

Cada lote/unidade carrega uma lista de `anchors[]`:

- `NF_INBOUND`: chave NF + item seq
- `BARCODE`: EAN/GTIN
- `SERIAL`: número de série
- `BATCH`: lote/validade
- `CONTRACT`: id do contrato/fornecedor
- `PO`: pedido de compra
- `MANUAL_CODE`: código digitado/cadastrado
- `PHOTO_OCR`: foto do papel + OCR (hash da evidência)
- `RFID_SCAN`: leitura RFID (hash/time)
- `WEIGH_TICKET`: ticket de balança

✅ Se não tiver NF, tudo bem — entra via **código + evidência** (foto/OCR, QR, RFID, etc.).
O importante é: sempre existe pelo menos 1 âncora validável.

---

3) Como isso vira estoque real: "Unidade / Lote / Pool"

A) UNIT (serializável)

Ex.: eletrônico, equipamento, implemento, itens caros.

- Cada unidade tem `serialNumber` único.
- Split pode ser por unidade.

B) LOT (lote/batch)

Ex.: medicamentos, sementes, fertilizantes, ração.

- `batchId` + validade + quantidade.
- PDV sempre "consome" de um lote específico (FIFO/FEFO).
- Split/repasse é atribuído ao fornecedor daquele lote.

C) POOL (fungível)

Ex.: grãos no silo sem segregação, silagem, biomassa.

- Não tenta "forçar unidade".
- Vira `poolId` com quotas/alocações (como já definimos).

✅ Isso evita fraude e evita modelar errado.

---

4) Cadastro obrigatório do produto (para o DEV)

Quando cadastrar um produto (SKU)

O sistema exige:

- SKU + nome + categoria
- `identityMode` (UNIT/LOT/POOL)
- **pelo menos 1 identificador**: EAN/QR/RFID/código interno
- Política padrão de rastreio:
  - LOT: exige lote/validade quando movimentar
  - UNIT: exige serial
  - POOL: exige poolId

Se faltar → bloqueia cadastro/uso

Isso vira regra de motor.

---

5) Entrada no estoque sem NF: permitido, mas com identidade + evidência

Você pediu isso explicitamente: “qualquer rastro existente seja por cadastro através de código”.

Então a entrada no estoque tem dois caminhos:

Caminho 1 — Entrada com NF (ideal)

- cria lotes com âncora NF

Caminho 2 — Entrada sem NF (permitido)

- operador informa:
  - `supplierOrgId` (ou “origem desconhecida” com restrição)
  - `batchId`/serial se aplicável
  - quantidade
  - preço/custo (opcional)
- e anexa evidência:
  - foto do documento, recibo, papel, nota manual (OCR) + hash
- gera lote com âncoras:
  - `MANUAL_CODE` + `PHOTO_OCR` (+ BARCODE se tiver)

✅ Você mantém rastreabilidade mesmo sem NF.

---

6) Venda PDV multi-fornecedor: sempre “por lote/unidade”

A cesta no PDV não é só SKU. É:

- `saleLine` aponta para `stockUnitId` (UNIT) ou `stockLotId` (LOT)
- o fornecedor/beneficiário vem do lote/unidade
- o split agrupa por fornecedor

Se tentar vender sem vínculo a lote/unidade/pool:

- **BLOCK**

---

7) Split “vale para toda a cadeia”

Qualquer evento de saída que gere valor precisa carregar referência da identidade:

- UNIT: lista de `serials`
- LOT: `stockLotId` + `batchId`
- POOL: `poolId` + snapshot quotas

E o motor gera `SplitPlan` automaticamente.

---

8) Regras anti-fraude (as mais importantes)

Regra 1 — Produto sem identidade não circula

`BLOCK_MOVEMENT_IF_IDENTITY_MISSING`

Regra 2 — LOT exige lote/validade

`BLOCK_IF_LOT_MODE_AND_BATCH_MISSING`

Regra 3 — UNIT exige serial

`BLOCK_IF_UNIT_MODE_AND_SERIAL_MISSING`

Regra 4 — POOL exige poolId e quotas snapshot

`BLOCK_IF_POOL_MODE_AND_POOL_OR_SNAPSHOT_MISSING`

Regra 5 — Beneficiário do split é read-only

`BLOCK_IF_SUPPLIER_CHANGED_MANUALLY`

---

9) O que isso resolve

- PDV multi-fornecedor: split perfeito na hora do pagamento ✅
- Entrada sem NF: ainda rastreável via evidência/código ✅
- Toda movimentação do Ciclo+ mantém cadeia de suprimentos auditável ✅
- Reduz fraude (troca de fornecedor, venda de “produto sem origem”) ✅

Próximo passo que eu faço para você

Eu consigo gerar um **pacote v2** (ZIP) atualizando o pacote anterior com:

- schema de `Product`, `ProductIdentity`, `StockLot/StockUnit`, `ProvenanceAnchor`
- policies: bloqueio de movimentação sem identidade
- template: `TPL_PRODUCT_IDENTITY_UNIVERSAL_V1`
- telas "Modo Fácil":
    - Cadastro de produto (identidade obrigatória)
    - Entrada sem NF (por código + foto/OCR)
    - PDV multi-fornecedor (por lote)

Só me diga uma coisa (sem complicar):
Você quer que, quando não houver fornecedor definido, o sistema trate como **"origem desconhecida" com venda bloqueada** ou **permita vender mas com split indo para a loja** (e auditoria marcando risco)?

Perfeito — isso vira um **princípio estrutural do Ciclo+**:

Nada existe "solto": todo **contrato / projeto / operação** nasce com uma **Identidade Única** e uma **Expectativa de Retorno Mensurável**.
Só **avança/movimenta/liquida** se cumprir **variáveis mínimas** (tolerâncias) e **provas (evidência)**.
Toda movimentação carrega essa identidade (rastro de ponta a ponta).

Abaixo está o modelo **técnico-operacional** pra embutir no sistema.

---

1) Identidade única universal

Crie um identificador padrão para tudo:

`workId` (identidade do trabalho)

- único, imutável, hashável
- exemplo de composição:
    - `WORK-{tipo}-{orgId}-{ano}-{sequencial}-{checksum}`
- tipos:
    - `CONTRACT` (contrato)
    - `PROJECT` (projeto)
    - `CYCLE` (ciclo produtivo)
    - `LOT` (lote)
    - `ASSET_CUSTODY` (custódia)
    - `SALE` (venda)
    - `NF` (documento)
    - `PAYMENT` (pagamento)

✅ **Regra:** todo evento e registro tem `workId` obrigatório.

---

2) Expectativa de retorno mensurável

Todo `workId` tem um bloco obrigatório:

`ReturnExpectation`

- `returnType`: `FINANCIAL | PRODUCTIVITY | DELIVERY | QUALITY | MIXED`
- `baseline`: valor/condição inicial (ex.: peso inicial, saldo, área plantada)
- `target`: meta (ex.: kg ganho, litros, sacas, % qualidade, R$)
- `timeWindow`: início/fim ou marcos (ex.: D+30, safra, 6 meses)
- `minAcceptable`: **mínimo aceitável** (limiar de êxito)
- `tolerances`: tolerâncias por variável (faixas)
- `verificationMethod`: como medir (pesagem, laudo, checklist, geofence, auditor)

Exemplo (boitel):

- baseline: 320 kg
- target: +120 kg em 120 dias
- mínimo aceitável: +80 kg
- tolerâncias: pesagem a cada 30 dias ± 3 dias

Exemplo (leite):

- target: volume mensal + qualidade (CCS/CBT) dentro das faixas
- mínimo aceitável: piso garantido + SLA de laudo

---

3) Máquina de estados "só avança com êxito"

Todo contrato/projeto segue o mesmo padrão de estados:

- `DRAFT`
- `SIGNED`
- `ACTIVE`
- `MILESTONE_PENDING`
- `MILESTONE_VERIFIED`
- `SETTLEMENT_READY`
- `SETTLED`
- `FAILED` / `DEFAULT`
- `DISPUTED`

✅ **Regra:** "movimenta-se ou liquida" somente quando:

- marcos foram verificados
- variáveis mínimas atingidas
- evidência válida anexada
- split/escrow pronto

---

4) Variáveis mínimas e “portões” de execução

Crie o conceito de **Gate** (portão) obrigatório:

`GateRule`

- `gateId`
- `requiredMetrics[]` (ex.: peso, volume, qualidade, entrega, documento)
- `minThresholds[]` (mínimos)
- `evidenceRequired`: `A | B | B_LAB | SIGNATURE | DOC`
- `actionOnFail`: `BLOCK | FREEZE | DISPUTE | ESCALATE_AUDIT`

Exemplos de Gates:

- **Gate de movimentação de ativo**: só sai se `OWNER_RELEASE_SIGNED` + `ESCROW_READY`
- **Gate de emissão de NF**: só libera se split estiver `READY`
- **Gate de liquidação**: só libera tranche se marco estiver `VERIFIED`

---

5) Todo movimento carrega a identidade

Qualquer evento do sistema deve conter:

`EventEnvelope`

- `eventId`
- `workId` (obrigatório)
- `orgId/tenantId`
- `actorUserId`
- `eventType`
- `timestamp`
- `evidenceRef` (se exigido)
- `previousHash` + `hash` (audit chain)

E, para movimentações:

- `assetRefs[]` (animalId/earTag, stockLotId, poolId etc.)
- `documentRefs[]` (NF, contrato, laudo)
- `moneyRefs[]` (escrowId, splitPlanId, paymentId)

✅ Isso garante rastreio “prova-a-prova”.

---

6) Regras do motor (bem objetivas)

Estas são as 5 regras “padrão ouro”:

1. **BLOCK se não tiver `workId`**
2. **BLOCK se `ReturnExpectation` estiver ausente** (contratos/projetos)
3. **BLOCK se Gate do estado atual não foi satisfeito**
4. **BLOCK se evidência exigida não existir/invalidar**
5. **BLOCK se split/escrow não estiver pronto quando necessário**

---

7) O que isso permite no agro 360

- Contrato de produção sob demanda: só libera pagamentos por etapas verificadas
- Boitel: só movimenta/vende com autorização do dono + split/escrow
- PDV: toda venda tem workId e liquida por split por fornecedor
- Pool fungível: saída só com snapshot de quotas e split associado
- Ambiental/CAR: Gate de compliance pode bloquear projeto/atividade automaticamente

---

Resultado prático

Você cria um sistema onde:

- toda operação é rastreável por uma identidade única
- existe uma "tese" de retorno mensurável desde o início
- só avança quando o mínimo aceitável for comprovado
- reduz fraude e "operações sem lastro" a zero

Se você quiser que eu já deixe **pronto para dev**, eu organizo isso em um pacote (ZIP) com:

- schema `Work`, `ReturnExpectation`, `GateRule`, `EventEnvelope`
- templates: `TPL_CONTRACT_WITH_RETURN_V1`, `TPL_PROJECT_WITH_RETURN_V1`
- policies padrão (5 regras ouro)
- mermaid da state machine universal

E eu assumo como padrão: **todo contrato/projeto nasce com workId + retorno mensurável obrigatório**.

Entendi — então **não há conflito** com o RTF. A NF continua sendo o **documento fiscal obrigatório para entrada**. A **foto** entra como:

1. **facilitador de lançamento** (OCR/voz) de operações já autorizadas e já existentes no sistema, e
2. **evidência operacional** do evento (lastro), não "substituto" fiscal.

Isso encaixa perfeitamente no núcleo do Ciclo+ (evento + evidência + auditoria + regras).

A forma correta de modelar é assim:

---

1) Separar “Documento Fiscal” de “Evidência Operacional”

Documento fiscal (NF/XML/PDF)

- usado para: **entrada de estoque**, custo, impostos, conformidade
- regra: **obrigatório para criar lote/estoque fiscal**

Evidência operacional (Foto/Áudio/OCR)

- usado para: **provar que a operação ocorreu** e **reduzir digitação**
- regra: obrigatório por risco (Tipo A/B) e anexado ao evento (auditável)

✅ A foto nunca “dispensa NF”. Ela **dispensa digitação** e **fortalece lastro**.

---

2) Caso de uso: retirada de sal mineral anotada em planilha/papel

Cenário

- Estoque de sal mineral já entrou via NF e existe como lote(s) no sistema.
- Operador faz retiradas diárias e anota numa planilha/caderno.
- Depois tira foto e o sistema lança automaticamente.

Como vira evento no Ciclo+

Cada retirada vira um evento `STOCK_ISSUE` (baixa/consumo), com:

- `stockLotId` (ou regra FIFO/FEFO escolhendo o lote)
- `productId/sku`
- `qty`
- `timestamp`
- `locationRef` (depósito/pasto)
- `workId` (se estiver ligado a um projeto/lote de produção)
- `evidenceRef` = foto do papel + OCR + hash (Tipo B) **ou** Tipo A se for rotineiro e baixo risco

✅ Exatamente o que você descreveu: operação já cadastrada, só que lançada sem digitar.

---

3) Fluxo operacional recomendado (simples e robusto)

A) No campo (offline se precisar)

1. Operador anota: data / produto / quantidade / local / responsável
2. Tira foto no app (ou manda áudio)
3. App cria “pré-evento” em fila com evidência anexada

B) No sistema (processamento)

4. OCR/voz extrai os campos (qty, produto, local etc.)
5. Motor de regras valida:
    - existe estoque?
    - quantidade não ultrapassa limite?
    - usuário tem permissão?
    - produto exige lote? exige saldo mínimo?
6. Se bater: grava evento definitivo + auditoria (hash, encadeamento)
7. Se não bater: gera “pendência” para conferência (sem bloquear a operação de campo)

---

4) Como garantir que OCR/voz não gere fraude/erro

Você resolve com 3 camadas:

Camada 1 — template de planilha/caderno

Padronize um modelo com campos fixos:

- Data
- Produto (SKU/código curto)
- Qtd
- Local
- Responsável (assinatura)

Isso aumenta acerto do OCR.

Camada 2 — validação automática

- se `qty > saldo` → bloqueia ou cria pendência
- se `sku` não reconhecido → pendência
- se `local` inválido → pendência
- se a mesma foto já foi processada (hash igual) → bloqueia duplicidade

Camada 3 — evidência e auditoria

- guarda a imagem original
- guarda o texto extraído (OCR)
- guarda o hash
- registra quem confirmou

---

5) Regras (bem diretas) para o DEV implementar

1. **Entrada estoque (NF)**: só cria `StockLot` se houver NF/Doc fiscal válido.
2. **Operação por foto/voz**: sempre cria um `EventDraft` com `evidenceRef`.
3. **Evento só "vale"** quando passar no motor de regras.
4. **OCR/voz nunca altera fiscal**: apenas movimenta estoque já existente.
5. **Anti-duplicação**: `evidenceHash` + `timestampWindow` para impedir replay.

---

6) Onde isso encaixa no núcleo patenteável

Esse recurso vira apenas:

- **método de captura de evidência e dados** (foto/voz)
- mantendo o pipeline: autenticação → RBAC → orgId → regras → estados → evidência → auditoria

Ou seja: **melhora usabilidade sem quebrar compliance fiscal**.

Se você quiser, eu descrevo (em texto, aqui mesmo) um **modelo padrão de planilha manual** para o operador (o layout ideal para OCR) + o **JSON do evento STOCK_ISSUE** + as **regras de validação** (para você entregar ao dev como "especificação fechada").

A seguir vai uma análise "sistêmica" do Ciclo+ (Agro 360) com foco em **falhas potenciais**, **lacunas de módulos**, e **pontos que precisam de amarração técnica** para não virar um ERP/marketplace solto — mantendo o núcleo patenteável (eventos + regras + estados + evidência + auditoria + escrow/split).

---

1) Falhas estruturais mais prováveis

1.1 Identidade universal ainda não está fechada para *tudo*

Você já definiu identidade para:

- orgId/tenantId (multi-organização),
- produtos (SKU/lote/unidade/pool),
- ativos animais (brinco + RG derivado),
- contratos/projetos (WorkID).

Falha comum: alguns fluxos "escapam" sem ID (ex.: devolução, perdas, ajustes, retrabalhos, reclassificação de produto, transferência entre depósitos, desmembramento de lote, mistura de lotes).

Correção: criar o módulo "Identity & Provenance Service" obrigatório:

- gera IDs,
- mantém árvore de proveniência (pai→filho) para qualquer transformação,
- impede eventos sem `workId` / `assetId` / `stockLotId` conforme o caso.

1.2 Transformações e conversões (ex.: mistura, processamento, fracionamento) estão subespecificadas

No agro real:

- mistura sal mineral,
- fraciona insumos,
- reembala,
- transforma (ração, silagem, grãos),
- abate/processa (carne),
- leite vira lote de produto (derivação).

Falha: sem "módulo de transformação" você perde rastreio e split correto.

Módulo faltante: Batch Transformation / Bill of Materials (BoM)

- Evento `TRANSFORM` que consome lotes A/B/C e produz lote D, com:
  - rendimento,
  - perdas,
  - novas regras de titularidade,
  - novo "rastro".

1.3 Disputa e arbitragem ainda está "leve"

Você já tem `DISPUTED` na máquina de estados, mas falta o que resolve na prática:

- divergência de peso,
- qualidade/lab contestado,
- entrega parcial,
- "foto" inconclusiva,
- prazo estourado,
- má fé.

Módulo faltante: Dispute Resolution (Ombudsman)

- abertura de disputa,
- congelamento parcial (escrow freeze),
- coleta de contra-evidência,
- laudo independente,
- decisão (auditor/árbitro),
- execução da decisão (libera/estorna/multa/penhora lógica).

Sem isso, o escrow vira gargalo e a confiança cai.

1.4 Garantias e “penhora lógica” precisam de formalização jurídica-operacional

Você propôs travar ativos do comprador e impedir venda/transferência/penhora posterior.

Falha: sem “registro formal de garantias” você pode ter:

- conflito com terceiros,
- questionamento sobre “trava” sem base contratual clara,
- dificuldades de execução.

Módulo faltante: Collateral Registry (Garantias)

- cadastro de garantias por WorkID (imóvel/CAR/IE, recebíveis, estoque, ativos),
- regras de ativação (default),
- regras de liberação (liquidação),
- logs e evidências de aceite,
- integração com consulta/validação documental (quando houver).

1.5 Compliance fiscal/documental (NF/CT-e/MDF-e) não está completo como módulo

Você falou de “travar emissão de notas” e “split ao gerar nota”.

Falha: sem módulo fiscal:

- não valida CFOP/NCM,
- não garante “documento certo no momento certo”,
- pode travar operação por inconsistência tributária.

Módulo faltante: Fiscal & Documents Hub

- NF-e/NFS-e/CT-e/MDF-e (conforme aplicável),
- guarda XML/PDF/recibos,
- validações básicas,
- “gates” ligados a documentos (sem doc → não movimenta / não liquida),
- trilha de auditoria documental.

(Isso pode ser integrado via parceiros/ERPs, mas precisa existir como módulo lógico no Ciclo+.)

1.6 Integração com dispositivos (balança, RFID, drones, tratores) precisa de “camada de confiabilidade”

Falha: se qualquer device manda dado sem validação forte:

- fraude por injeção,
- leituras repetidas,
- dados fora de faixa.

Módulo faltante: Device Attestation & Telemetry

- cadastro de dispositivo,
- assinatura do dispositivo,
- validação (range checks),
- anti-replay,
- sincronização offline,
- status de confiabilidade por device.

1.7 Modelo de risco ainda precisa virar “tabela objetiva” para o motor de evidências

Você já tem Tipo A e Tipo B, mas falta uma matriz explícita:

- quais eventos são “críticos” em cada módulo?
- quando exigir laudo?
- quando exigir dupla assinatura?

Módulo faltante: Risk Matrix + Evidence Policy Catalog

- `eventType → riskLevel → evidenceType → requiredFields`.

Sem isso, o sistema vira subjetivo e difícil de operar.

---

2) Módulos faltantes por área (checklist)

2.1 Cadeia de suprimentos/estoque

- ✅ Estoque por lote / baixa com evidência
- ❌ **Transformação (BoM / mistura / fracionamento / reembalagem)**
- ❌ **Rastreio de recall** (se um lote deu problema, achar onde foi parar)
- ❌ **Inventário cíclico** (contagem, divergência, aprovação e evidência)

2.2 Animais e custódia

- ✅ Brinco/RFID + dono + custódia

- ✅ RG do filho (derivação)
- ❌ **Eventos zootécnicos completos** (vacina, manejo, mortalidade, tratamentos) com impacto em compliance e contrato
- ❌ **Sanidade e restrições** (bloqueio automático por evento sanitário)

2.3 Leite (seu problema crítico)

Você descreveu: produtor entrega 30 dias, recebe com mais 30, indústria retém, risco de calote.

Falta um módulo específico para isso:

- ❌ **Milk Settlement & Price Stabilization**
  - contrato com piso + fórmula,
  - medições (volume + qualidade),
  - escrow/tranches semanal/quinzenal,
  - gate de laudo,
  - ajuste por variação com "banda" (tolerância),
  - penalidades automáticas e travas (garantias).

Sem esse módulo, o sistema não resolve o principal "dor" do leite.

2.4 Marketplace (insumos + produção sob demanda + leilão)

- ✅ conceito de marketplace inverso/reverso + leiloeiro credenciado
- ❌ **Catálogo e normalização de produtos** (NCM, unidade, embalagens, equivalências)
- ❌ **Reputação/score de fornecedores e compradores**
- ❌ **Motor de negociação** (propostas, contraproposta, SLA, expiração, histórico)

2.5 Contratos por ciclos de produção

- ✅ contratos com WorkID, retorno mensurável, gates
- ❌ **Biblioteca de cláusulas e templates versionados** (com assinatura e hash)
- ❌ **Gestão de anexos/escopos** (mudança de escopo com trilha)

2.6 Financeiro (escrow/split)

- ✅ escrow/tranches/split por marcos
- ❌ **Gestão de chargeback/estorno** (especialmente no PDV)
- ❌ **Conciliação bancária** (PSP/bancos vs ledger interno)
- ❌ **Ledger contábil interno** (mínimo: reason codes + partidas)

---

3) Falhas de segurança e governança que precisam estar explícitas

3.1 Permissões e segregação de função

Você precisa de “SoD” (segregation of duties) para reduzir fraude:

- quem registra evidência não pode aprovar sozinho,
- quem é custodiante não pode liquidar sozinho,
- auditor read-only não pode alterar.

Módulo faltante: Governance & SoD Rules

3.2 Imutabilidade “de verdade”

Audit append-only com hash é ótimo, mas precisa definir:

- onde armazena,
- como faz verificação,
- como prova integridade externamente (opcional).

Módulo faltante: Audit Verification Service

- verificador de hash chain,
- exportação de prova,
- checks periódicos.

---

4) O que eu considero “top 7” para fechar o sistema (prioridade)

1. **Transformação / BoM** (mistura, fracionamento, derivação)
2. **Disputas & arbitragem** (escrow freeze + decisão)
3. **Garantias/penhora lógica** (registry + regras de execução)
4. **Fiscal & Documents Hub** (NF/CT-e/MDF-e como gates)
5. **Device attestation & telemetria** (balança/RFID/drones)
6. **Risk matrix objetiva** (Tipo A/B/B_LAB por evento)
7. **Leite: módulo de liquidação + estabilidade de preço** (piso/fórmula + tranches)

---

5) Falhas “de produto” (UX/operacional) que quebram adoção

- modo offline sem fila robusta = operador desiste
- OCR sem template e sem validação = vira bagunça
- regra dura demais sem “pendência” = trava operação real
- falta de “painel do auditor” com filtros e parecer = perde confiança

✅ Solução: sempre ter:

- **pendências** (exception queue),

- **modo fácil**,
- **confirmar por voz + foto**,
- **painel de auditoria read-only** com parecer.

Se você quiser, eu devolvo isso no formato que o dev mais usa:

- um **mapa de módulos** (core + 20 módulos) com
  - eventos principais,
  - estados,
  - evidências,
  - gates,
  - integrações externas,
  - e as "regras de bloqueio" (por módulo).

Consigo sim — e do jeito que ajuda DEV mesmo: **organizado por módulos**, com **o que existe**, **o que falta**, **eventos**, **gates**, **evidências** e **integrações**. Vou te entregar aqui em um "mapa mestre" (sem depender de arquivos).

Mapa mestre do Ciclo+ Agro 360

0) Núcleo patenteável (obrigatório em tudo)

Pipeline único por evento:

1. Autenticação
2. RBAC / permissões
3. Filtro `orgId/tenantId`
4. Motor de regras (pré-execução)
5. Máquina de estados (transições permitidas)
6. Evidência (A/B/DOC/LAUDO) proporcional ao risco
7. Auditoria imutável (hash + encadeamento + timestamp)
8. Liquidação condicionada (escrow/tranches/split) quando aplicável

Regra mestre: nenhum módulo "fura" o núcleo.

A) Identidade & Proveniência (MÓDULO BASE)

A1. Identity Service (IDs universais)

Objetivo: garantir que nada exista sem identidade.

- `workId` (contratos/projetos/ciclos)
- `assetId` (animal/ativo físico)
- `productId/sku`

- `stockLotId / stockUnitId / poolId`
- `docId` (NF/CT-e/laudo)

Gates mínimos

- `workId` obrigatório em qualquer contrato/projeto/liquidação
- `stockLotId` obrigatório em qualquer baixa/PDV
- `animalId/earTag` obrigatório em custódia

Evidência

- Tipo A em rotinas
- Tipo B em eventos críticos (foto/GPS)

✅ Você já tem parte disso.
❗ Falta amarrar "transformações" (ver módulo B3).

B) Supply Chain (Estoque, Insumos, PDV, Documentos)

B1. Entrada de estoque (Fiscal)

Objetivo: criar lotes fiscais e rastreáveis.

- Entrada **somente com NF** (XML/PDF) → gera `StockLot`
- Lote herda origem e fornecedor

Gates

- NF válida / vinculada / item reconhecido
- Produto cadastrado com identidade

B2. Saída/Consumo por operação (sem digitação)

Objetivo: retirar estoque usando foto/voz como "entrada de dados + evidência".

- Evento: `STOCK_ISSUE`
- Fonte: OCR da planilha manual + voz opcional
- Nunca substitui NF, só evita digitar

Gates

- `qty <= saldo`
- permissão do operador
- se item crítico: exigir foto/GPS e/ou laudo

B3. Transformação / Mistura / Fracionamento (FALTA)

Objetivo: não perder rastreio quando o mundo real mistura/transforma.

- Evento: `TRANSFORM_BATCH`
- Consome lotes A/B/C → produz lote D
- Define rendimento/perda e herança de titularidade

Gates

- fórmulas permitidas (BoM)
- evidência mínima (foto/registro)
- se alto risco: laudo/assinatura

➡️ **Esse é um dos módulos mais importantes que falta.**

B4. PDV balcão multi-fornecedor + split

Objetivo: venda com cesta de múltiplos fornecedores e split automático.

- Cada item da venda aponta para `stockLotId` (ou `stockUnitId`)
- Split agrupa por fornecedor/beneficiário

Gates

- todo item precisa ter fornecedor (herdado do lote)
- splitPlan READY antes de finalizar

B5. Fiscal & Document Hub (FALTA lógico)

Objetivo: centralizar documentos como "gates".

- NF-e / NFS-e / CT-e / MDF-e (quando aplicável)
- Validação básica + armazenamento + hash
- Gate: "sem doc certo, não movimenta / não liquida"

C) Animais, Custódia e Ativos de Terceiros

C1. Identidade animal (brinco/RFID)

- `animalId` ligado ao `earTagId`
- dono vs custodiante

Gates

- sem brinco → bloqueia movimentação/venda

C2. Genealogia (filho segue RG da mãe)

- evento `BIRTH_REGISTERED`
- gera RG derivado
- herança de dono (regra)

C3. Custódia boitel / terceiros (robusto)

Objetivo: segurança total do dono do ativo.

- check-in com evidência B
- trava de custódia
- venda/movimento só com liberação do dono + escrow

Gates

- `OWNER_RELEASE_SIGNED`
- `ESCROW_READY`
- conferência final por evidência B

C4. Mistura de animais / quotas (quando não separa)

Objetivo: se não dá para individualizar, vira Pool.

- pool por quotas
- snapshot versionado

Gates

- snapshot de quotas assinado
- split proporcional

D) Contratos, Projetos e Ciclos (WorkID + retorno mensurável)

D1. Contrato/Projeto sempre com retorno mensurável

Objetivo: toda operação tem “meta e mínimo aceitável”.

- baseline / target / mínimo / tolerâncias / janela de tempo
- gates por etapa

D2. Módulo de templates jurídicos (FALTA)

Objetivo: contratos por ciclo (cria/recria/engorda; leite; aves; peixes; silvicultura…)

- biblioteca de cláusulas
- versionamento + hash

- assinatura digital
- trilha de alteração (aditivos)

E) Financeiro, Escrow, Split, Tranches e Garantias

E1. Escrow obrigatório em negociação direta

- negociou → pesou (evidência) → escrow → entregou (evidência) → libera

E2. Produção sob demanda com tranches

- comprador dá garantia
- pagamentos por etapas verificadas

E3. Collateral / Garantias (FALTA)

Objetivo: "penhora lógica" operacional e juridicamente organizada.

- cadastro de garantias por WorkID
- travas e desbloqueios
- execução por default

E4. Conciliação & estorno (FALTA)

- conciliar PSP/bancos vs ledger interno
- chargeback e devoluções (PDV)

F) Leite (módulo específico que falta)

Problema: preço instável + calote + pagamento 30/60 dias.

F1. Milk Settlement & Price Stability (FALTA)

Objetivo: produtor saber preço e reduzir calote.

- contrato com:
  - piso + fórmula (índices/qualidade)
  - banda de volume (mín/máx)
  - tranches semanais/quinzenais
  - escrow obrigatório do comprador
  - gates de qualidade (CCS/CBT/gordura/proteína)

Como lidar com volume variável

- define "volume comprometido" + "banda tolerável"

- se entregar menos: ajuste por tranche (penalidade/reatribuição)
- se entregar mais: preço por faixa (opcional) ou novo aditivo automático

---

G) Disputas, Auditoria e Governança

G1. Dispute Resolution (FALTA)

- abrir disputa
- congelar escrow
- contra-evidência
- laudo independente
- decisão e execução

G2. Auditoria read-only + parecer

- painel do auditor
- parecer sem alterar registro

G3. Segregação de funções (SoD)

- quem registra não aprova sozinho
- quem custodia não liquida sozinho

H) Integrações e Dispositivos (IoT/IA)

H1. Device Attestation (FALTA)

- balança, RFID, drones, tratores
- anti-replay, assinatura do device
- confiabilidade do dado

H2. IA (assistência, não autoridade)

- sugere parâmetros e padrões
- nunca “libera” sozinha sem gate/assinatura quando crítico

O que está faltando (lista curta priorizada)

1. **Transformação / BoM / mistura / fracionamento**
2. **Leite: módulo de estabilidade + liquidação**
3. **Disputas & arbitragem** (escrow freeze + decisão)
4. **Garantias/collateral registry** (penhora lógica bem definida)
5. **Fiscal & Document Hub** (NF/CT-e/MDF-e como gates)
6. **Conciliação bancária + estornos/chargeback**

7. **Device attestation** (balança/RFID/drones)

Próximo passo (pra "organizar de vez")

Eu posso transformar esse mapa em um **backlog fechado** (DEV-ready) com:

- módulos → épicos → histórias
- eventos e estados por história
- gates e evidências
- prioridades (MVP / fase 2 / fase 3)

Se você me disser só **qual é o MVP que você quer primeiro**:

- **(1) marketplace + PDV + insumos**, ou
- **(2) custódia boitel + contratos + escrow**, ou
- **(3) leite (resolver calote e preço)**

…eu já te devolvo o backlog completo na ordem certa.

Fechado. Vou desenvolver os **3** (1) marketplace+PDV+insumos, (2) custódia+contratos+escrow, (3) leite (preço+calote).
Como "primeiro", eu escolho **(1) Marketplace + PDV + Insumos**, porque ele cria a **base de identidade + estoque + split + documentos**, que depois sustenta custódia e leite sem retrabalho.

Abaixo está o desenvolvimento em formato **DEV-ready**: módulos → épicos → histórias → eventos/estados/gates/evidências + integrações.

Fase 1 (Primeiro): Marketplace de Insumos + Estoque + PDV + Split

Módulo 1 — Catálogo e Identidade de Produto

Objetivo: nenhum produto existe sem identidade e modo de rastreio.

História 1.1 — Cadastro de Produto com Identidade Obrigatória

- **Dados:** `productId`, `sku`, `identityMode (UNIT|LOT|POOL)`, `identifiers (EAN/QR/RFID/internalCode)`
- **Evento:** `PRODUCT_CREATED`
- **Gate:** `IDENTITY_PRESENT`
- **Evidência:** A

História 1.2 — Regras de rastreio por categoria

- Ex.: medicamento = LOT + validade; equipamento = UNIT + serial; granel = POOL
- **Evento:** `PRODUCT_POLICY_SET`
- **Gate:** policy válida (motor de regras)
- **Evidência:** A

Módulo 2 — Entrada fiscal (NF) e Lotes de Estoque

Objetivo: estoque "oficial" nasce por NF e vira lote rastreável.

História 2.1 — Importar NF (XML/PDF) e gerar `StockLot`

- **Evento:** `INBOUND_NF_IMPORTED`
- **Gate:** NF válida + SKU mapeado
- **Evidência:** DOC (arquivo NF)

História 2.2 — Estoque multi-depósito e FEFO/FIFO

- **Evento:** `STOCK_LOT_CREATED`, `STOCK_MOVED`
- **Gate:** saldo e permissões
- **Evidência:** A (rotina) / B (alto risco)

Módulo 3 — Baixa por Foto/voz (sem digitação)

Objetivo: operador registra retirada em papel; sistema lança via OCR/voz.

História 3.1 — Captura de foto + OCR para `EventDraft`

- **Evento:** `EVIDENCE_CAPTURED`, `EVENT_DRAFT_CREATED`
- **Gate:** hash anti-duplicação + usuário autenticado
- **Evidência:** B (foto do caderno/planilha)

História 3.2 — Validação e conversão do Draft em evento real

- **Evento:** `STOCK_ISSUE_CONFIRMED`
- **Gate:** `qty<=saldo`, SKU válido, local válido
- **Evidência:** B ou A conforme risco

História 3.3 — Fila de pendências (quando OCR falha)

- **Evento:** `EVENT_PENDING_REVIEW`
- **Gate:** revisão por perfil (SoD)
- **Evidência:** A (aprovação)

Módulo 4 — Marketplace (insumos) + modos de negociação

Objetivo: compra e venda por: (a) negociação direta, (b) leilão, (c) sob demanda (contrato).

História 4.1 — Anúncios de insumos (oferta)

- **Evento:** `LISTING_CREATED`
- **Gate:** estoque disponível (lote) ou capacidade (sob demanda)
- **Evidência:** A

História 4.2 — Pedido / proposta / contraproposta

- **Evento:** `OFFER_MADE`, `OFFER_COUNTERED`, `OFFER_ACCEPTED`
- **Gate:** regras comerciais + RBAC
- **Evidência:** A

História 4.3 — Leilão (módulo separado com leiloeiro credenciado)

- **Evento:** `AUCTION_CREATED`, `BID_PLACED`, `BID_WON`
- **Gate:** leiloeiro autorizado + prazos
- **Evidência:** A
- **Regra:** após lance vencedor → **escrow obrigatório**

Módulo 5 — PDV balcão multi-fornecedor + Split

Objetivo: vender cesta com produtos de vários fornecedores e dividir no pagamento.

História 5.1 — PDV consome por lote (não por SKU genérico)

- **Evento:** `PDV_SALE_CREATED`
- **Gate:** `saleLines[].stockLotId` obrigatório
- **Evidência:** A

História 5.2 — SplitPlan automático por fornecedor

- **Evento:** `SPLITPLAN_READY`
- **Gate:** todos itens têm `supplierOrgId` herdado do lote
- **Evidência:** A

História 5.3 — Estorno/devolução com trilha reversa

- **Evento:** `PDV_RETURN_CREATED`, `SPLIT_REVERSAL_APPLIED`
- **Gate:** referenciar `saleLineId` original
- **Evidência:** A/B conforme risco

Módulo 6 — Document Hub (gates fiscais e operacionais)

Objetivo: documento certo no tempo certo.

História 6.1 — Repositório de docs (NF, recibos, laudos)

- **Evento:** `DOC_ATTACHED`
- **Gate:** doc classificado + hash gerado
- **Evidência:** DOC

História 6.2 — Gate "sem documento → sem avançar"

- Ex.: saída interestadual sem CT-e/MDF-e (se aplicável)
- **Evento:** `OPERATION_BLOCKED`
- **Gate:** policy fiscal
- **Evidência:** A

Entregável da Fase 1 (MVP)

- Estoque por lote + entrada fiscal + PDV split multi-fornecedor
- Marketplace de insumos com 3 modos (direto/leilão/sob demanda)
- Foto/voz para baixas e apontamentos (sem digitação)
- Document hub básico + gates

Fase 2: Custódia de ativos + Contratos por ciclo + Escrow obrigatório

Módulo 7 — Custódia de terceiros (boitel, armazém, consignação)

Objetivo: segurança total do dono.

História 7.1 — Check-in com evidência forte

- **Evento:** `ASSET_CHECKIN`
- **Gate:** ID do ativo (animal/brinco ou lote)
- **Evidência:** B (foto+GPS+peso)

História 7.2 — Trava de custódia e "proibir NF sem liberar"

- **Evento:** `CUSTODY_LOCKED`, `INVOICE_BLOCKED`
- **Gate:** ativos terceiros presentes → bloqueia emissão sem split/escrow
- **Evidência:** A

História 7.3 — Liberação do dono + escrow

- **Evento:** `OWNER_RELEASE_SIGNED`, `ESCROW_FUNDED`
- **Gate:** assinatura + fundos ok
- **Evidência:** SIGNATURE + A

Módulo 8 — Contratos por ciclos (cria/recria/engorda; aves; peixes; silvicultura…)

Objetivo: contrato nasce com WorkID + retorno mensurável.

História 8.1 — Template de contrato versionado + assinatura

- **Evento:** `CONTRACT_CREATED`, `CONTRACT_SIGNED`
- **Gate:** WorkID + retorno mínimo definido
- **Evidência:** SIGNATURE

História 8.2 — Marcos por ciclo (gates)

- Ex.: pesagens, vacinas, laudos, vistorias
- **Evento:** `MILESTONE_RECORDED`, `MILESTONE_VERIFIED`
- **Gate:** mínimo aceitável + tolerância
- **Evidência:** B ou B_LAB

Módulo 9 — Escrow/tranches/split por marcos

Objetivo: ninguém recebe sem marco verificado.

História 9.1 — Escrow obrigatório (direto e pós-leilão)

- **Evento:** `ESCROW_CREATED`, `ESCROW_FUNDED`
- **Gate:** contrato ativo
- **Evidência:** A

História 9.2 — Tranches por etapa

- **Evento:** `TRANCHE_RELEASE_REQUESTED`, `TRANCHE_RELEASED`
- **Gate:** milestone verified
- **Evidência:** A (+ evidência do marco)

Entregável da Fase 2

- Custódia segura + bloqueios anti-fraude (NF/transferência)
- Contratos por ciclo com WorkID + gates
- Escrow/tranches/split amarrado a evidências

Fase 3: Leite — estabilidade de preço + anti-calote

Módulo 10 — Milk Settlement & Price Stability

Objetivo: produtor saber o preço e reduzir calote.

História 10.1 — Contrato de fornecimento com fórmula + piso

- `workId`, `volumeCommitted`, `bandTolerance`, `priceFormula`, `floorPrice`
- **Evento:** `MILK_CONTRACT_SIGNED`
- **Gate:** garantia do comprador definida
- **Evidência:** SIGNATURE

História 10.2 — Medição por período (volume + qualidade)

- **Evento:** `MILK_DELIVERY_RECORDED`
- **Gate:** evidência de coleta + laudo (quando aplicável)
- **Evidência:** B + B_LAB

História 10.3 — Pagamento por tranches (semanal/quinzenal) + ajuste no fechamento

- **Evento:** `MILK_TRANCHE_RELEASED`, `MONTHLY_RECONCILED`
- **Gate:** escrow suficiente + SLAs
- **Evidência:** A

Como lidar com volume variável (sem travar o sistema)

- define **volume comprometido** (mínimo)
- define **banda** (ex.: ±15%)
- pagamento por tranche baseado no **real entregue**
- fechamento mensal ajusta diferença (bônus/penalidade) conforme banda

História 10.4 — Anti-calote (garantia + travas)

- se atraso → `DEFAULT_TRIGGERED`
- aciona garantias e trava operações do comprador dentro do Ciclo+:
  - bloqueia novas compras/contratos
  - bloqueia emissão de documentos dentro da plataforma
  - congela ativos/recebíveis vinculados (penhora lógica)
- **Evento:** `COLLATERAL_FROZEN`, `REMEDIATION_PLAN_CREATED`
- **Evidência:** A + auditoria

Entregável da Fase 3

- preço previsível (piso+fórmula)
- pagamento antecipado parcial por escrow/tranches
- laudos como gates
- punição automática e execução de garantias dentro do ecossistema

---

Módulos transversais que precisam existir desde o começo

1. **Disputas & Arbitragem** (abre disputa, congela escrow, decide)
2. **Conciliação bancária + estornos**
3. **Device trust** (balança/RFID/drones)
4. **Risk matrix objetiva** (evento → evidência A/B/DOC/LAUDO)
5. **SoD** (quem registra não aprova sozinho)

Quer que eu já converta isso em "Backlog de DEV" (lista numerada)?

Eu posso te entregar o backlog em 3 colunas:

- **Sprint 1–4 (Fase 1)**
- **Sprint 5–8 (Fase 2)**
- **Sprint 9–12 (Fase 3)**

com:

- nome da história,
- endpoints sugeridos,
- tabelas mínimas,
- e eventos/gates por item.

Abaixo está o **Backlog DEV completo (Sprint 1–12)**, já organizado por **Fase 1 (Marketplace+PDV+Insumos) → Fase 2 (Custódia+Contratos+Escrow) → Fase 3 (Leite)**, com **histórias**, **eventos**, **gates/evidências**, **tabelas mínimas** e **endpoints sugeridos**.

Padrão global: todo endpoint cria **evento** com `orgId + workId (quando aplicável) + evidenceType + hashchain`.

Sprint 1 — Fundação: Identidade, Auditoria, Regras, Estados

1. Identity Service (IDs universais)

- **Histórias**
  1. Gerar `productId`, `stockLotId`, `workId`, `assetId`, `docId`, `splitPlanId`, `escrowId`
  2. Validar unicidade e formato + checksum

- **Eventos**: `ID_ISSUED`
- **Gate/evidência**: A
- **Tabelas mínimas**
  - `identity_registry(id, type, orgId, createdAt, hashSnapshot)`
- **Endpoints**
  - `POST /ids/issue` (type, orgId, metadata) → id
  - `GET /ids/validate/{id}`

2. Auditoria imutável (append-only + hash chain)

- **Histórias**
  1. Persistir `EventEnvelope` (hash + previousHash)
  2. Verificador de integridade (cadeia)
- **Eventos**: qualquer evento do sistema
- **Gate**: bloquear se `hash/previousHash` ausente
- **Tabelas**
  - `audit_log(eventId, orgId, workId, type, payloadJson, hash, previousHash, ts, evidenceRef)`
- **Endpoints**
  - `POST /events` (interno)
  - `GET /audit/verify?from=&to=`

3. Motor de regras + máquina de estados (genérico)

- **Histórias**
  1. Engine de policy (bloqueia/permite)
  2. FSM por entidade (`Work`, `Custody`, `Sale`, `MilkContract`)
- **Eventos**: `OPERATION_BLOCKED`, `STATE_CHANGED`
- **Tabelas**
  - `policies(policyId, version, json)`
  - `state_machine(entityType, version, json)`
- **Endpoints**
  - `POST /policies/publish`
  - `POST /fsm/apply` (entityType, entityId, eventType)

Sprint 2 — Catálogo de Produtos + Políticas de Rastreio

4. Cadastro de produto com identidade obrigatória

- **Histórias**
  1. Produto com `identityMode = UNIT|LOT|POOL`
  2. Código interno + EAN/QR/RFID
- **Eventos**: `PRODUCT_CREATED`
- **Gates**
  - `identityMode` obrigatório
- **Tabelas**
  - `product(productId, orgId, sku, name, identityMode, uom, riskClass, active)`
  - `product_identifiers(productId, type, value)`

- **Endpoints**
  - `POST /products`
  - `GET /products/{productId}`

5. Risk Matrix (evento→evidência mínima) — base

- **Histórias**
  1. Matriz inicial por módulos (estoque, PDV, custódia)
- **Eventos**: `RISK_POLICY_SET`
- **Tabelas**
  - `risk_matrix(eventType, riskLevel, evidenceRequired, extraRequirementsJson)`
- **Endpoints**
  - `POST /risk-matrix/set`

Sprint 3 — Document Hub (NF/Docs) + Entrada Fiscal de Estoque

6. Document Hub

- **Histórias**
  1. Upload e classificação (NF, recibo, laudo, contrato)
  2. Hash do documento + vínculo em Work/StockLot
- **Eventos**: `DOC_ATTACHED`
- **Gate**: doc classificado + hash
- **Tabelas**
  - `documents(docId, orgId, class, fileRef, hash, createdAt)`
  - `document_links(docId, entityType, entityId)`
- **Endpoints**
  - `POST /docs/upload`
  - `POST /docs/link`

7. Importar NF e criar StockLot (fiscal)

- **Histórias**
  1. Criar lote por item da NF
  2. Fornecedor + custo + validade
- **Eventos**: `INBOUND_NF_IMPORTED`, `STOCK_LOT_CREATED`
- **Gates**
  - NF obrigatória (DOC)
  - SKU reconhecido
- **Tabelas**
  - `stock_lot(stockLotId, orgId, productId, supplierOrgId, qty, uom, unitCost, expiry, nfDocId, locationId, status)`
- **Endpoints**
  - `POST /stock/inbound/nf`

Sprint 4 — Estoque: Movimentações + OCR/Voz (sem digitação) + Pendências

8. Movimentação interna e consumo

- **Histórias**
  1. Transferência depósito→depósito
  2. Consumo/baixa por operação
- **Eventos**: `STOCK_MOVED`, `STOCK_ISSUE_CONFIRMED`
- **Gates**
  - `qty<=saldo`
  - se crítico: evidência B
- **Tabelas**
  - `stock_balance(stockLotId, locationId, qtyOnHand)`
- **Endpoints**
  - `POST /stock/move`
  - `POST /stock/issue`

9. OCR/Foto e Voz para lançamento

- **Histórias**
  1. `EventDraft` criado por foto/áudio
  2. OCR extrai campos → valida → confirma ou pendência
- **Eventos**: `EVENT_DRAFT_CREATED`, `EVENT_PENDING_REVIEW`, `EVENT_DRAFT_APPLIED`
- **Gates**
  - anti-duplicação por `evidenceHash`
  - revisão SoD quando OCR falhar
- **Tabelas**
  - `event_draft(draftId, orgId, type, extractedJson, evidenceDocId, status)`
  - `pending_queue(itemId, type, refId, reason, assignedRole)`
- **Endpoints**
  - `POST /drafts/from-photo`
  - `POST /drafts/from-voice`
  - `POST /drafts/{draftId}/apply`
  - `POST /pending/{id}/resolve`

✅ **Fim da Fase 1 — Parte "estoque forte + entrada fiscal + sem digitação" pronta**

Sprint 5 — Marketplace (Insumos): Ofertas, Propostas, Negociação direta

10. Listings (oferta de insumos)

- **Histórias**
  1. Oferta vinculada a lote (ou capacidade)
  2. Reserva de saldo (hold)
- **Eventos**: `LISTING_CREATED`, `STOCK_RESERVED`

- **Gates**
  - saldo disponível
  - produto com identidade
- **Tabelas**
  - `listing(listingId, orgId, type, productId, stockLotId?, qty, price, termsJson, status)`
  - `stock_hold(holdId, stockLotId, qty, expiresAt, reasonRef)`
- **Endpoints**
  - `POST /market/listings`
  - `POST /market/listings/{id}/reserve`

11. Propostas e contrapropostas

- **Eventos**: `OFFER_MADE`, `OFFER_COUNTERED`, `OFFER_ACCEPTED`
- **Gates**: RBAC + regras comerciais
- **Tabelas**
  - `offer(offerId, listingId, buyerOrgId, price, qty, status, expiresAt)`
- **Endpoints**
  - `POST /market/offers`
  - `POST /market/offers/{id}/accept`

Sprint 6 — PDV balcão multi-fornecedor + Split + Pagamento

12. PDV por lote (cesta multi-fornecedor)

- **Histórias**
  1. criar venda com itens apontando `stockLotId`
  2. baixar estoque automaticamente
- **Eventos**: `PDV_SALE_CREATED`, `STOCK_ISSUE_CONFIRMED`
- **Gates**
  - item sem lote → bloqueia
- **Tabelas**
  - `pdv_sale(saleId, storeOrgId, total, status, createdAt)`
  - `pdv_sale_line(lineId, saleId, stockLotId, productId, supplierOrgId, qty, unitPrice)`
- **Endpoints**
  - `POST /pdv/sales`

13. SplitPlan automático

- **Eventos**: `SPLITPLAN_READY`, `SPLIT_SETTLED`
- **Gates**
  - todos itens com `supplierOrgId`
  - split READY antes de “finalizar”
- **Tabelas**
  - `split_plan(splitPlanId, sourceType, sourceId, status)`
  - `split_line(splitPlanId, beneficiaryOrgId, amount)`
- **Endpoints**
  - `POST /split/calc/from-sale/{saleId}`
  - `POST /split/settle/{splitPlanId}`

14. Estorno/devolução (reversão de split)

- **Eventos**: `PDV_RETURN_CREATED`, `SPLIT_REVERSAL_APPLIED`
- **Gates**: referenciar venda original
- **Endpoints**
  - `POST /pdv/returns`

✅ **Fim da Fase 1 — Marketplace + PDV + Split**

Sprint 7 — Custódia (boitel/terceiros): check-in, trava, bloqueios anti-fraude

15. Check-in de ativo com evidência forte

- **Eventos**: `ASSET_CHECKIN`, `CUSTODY_LOCKED`
- **Gates**
  - animal: brinco obrigatório
  - evidência B (foto+GPS+peso)
- **Tabelas**
  - `asset(assetId, type, identityRef, ownerOrgId, custodianOrgId, status)`
  - `custody(custodyId, assetId, status, lockedAt)`
- **Endpoints**
  - `POST /custody/checkin`
  - `POST /custody/{id}/lock`

16. Bloquear NF/transferência se ativo de terceiros presente

- **Eventos**: `INVOICE_BLOCKED`
- **Gates**: se `custody.status=LOCKED` → bloqueia saída fiscal sem liberação do dono + split
- **Endpoints**
  - `POST /compliance/check-invoice-eligibility`

Sprint 8 — Contratos por ciclo + WorkID + Gates + Assinatura

17. Templates jurídicos versionados (cláusulas)

- **Eventos**: `CONTRACT_TEMPLATE_PUBLISHED`
- **Tabelas**
  - `contract_template(templateId, version, hash, json)`
- **Endpoints**
  - `POST /contracts/templates/publish`

18. Criar contrato com retorno mensurável

- **Eventos**: `CONTRACT_CREATED`, `CONTRACT_SIGNED`
- **Gates**
  - `workId` obrigatório
  - `ReturnExpectation` obrigatório
  - assinatura digital
- **Tabelas**
  - `work(workId, type, orgId, counterpartyOrgId, status, returnJson)`
  - `contract(contractId, workId, templateId, status, signedAt)`
- **Endpoints**
  - `POST /contracts`
  - `POST /contracts/{id}/sign`

19. Marcos (milestones) + evidência + verificação

- **Eventos**: `MILESTONE_RECORDED`, `MILESTONE_VERIFIED`
- **Gates**
  - mínimo aceitável e tolerância
  - evidência B / B_LAB
- **Tabelas**
  - `milestone(milestoneId, workId, type, status, metricsJson, evidenceDocId)`
- **Endpoints**
  - `POST /work/{workId}/milestones`
  - `POST /milestones/{id}/verify`

✅ **Fim da Fase 2 — Custódia + Contratos + Gates**

Sprint 9 — Escrow/Tranches/Split por marcos + Garantias (base)

20. Escrow obrigatório (direto e pós-leilão)

- **Eventos**: `ESCROW_CREATED`, `ESCROW_FUNDED`
- **Gates**: sem escrow → bloqueia liquidação
- **Tabelas**
  - `escrow(escrowId, workId, amount, status)`
- **Endpoints**
  - `POST /escrow/create`
  - `POST /escrow/{id}/funded`

21. Tranches por etapa

- **Eventos**: `TRANCHE_RELEASE_REQUESTED`, `TRANCHE_RELEASED`
- **Gates**
  - milestone verified
- **Tabelas**
  - `tranche(trancheId, escrowId, milestoneId, amount, status)`
- **Endpoints**

- `POST /escrow/{id}/tranches/release`

22. Garantias (Collateral Registry — MVP)

- **Eventos**: `COLLATERAL_REGISTERED`, `COLLATERAL_FROZEN`, `COLLATERAL_RELEASED`
- **Tabelas**
  - `collateral(collateralId, workId, type, ref, value, status)`
- **Endpoints**
  - `POST /collateral/register`
  - `POST /collateral/{id}/freeze`
  - `POST /collateral/{id}/release`

Sprint 10 — Leite: contrato com piso/fórmula + banda de volume + laudos

23. Milk Contract (WorkID) com preço previsível

- **Eventos**: `MILK_CONTRACT_SIGNED`
- **Gates**
  - `floorPrice` + `priceFormula`
  - garantia do comprador cadastrada
- **Tabelas**
  - `milk_contract(milkContractId, workId, volumeCommitted, bandPct, formulaJson, floorPrice)`
- **Endpoints**
  - `POST /milk/contracts`
  - `POST /milk/contracts/{id}/sign`

24. Coletas/entregas (volume) + laudos (qualidade)

- **Eventos**: `MILK_DELIVERY_RECORDED`, `LAB_REPORT_ATTACHED`
- **Gates**
  - evidência B (coleta) + B_LAB (laudo) quando aplicável
- **Tabelas**
  - `milk_delivery(deliveryId, milkContractId, date, volume, evidenceDocId, status)`
  - `lab_report(reportId, deliveryId, metricsJson, docId)`
- **Endpoints**
  - `POST /milk/deliveries`
  - `POST /milk/deliveries/{id}/lab`

Sprint 11 — Leite: tranches (pagamento) + fechamento mensal + anti-calote

25. Tranches semanais/quinzenais com escrow

- **Eventos**: `MILK_TRANCHE_RELEASED`
- **Gates**
  - escrow suficiente
  - entregas registradas
- **Tabelas**
  - `milk_tranche(trancheId, milkContractId, period, amount, status)`
- **Endpoints**
  - `POST /milk/tranches/release`

26. Fechamento mensal (reconciliação por banda)

- **Eventos**: `MONTHLY_RECONCILED`
- **Gates**
  - aplicar banda (±%)
  - gerar ajuste
- **Tabelas**
  - `milk_settlement(settlementId, milkContractId, month, calcJson, deltaAmount)`
- **Endpoints**
  - `POST /milk/settlement/close`

27. Default e travas do comprador

- **Eventos**: `DEFAULT_TRIGGERED`, `COLLATERAL_FROZEN`, `BUYER_BLOCKED`
- **Gates**
  - atraso > SLA
- **Endpoints**
  - `POST /milk/default/trigger`

Sprint 12 — Disputas, Governança (SoD), Verificação de auditoria, Device trust

28. Disputas & arbitragem (MVP)

- **Eventos**: `DISPUTE_OPENED`, `ESCROW_FROZEN`, `DISPUTE_RESOLVED`
- **Gates**
  - congelar parcial/todo
  - decisão por perfil auditor/árbitro
- **Tabelas**
  - `dispute(disputeId, workId, reason, status, decisionJson)`
- **Endpoints**
  - `POST /disputes`
  - `POST /disputes/{id}/resolve`

29. Segregação de função (SoD)

- **Eventos**: `SOD_VIOLATION_BLOCKED`
- **Tabelas**
  - `sod_rules(ruleId, json)`
- **Endpoints**
  - `POST /governance/sod/publish`

30. Device attestation (MVP)

- **Eventos**: `DEVICE_REGISTERED`, `DEVICE_DATA_ACCEPTED`, `DEVICE_DATA_REJECTED`
- **Tabelas**
  - `device(deviceId, orgId, type, publicKey, trustLevel)`
- **Endpoints**
  - `POST /devices/register`
  - `POST /devices/data`

Resumo final (o que você terá ao fim das 12 sprints)

✅ Marketplace de insumos (direto/leilão/sob demanda)
✅ Estoque por lote com entrada fiscal e baixas via foto/voz (sem digitar)
✅ PDV balcão com split por fornecedor e reversões
✅ Custódia boitel/terceiros com travas e liberação do dono + escrow
✅ Contratos por ciclos com WorkID + retorno mensurável + marcos verificáveis
✅ Leite com piso/fórmula, banda de volume, tranches e anti-calote
✅ Disputas, auditoria verificável, SoD e device trust

Próximo passo para eu “fechar” 100% para o DEV

Me diga qual stack vocês vão usar:

- **Backend**: Node/Nest, Java/Spring, Python/FastAPI, ou .NET?
- **DB**: Postgres?
- **Fila**: Rabbit/Kafka/SQS?

Com isso, eu te devolvo a mesma estrutura já como:

- **ERD (tabelas relacionais)**
- **contratos de API (OpenAPI)**
- **event types padronizados** (lista única)
- **policies iniciais** do motor de regras (JSON pronto).

Perfeito — essa stack (NestJS + Firestore + Expo RN + Firebase Auth + Asaas) **encaixa bem** com o Ciclo+, mas para ficar **100% alinhada** com o núcleo

(evento→regras→estados→evidência→auditoria imutável→liquidação condicionada) você precisa ajustar **3 coisas grandes** e preencher alguns módulos.

Vou organizar em:

1. o que está ótimo e já encaixa
2. falhas/lacunas (o que falta para cumprir "imutável / hashchain / liquidação condicionada / evidência")
3. modelo recomendado de coleções Firestore + índices
4. padrão de "EventEnvelope" e hashchain (como fazer no Firestore)
5. módulos adicionais mínimos (sem estourar o MVP)

---

1) O que está bem encaixado

✅ **Multi-tenant por `tenantId` em tudo** + Guards (`FirebaseAuthGuard`, `TenantGuard`, `RolesGuard`) — isso casa com o "filtro orgId obrigatório" do núcleo.
✅ **Movimentações imutáveis em subcoleção `inventory/{itemId}/moves`** — é exatamente o "evento" no sentido da patente.
✅ **Máquina de estados para requests** — também casa com a máquina de estados formal.
✅ **N:N por subcoleções** — adequado para Firestore e para evitar duplicação bruta.
✅ **Offline-ready** no app com listeners e cache — excelente para operação de campo.

---

2) Falhas/lacunas que precisam ser corrigidas (as mais importantes)

Falha A — "Imutável" no Firestore não é automaticamente "imutável"

Mesmo que você só "adicione documentos", alguém com permissão errada pode atualizar/excluir. Para ficar alinhado ao Ciclo+ (append-only + resistente a adulteração), você precisa de:

- **Regra de segurança** que **proíba update/delete** na coleção de auditoria (e em moves) para clientes.
- **Servidor** (Nest/Cloud Function) como único escritor.
- **Hashchain**: cada evento precisa carregar `hash` e `previousHash` e permitir verificação.

Sem isso, "auditável" vira "histórico editável".

Falha B — Está faltando o "EventEnvelope" unificado (core)

Hoje você tem módulos e endpoints, mas o núcleo exige que **toda operação crítica** gere um evento com o mesmo envelope:

- auth + RBAC + tenant filter + motor de regras + FSM + evidência + auditoria + (quando aplicável) settlement.

Se cada módulo gravar log do seu jeito, você perde consistência e prova.

Falha C — Liquidação condicionada / escrow / split ainda não está modelado

Você citou Asaas (billing/webhooks), mas o Ciclo+ exige:

- **escrow lógico** (custódia de valor no sistema),
- **splitPlan** (linhas por beneficiário),
- **liberação por marcos verificados**,
- trilha de auditoria disso.

Asaas pode pagar/splitar, mas o **"escrow lógico + gates"** precisa existir no seu modelo e no motor de regras.

Falha D — Evidência (foto/voz/OCR) precisa virar "primeira classe"

Você quer foto de anotação manual e comando de voz. Isso deve virar:

- `evidence` (metadata + hash + storageRef),
- regra de evidência por risco,
- vínculo com evento,
- e "anti-duplicação" (hash).

Falha E — Disputas e congelamento (escrow freeze) não existe

No agro real (peso, qualidade, entrega, calote) isso é inevitável. Sem módulo de disputa, o sistema trava e perde confiança.

---

3) Coleções Firestore recomendadas (simples e coerentes)

Você pode manter seu modelo atual, mas sugiro **padronizar** assim (tudo com `tenantId`):

Core

- `tenants/{tenantId}`
- `users/{uid}` (ou `tenants/{tenantId}/users/{uid}` se preferir escopo)
- `roles/{roleId}` / `permissions/{permId}` (ou embutido em claims)

Catálogo e fornecedores

- `catalog_items/{catalogItemId}` (itens "globais" do tenant)

- `supplier_items/{supplierItemId}` (vincula fornecedor↔item↔preço↔condições)
- `suppliers/{supplierId}`

Estoque

- `inventory_items/{itemId}` (o "produto operacional" do estoque: aponta p/ catalogItem)
- `inventory_items/{itemId}/moves/{moveId}` (**append-only**)
- `inventory_items/{itemId}/balances/{locationId}` (saldo atual por local)
- `locations/{locationId}`

Requests (seu fluxo request state machine)

- `requests/{requestId}`

Evidências e documentos

- `evidences/{evidenceId}` (foto, áudio, OCR, doc fiscal, laudo)
- `docs/{docId}` (se quiser separar fiscal/contrato/laudo)

Auditoria imutável universal (obrigatória)

- `audit_events/{eventId}` (**append-only**, hashchain)

Financeiro lógico (Ciclo+)

- `works/{workId}` (contratos/projetos/ciclos — mesmo que no MVP você use pouco)
- `escrows/{escrowId}`
- `split_plans/{splitPlanId}`
- `settlements/{settlementId}`

Disputas

- `disputes/{disputeId}`

---

4) Padrão obrigatório: EventEnvelope + Hashchain no Firestore

4.1. EventEnvelope (o "formato único" do sistema)

Todo endpoint relevante deve gerar um `audit_event` com campos mínimos:

- `eventId`
- `tenantId`
- `actorUid`
- `actorRole`

- `eventType` (ex.: `STOCK_MOVE_CREATED`, `REQUEST_APPROVED`, `SPLITPLAN_READY`)
- `entityType` / `entityId` (ex.: `inventory_item`, `request`, `work`)
- `timestamp`
- `payload` (resumo determinístico — cuidado com campos voláteis)
- `evidenceIds[]`
- `stateBefore` / `stateAfter` (quando houver FSM)
- `previousHash`
- `hash`

4.2. Como fazer hashchain no Firestore (padrão praticável)

- Mantenha um documento "ponteiro" por tenant, algo como:
  - `tenants/{tenantId}/audit_head` com `{ lastEventId, lastHash }`
- Ao gravar um evento:

1. transação: lê `audit_head`
2. define `previousHash = audit_head.lastHash`
3. calcula `hash = sha256(canonicalJson(eventWithoutHash) + previousHash)`
4. grava `audit_events/{eventId}` (create-only)
5. atualiza `audit_head.lastHash = hash`

Importante: isso deve rodar **no backend** (Nest) com Admin SDK (não no app).

4.3. Regras Firestore para "imutável"

Para `audit_events` e `moves`:

- permitir `create` só por servidor (ideal: somente Admin SDK)
- negar `update` e `delete` sempre

Se você precisa que o app escreva "rascunhos" (OCR/voz):

- escreva em `drafts/` (editável)
- o backend "promove" para `moves` e `audit_events` (imutável)

---

5) Ajustes nos módulos NestJS (mínimos e alinhados ao núcleo)

Você listou módulos: Auth, Tenants, Suppliers, Items, Requests, StockMovements, StockBalance, SupplierItems, Catalog.

Eu adicionaria **mais 5** (pequenos, mas críticos):

1. **AuditModule**
   - serviço `emitEvent()` que todo módulo chama
   - aplica hashchain e escreve `audit_events`
2. **EvidenceModule**
   - upload/ref + hash + OCR job + anti-duplicação

    - liga evidência ao evento
3. **StateMachineModule** (ou “WorkflowModule”)
    - regra única de transição (requests, works, custody…)
    - valida campos obrigatórios por estado
4. **RulesEngineModule**
    - validações pre-execução (limite, saldo, permissão extra, compliance)
    - risco → exige evidência A/B/DOC/LAUDO
5. **SettlementModule** (escrow/split)
    - integra Asaas via webhooks
    - mantém escrow lógico + splitPlan + libera por gates

Isso mantém o “Ciclo+ core” sem virar um monstro.

---

6) Índices Firestore (o que você citou + os necessários)

Além de `tenantId + status + createdAt` e `tenantId + itemId + performedAt`, inclua:

- `inventory_items`:
    - `tenantId + catalogItemId`
    - `tenantId + locationId` (se consultar por local)
- `moves` (se for coleção por item, você filtra por subcoleção; ok)
- `audit_events`:
    - `tenantId + timestamp desc`
    - `tenantId + entityType + entityId + timestamp desc`
    - `tenantId + eventType + timestamp desc`
- `evidences`:
    - `tenantId + createdAt desc`
    - `tenantId + hash` (para anti-duplicação)

---

7) Checklist de falhas que eu já corrigiria antes de escalar

Segurança

- Claims JWT no Firebase Auth: `tenantId`, `role`, `permissions` ✅
- Firestore rules: checar token vs `resource.data.tenantId` ✅
- **Mas**: garantir que coleções imutáveis não tenham update/delete (audit, moves) ❗
- Segregação de função (SoD): quem cria não aprova sozinho (mínimo para pagamentos, ajustes e disputas) ❗

Integridade

- Hashchain por tenant (audit_events) ❗
- Anti-replay de evidências (hash do arquivo) ❗
- Idempotência nos endpoints (principalmente webhooks Asaas e sync offline) ❗

Operação real

- “Pendências” para OCR/voz (não bloquear a fazenda por falha de leitura) ✅/❗
- Disputas com freeze de escrow ❗

---

8) O que está faltando (módulos faltantes) — versão curta

1. **AuditModule (unificado + hashchain)**
2. **EvidenceModule (foto/voz/OCR + hash + vínculo)**
3. **RulesEngine + RiskMatrix** (evidência por risco)
4. **SettlementModule (escrow/split/tranches + Asaas webhooks)**
5. **DisputeModule (freeze + decisão + execução)**
6. (opcional fase 2) **TransformModule (mistura/fracionamento/BoM)**

Próximo passo (sem te perguntar de novo “o que você quer”)

Se você quiser que eu “deixe pronto para DEV”, eu te devolvo na próxima mensagem:

- **Estrutura de coleções** com exemplos de documentos (JSON) para:
  - `inventory_items`, `moves`, `balances`, `requests`, `audit_events`, `evidences`, `works`, `escrows`, `split_plans`
- **Lista fechada de eventTypes** (padrão do sistema)
- **Fluxos de request FSM** e **gates mínimos** por estado
- **Padrão de endpoints NestJS** (controllers/services) e onde chamar `emitEvent()`
- **Regras Firestore** (esqueleto) para multi-tenant + imutabilidade

Se preferir, eu começo pelo **MVP real (Sprints 1–4)** na sua stack: estoque + requests + OCR/voz + auditoria unificada, e já deixo tudo no padrão Ciclo+.

Firestore — Coleções e documentos (MVP Sprints 1–4)

1.1 `tenants/{tenantId}`

```
{
  "tenantId": "t_abc",
  "name": "Fazenda X",
  "createdAt": 1730000000000,
  "auditHead": { "lastEventId": "evt_...", "lastHash": "..." }
}
```

1.2 `catalog_items/{catalogItemId}`

Catálogo “padrão” do tenant (um item pode virar inventário em vários locais).

```
{
  "tenantId": "t_abc",
  "catalogItemId": "cat_001",
  "name": "Sal mineral 25kg",
  "sku": "SAL25",
```

```
  "uom": "KG",
  "identityMode": "LOT",
  "riskClass": "LOW",
  "active": true,
  "createdAt": 1730000000000
}
```

1.3 `inventory_items/{itemId}`

Representa o item “operacional” (referencia o catálogo, tem política e pode ter múltiplos lotes).

```
{
  "tenantId": "t_abc",
  "itemId": "inv_001",
  "catalogItemId": "cat_001",
  "defaultLocationId": "loc_01",
  "policy": {
    "issueEvidenceRequired": "A",
    "lossEvidenceRequired": "B"
  },
  "createdAt": 1730000000000
}
```

1.4 `inventory_items/{itemId}/lots/{stockLotId}`

Lote fiscal (entra via NF/Doc fiscal).

```
{
  "tenantId": "t_abc",
  "stockLotId": "lot_0001",
  "itemId": "inv_001",
  "supplierId": "sup_01",
  "qtyInitial": 1000,
  "qtyAvailable": 1000,
  "uom": "KG",
  "unitCost": 4.5,
  "expiry": 1760000000000,
  "nfDocId": "doc_nf_123",
  "locationId": "loc_01",
  "status": "ACTIVE",
  "createdAt": 1730000000000
}
```

1.5 `inventory_items/{itemId}/moves/{moveId}` (append-only)

Movimentação imutável (entrada/saída/perda/transferência).

```
{
  "tenantId": "t_abc",
  "moveId": "mv_0001",
  "itemId": "inv_001",
  "stockLotId": "lot_0001",
  "type": "ISSUE",
  "qty": 25,
  "uom": "KG",
  "locationId": "loc_01",
  "balanceBefore": 1000,
  "balanceAfter": 975,
  "performedBy": { "uid": "u_1", "role": "OPERATOR" },
```

```
  "performedAt": 1730000000000,
  "evidenceIds": ["ev_001"],
  "workId": null,
  "eventId": "evt_0001"
}
```

1.6 `inventory_items/{itemId}/balances/{locationId}`

Saldo atual (derivado, pode ser atualizado por transação do backend).

```
{
  "tenantId": "t_abc",
  "itemId": "inv_001",
  "locationId": "loc_01",
  "qtyOnHand": 975,
  "updatedAt": 1730000000000,
  "lastMoveId": "mv_0001"
}
```

1.7 `requests/{requestId}` (FSM)

Solicitações com máquina de estados.

```
{
  "tenantId": "t_abc",
  "requestId": "req_0001",
  "type": "STOCK_REQUEST",
  "status": "pending",
  "items": [{ "itemId": "inv_001", "qty": 25, "uom": "KG" }],
  "createdBy": { "uid": "u_1" },
  "approvedBy": null,
  "createdAt": 1730000000000,
  "updatedAt": 1730000000000,
  "eventIds": ["evt_req_1"]
}
```

1.8 `evidences/{evidenceId}`

Evidência (foto/voz/doc), com hash (anti-duplicação).

```
{
  "tenantId": "t_abc",
  "evidenceId": "ev_001",
  "type": "PHOTO",
  "riskType": "B",
  "storageRef": "gs://bucket/evidences/t_abc/ev_001.jpg",
  "sha256": "ab12...",
  "ocr": {
    "status": "DONE",
    "extracted": { "sku": "SAL25", "qty": 25, "location": "loc_01",
"date": "2026-02-05" },
    "confidence": 0.88
  },
  "createdBy": { "uid": "u_1" },
  "createdAt": 1730000000000
}
```

1.9 `event_drafts/{draftId}` (editável)

Rascunho criado pelo app (offline), validado e “promovido” pelo backend.

```
{
  "tenantId": "t_abc",
  "draftId": "dr_0001",
  "draftType": "STOCK_ISSUE",
  "status": "PENDING_VALIDATION",
  "proposed": { "itemId": "inv_001", "stockLotId": "lot_0001", "qty":
25, "locationId": "loc_01" },
  "evidenceIds": ["ev_001"],
  "createdBy": { "uid": "u_1" },
  "createdAt": 1730000000000
}
```

1.10 `audit_events/{eventId}` (append-only + hashchain)

Evento unificado (core).

```
{
  "tenantId": "t_abc",
  "eventId": "evt_0001",
  "eventType": "STOCK_ISSUE_CONFIRMED",
  "entityType": "inventory_move",
  "entityId": "mv_0001",
  "actor": { "uid": "u_1", "role": "OPERATOR", "permissions":
["stock.issue"] },
  "timestamp": 1730000000000,
  "stateBefore": null,
  "stateAfter": null,
  "evidenceIds": ["ev_001"],
  "payload": { "itemId": "inv_001", "stockLotId": "lot_0001", "qty":
25, "locationId": "loc_01" },
  "previousHash": "0000...genesisOrPrev",
  "hash": "9f12..."
}
```

---

2) Lista fechada de `eventType` (MVP Sprints 1–4)

Identidade / Auditoria

- `ID_ISSUED`
- `AUDIT_EVENT_APPENDED`
- `AUDIT_CHAIN_VERIFIED`
- `OPERATION_BLOCKED`

Evidência / OCR / Draft

- `EVIDENCE_UPLOADED`
- `OCR_STARTED`
- `OCR_DONE`
- `EVENT_DRAFT_CREATED`
- `EVENT_DRAFT_VALIDATED`
- `EVENT_DRAFT_REJECTED`
- `EVENT_PENDING_REVIEW`
- `EVENT_DRAFT_APPLIED`

Catálogo / Itens

- `CATALOG_ITEM_CREATED`
- `INVENTORY_ITEM_CREATED`

Estoque

- `INBOUND_NF_IMPORTED`
- `STOCK_LOT_CREATED`
- `STOCK_MOVED`
- `STOCK_ISSUE_CONFIRMED`
- `STOCK_LOSS_RECORDED`

Requests (FSM)

- `REQUEST_CREATED`
- `REQUEST_APPROVED`
- `REQUEST_FULFILLMENT_STARTED`
- `REQUEST_DONE`
- `REQUEST_REJECTED`

---

3) FSM de Requests + campos obrigatórios por estado

Estados: `pending → approved → fulfillment → done` (+ `rejected`)

Transições válidas

- `pending -> approved` (aprovador)
- `approved -> fulfillment` (operador)
- `fulfillment -> done` (operador)
- `pending -> rejected` (aprovador)
- `approved -> rejected` (gestor, opcional)

Campos obrigatórios

- `pending`: `type`, `items[]`, `createdBy`
- `approved`: `approvedBy`, `approvedAt`
- `fulfillment`: `fulfillmentStartedBy`, `fulfillmentStartedAt`
- `done`: `doneBy`, `doneAt`, `moveIds[]` (as movimentações que atenderam)
- `rejected`: `rejectedBy`, `rejectedAt`, `reason`

Gate: qualquer tentativa fora da transição → `OPERATION_BLOCKED` + auditar tentativa.

---

4) Risk Matrix (mínimo para começar)

Regra: `eventType -> evidenceRequired`

- `STOCK_ISSUE_CONFIRMED`: A (rotina) **ou** B se `qty` acima de limite ou item `riskClass != LOW`
- `STOCK_LOSS_RECORDED`: B (foto) e, se `riskClass=HIGH`, `DOC/LAUDO`
- `INBOUND_NF_IMPORTED`: DOC (NF)
- `REQUEST_APPROVED`: A (mas exige SoD mínimo: aprovador != solicitante)

Implementação: tabela/coleção `risk_matrix` e um serviço `EvidencePolicyService` no backend.

---

5) NestJS — padrão de módulos e onde chamar o core

5.1 Módulos mínimos (MVP)

Você já tem: Auth, Tenants, Suppliers, Items, Requests, StockMovements, StockBalance, SupplierItems, Catalog.
Adicione (pequenos e centrais):

1. `AuditModule`
2. `EvidenceModule`
3. `RulesModule` (policy + risk matrix)
4. `FsmModule` (requests)

Serviço central obrigatório

`CoreEventService.emit()`: todo controller chama antes de retornar sucesso.

Fluxo recomendado por endpoint:

1. Guards: `FirebaseAuthGuard` → `TenantGuard` → `RolesGuard`
2. `RulesService.preValidate(dto, context)`
3. `FsmService.applyTransition(...)` (se tiver estado)
4. `DomainService.execute()` (ex.: criar move em transação)
5. `AuditService.appendEvent(eventEnvelope)` (hashchain)
6. Retornar response

5.2 Endpoints (MVP Sprints 1–4)

Auth

- `GET /api/auth/me`

Evidence

- `POST /api/evidences/upload` → cria evidence + `EVIDENCE_UPLOADED`
- `POST /api/evidences/{id}/ocr/start`
- `GET /api/evidences/{id}`

Drafts

- `POST /api/drafts` (cria draft)
- `POST /api/drafts/{id}/apply` (backend valida e promove)

Catalog/Items

- `POST /api/catalog`
- `POST /api/items`

Stock

- `POST /api/stock/inbound/nf`
- `POST /api/stock/movements` (move/issue/loss)
- `GET /api/balance/current?itemId=&locationId=`

Requests

- `POST /api/requests`
- `POST /api/requests/{id}/approve`
- `POST /api/requests/{id}/start-fulfillment`
- `POST /api/requests/{id}/done`

---

6) Firestore Rules — esqueleto seguro (multi-tenant + imutável)

Ideia: o app só escreve em coleções "operacionais" permitidas; as coleções **imutáveis** só o backend (Admin) escreve.
No Firestore Rules, **Admin SDK ignora rules**, então basta garantir que **clientes não conseguem update/delete**.

Exemplo (conceitual):

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

    function isAuthed() { return request.auth != null; }
    function tenantMatches(datai(t) { return
request.auth.token.tenantId == t; }
    function hasRole(r) { return request.auth.token.role == r; }

    // Catálogo/itens (cliente pode criar/ler dentro do tenant)
    match /catalog_items/{id} {
```

```
      allow read: if isAuthed() &&
tenantMatches(resource.data.tenantId);
      allow create: if isAuthed() &&
tenantMatches(request.resource.data.tenantId);
      allow update, delete: if false; // preferível: somente servidor
    }

    // Drafts (cliente pode criar e atualizar)
    match /event_drafts/{id} {
      allow read: if isAuthed() &&
tenantMatches(resource.data.tenantId);
      allow create: if isAuthed() &&
tenantMatches(request.resource.data.tenantId);
      allow update: if isAuthed() &&
tenantMatches(resource.data.tenantId); // limitado por campos, ideal
      allow delete: if false;
    }

    // Evidence (cliente cria; update só servidor para OCR fields)
    match /evidences/{id} {
      allow read: if isAuthed() &&
tenantMatches(resource.data.tenantId);
      allow create: if isAuthed() &&
tenantMatches(request.resource.data.tenantId);
      allow update: if false; // OCR escreve via backend/Admin
      allow delete: if false;
    }

    // Moves e Audit: imutáveis para cliente
    match /inventory_items/{itemId}/moves/{moveId} {
      allow read: if isAuthed() &&
tenantMatches(resource.data.tenantId);
      allow create, update, delete: if false; // somente backend/Admin
    }

    match /audit_events/{eventId} {
      allow read: if isAuthed() &&
tenantMatches(resource.data.tenantId);
      allow create, update, delete: if false; // somente backend/Admin
    }
  }
}
```

(Depois refinamos para permitir leitura por role, e escrita limitada em drafts.)

---

7) Frontend (Expo + RN) — padrão offline sem bagunçar integridade

Recomendação prática

- O app cria:
  - `evidences` (upload → storageRef + hash)
  - `event_drafts` (com proposed + evidenceIds)
- O backend "promove":
  - cria `moves` (imutável)

    - atualiza `balances`
    - cria `audit_events` (hashchain)
    - marca draft como `APPLIED` (ou rejeita/pendência)

Com React Query + listeners:

- query `drafts` (status pendente)
- query `balances`
- query `requests`

---

8) Sprints 1–4 já prontos na sua stack (o que o dev implementa primeiro)

Sprint 1

- `AuditModule` (appendEvent + head pointer + hashchain)
- `RulesModule` (só valida tenant/role + estrutura)
- Firestore rules base (multi-tenant + imutáveis bloqueados)
- `GET /auth/me`

Sprint 2

- `CatalogModule` + `ItemsModule` com identityMode
- `RiskMatrix` básico
- `EvidenceModule` upload (sem OCR ainda)

Sprint 3

- `DocumentHub` (docs + link)
- Entrada NF → cria `lots` + audita

Sprint 4

- `DraftsModule` + "apply draft"
- `StockMovements` (issue/move/loss) com validações
- Pendências (quando OCR/extração falhar)

Próximo passo (eu já sigo sem te pedir mais nada)

Na próxima mensagem eu te entrego:

1. **Modelos DTO** (NestJS) para: `CreateDraftDto`, `ApplyDraftDto`, `CreateMovementDto`, `CreateRequestDto`, `ApproveRequestDto`
2. **Padrão de transação Firestore** (Admin SDK) para:
    - aplicar movimento (balanceBefore/After)

- append audit event com hashchain

3. **Tabela de policies iniciais (JSON)** para:
   - `qty <= saldo`
   - evidência por risco
   - SoD básico (approve != create)

) DTOs NestJS (com validação)

1.1 Evidência (upload metadata)

Upload real do arquivo pode ser via Firebase Storage direto do app; o backend recebe os metadados e grava `evidences/{id}`.

```
// evidence.dto.ts
import { IsArray, IsEnum, IsNotEmpty, IsObject, IsOptional, IsString }
from 'class-validator';

export enum EvidenceType { PHOTO='PHOTO', AUDIO='AUDIO', DOC='DOC' }
export enum EvidenceRisk { A='A', B='B', DOC='DOC', B_LAB='B_LAB' }

export class CreateEvidenceDto {
  @IsEnum(EvidenceType) type: EvidenceType;
  @IsEnum(EvidenceRisk) riskType: EvidenceRisk;

  @IsString() @IsNotEmpty() storageRef: string; // gs://... or
https://...
  @IsString() @IsNotEmpty() sha256: string;

  @IsOptional() @IsObject() ocr?: {
    status?: 'PENDING'|'DONE'|'FAILED';
    extracted?: Record<string, any>;
    confidence?: number;
  };

  @IsOptional() @IsArray() tags?: string[];
}
```

1.2 Draft (criado pelo app, editável)

```
// drafts.dto.ts
import { IsArray, IsEnum, IsNotEmpty, IsObject, IsOptional, IsString,
ValidateNested } from 'class-validator';
import { Type } from 'class-transformer';

export enum DraftType { STOCK_ISSUE='STOCK_ISSUE',
STOCK_MOVE='STOCK_MOVE', STOCK_LOSS='STOCK_LOSS' }

export class ProposedStockOpDto {
  @IsString() @IsNotEmpty() itemId: string;
  @IsOptional() @IsString() stockLotId?: string; // opcional se vocês
escolherem lot via FEFO/FIFO
  @IsString() @IsNotEmpty() locationId: string;

  @IsNotEmpty() qty: number;
  @IsString() @IsNotEmpty() uom: string;

  @IsOptional() @IsString() reason?: string; // ex: "consumo",
"perda", "transferência"
```

```
  @IsOptional() @IsString() toLocationId?: string; // para
transferências
}

export class CreateDraftDto {
  @IsEnum(DraftType) draftType: DraftType;

  @ValidateNested() @Type(() => ProposedStockOpDto)
  proposed: ProposedStockOpDto;

  @IsArray() evidenceIds: string[]; // pelo menos 1 se policy exigir

  @IsOptional() @IsString() workId?: string;
  @IsOptional() @IsObject() clientMeta?: Record<string, any>; // ex:
offline timestamp, deviceId
}

export class ApplyDraftDto {
  @IsOptional() @IsString() reviewerComment?: string;
  @IsOptional() @IsString() overrideStockLotId?: string; // caso OCR
errou o lote
}
```

1.3 Movimentação (endpoint “direto”, quando não usar draft)

```
// movements.dto.ts
import { IsEnum, IsNotEmpty, IsOptional, IsString, IsArray } from
'class-validator';

export enum MovementType { INBOUND='INBOUND', ISSUE='ISSUE',
MOVE='MOVE', LOSS='LOSS' }

export class CreateMovementDto {
  @IsEnum(MovementType) type: MovementType;

  @IsString() @IsNotEmpty() itemId: string;
  @IsOptional() @IsString() stockLotId?: string;

  @IsString() @IsNotEmpty() locationId: string;
  @IsOptional() @IsString() toLocationId?: string; // MOVE

  @IsNotEmpty() qty: number;
  @IsString() @IsNotEmpty() uom: string;

  @IsArray() evidenceIds: string[];

  @IsOptional() @IsString() workId?: string;
  @IsOptional() @IsString() reason?: string;
}
```

1.4 Requests (FSM)

```
// requests.dto.ts
import { IsArray, IsEnum, IsNotEmpty, IsOptional, IsString,
ValidateNested } from 'class-validator';
import { Type } from 'class-transformer';

export enum RequestStatus { pending='pending', approved='approved',
fulfillment='fulfillment', done='done', rejected='rejected' }
```

```
export class RequestItemDto {
  @IsString() @IsNotEmpty() itemId: string;
  @IsNotEmpty() qty: number;
  @IsString() @IsNotEmpty() uom: string;
}

export class CreateRequestDto {
  @IsString() @IsNotEmpty() type: string; // ex: STOCK_REQUEST
  @IsArray() @ValidateNested({ each: true }) @Type(() =>
RequestItemDto)
  items: RequestItemDto[];

  @IsOptional() @IsString() notes?: string;
  @IsOptional() @IsString() workId?: string;
}

export class ApproveRequestDto {
  @IsOptional() @IsString() comment?: string;
}
export class RejectRequestDto {
  @IsString() @IsNotEmpty() reason: string;
}
```

---

2) Firestore Admin SDK — transações padrão (imutável + balanceBefore/After + audit hashchain)

2.1 Funções utilitárias (hash determinístico)

Use canonical JSON para garantir reprodutibilidade.

```
import * as crypto from 'crypto';

export function canonicalJson(obj: any): string {
  // ordena chaves recursivamente
  const sorter = (x: any): any => {
    if (Array.isArray(x)) return x.map(sorter);
    if (x && typeof x === 'object') {
      return Object.keys(x).sort().reduce((acc, k) => {
        acc[k] = sorter(x[k]);
        return acc;
      }, {} as any);
    }
    return x;
  };
  return JSON.stringify(sorter(obj));
}

export function sha256Hex(s: string): string {
  return crypto.createHash('sha256').update(s).digest('hex');
}
```

2.2 `appendAuditEvent()` (hashchain por tenant)

- `tenants/{tenantId}` guarda `auditHead.lastHash`.
- `audit_events/{eventId}` é **create-only** (imutável).

```
import { Firestore } from '@google-cloud/firestore';

export async function appendAuditEvent(
  db: Firestore,
  tenantId: string,
  event: Omit<any, 'hash' | 'previousHash'>
) {
  const tenantRef = db.collection('tenants').doc(tenantId);
  const eventRef = db.collection('audit_events').doc(event.eventId);

  await db.runTransaction(async (tx) => {
    const tenantSnap = await tx.get(tenantRef);
    if (!tenantSnap.exists) throw new Error('TENANT_NOT_FOUND');

    const head = tenantSnap.get('auditHead') || {};
    const previousHash = head.lastHash || 'GENESIS';

    const eventToHash = { ...event, previousHash };
    const hash = sha256Hex(canonicalJson(eventToHash));

    // create-only
    tx.create(eventRef, { ...eventToHash, hash });

    tx.update(tenantRef, {
      auditHead: { lastEventId: event.eventId, lastHash: hash }
    });
  });
}
```

2.3 `applyStockIssueFromDraft()` (promove Draft → Move + Balance + Audit)

- Lê draft + evidence
- Valida políticas (`qty<=saldo`, evidência obrigatória, tenantId)
- Cria move **append-only**
- Atualiza balance
- Marca draft como APPLIED (editável)
- Escreve audit_event com hashchain

```
export async function applyStockIssueFromDraft(db: Firestore,
tenantId: string, draftId: string, actor: any) {
  const draftRef = db.collection('event_drafts').doc(draftId);

  await db.runTransaction(async (tx) => {
    const draftSnap = await tx.get(draftRef);
    if (!draftSnap.exists) throw new Error('DRAFT_NOT_FOUND');

    const draft = draftSnap.data()!;
    if (draft.tenantId !== tenantId) throw new
Error('TENANT_MISMATCH');
    if (draft.status !== 'PENDING_VALIDATION') throw new
Error('DRAFT_NOT_APPLICABLE');

    const { itemId, stockLotId, locationId, qty, uom } =
draft.proposed;

    // balance doc
    const balRef = db.collection('inventory_items').doc(itemId)
      .collection('balances').doc(locationId);
```

```
    const balSnap = await tx.get(balRef);
    const qtyOnHand = balSnap.exists ? (balSnap.data()!.qtyOnHand ||
0) : 0;

    if (qty <= 0) throw new Error('INVALID_QTY');
    if (qty > qtyOnHand) throw new Error('INSUFFICIENT_STOCK');

    // move id
    const moveId =
`mv_${Date.now()}_${Math.random().toString(16).slice(2)}`;
    const moveRef = db.collection('inventory_items').doc(itemId)
      .collection('moves').doc(moveId);

    const balanceBefore = qtyOnHand;
    const balanceAfter = qtyOnHand - qty;

    const moveDoc = {
      tenantId, moveId, itemId, stockLotId: stockLotId ?? null,
      type: 'ISSUE', qty, uom, locationId,
      balanceBefore, balanceAfter,
      performedBy: { uid: actor.uid, role: actor.role },
      performedAt: Date.now(),
      evidenceIds: draft.evidenceIds || [],
      workId: draft.workId ?? null
    };

    // append-only move
    tx.create(moveRef, moveDoc);

    // update balance (create or update)
    if (balSnap.exists) {
      tx.update(balRef, { qtyOnHand: balanceAfter, updatedAt:
Date.now(), lastMoveId: moveId });
    } else {
      tx.create(balRef, { tenantId, itemId, locationId, qtyOnHand:
balanceAfter, updatedAt: Date.now(), lastMoveId: moveId });
    }

    // mark draft applied
    tx.update(draftRef, { status: 'APPLIED', appliedAt: Date.now(),
moveId });
  });

  // audit event outside the transaction (ok), OR you can embed it in
the same transaction if you prefer.
  const eventId =
`evt_${Date.now()}_${Math.random().toString(16).slice(2)}`;
  await appendAuditEvent(db, tenantId, {
    tenantId,
    eventId,
    eventType: 'STOCK_ISSUE_CONFIRMED',
    entityType: 'inventory_move',
    entityId: `moves:${draftId}`,
    actor,
    timestamp: Date.now(),
    evidenceIds: [], // opcional: ler draft novamente ou passar como
argumento
    payload: { draftId }
  });
}
```

Nota prática: se você quiser que o audit e o apply sejam "100% atômicos", dá para colocar o `appendAuditEvent()` dentro da mesma `runTransaction` (fazendo `tx.create` do audit + update do head). Eu recomendo isso para operações críticas.

---

3) Policies iniciais (JSON) para o RulesEngine

3.1 Estoque: `qty <= saldo` + evidência por risco

```
{
  "policyId": "POLICY_STOCK_ISSUE_V1",
  "version": "1.0.0",
  "rules": [
    {
      "id": "BLOCK_IF_QTY_INVALID",
      "when": { "eventType": "STOCK_ISSUE_CONFIRMED", "qty_lte": 0 },
      "then": { "action": "BLOCK", "reason": "INVALID_QTY" }
    },
    {
      "id": "BLOCK_IF_INSUFFICIENT_STOCK",
      "when": { "eventType": "STOCK_ISSUE_CONFIRMED",
"qty_gt_balance": true },
      "then": { "action": "BLOCK", "reason": "INSUFFICIENT_STOCK" }
    },
    {
      "id": "EVIDENCE_REQUIRED_BY_RISK",
      "when": { "eventType_in": ["STOCK_ISSUE_CONFIRMED",
"STOCK_LOSS_RECORDED"] },
      "then": {
        "action": "REQUIRE_EVIDENCE",
        "matrix": {
          "STOCK_ISSUE_CONFIRMED": { "default": "A",
"ifRiskClassNotLOW": "B", "ifQtyAbove": { "threshold": 200,
"evidence": "B" } },
          "STOCK_LOSS_RECORDED": { "default": "B", "ifRiskClassHIGH":
"DOC" }
        }
      }
    }
  ]
}
```

3.2 Requests: FSM + bloqueio de transições inválidas

```
{
  "policyId": "POLICY_REQUEST_FSM_V1",
  "version": "1.0.0",
  "rules": [
    {
      "id": "ALLOW_VALID_TRANSITIONS_ONLY",
      "when": { "entityType": "request" },
      "then": {
        "action": "ENFORCE_FSM",
        "allowed": {
          "pending": ["approved", "rejected"],
          "approved": ["fulfillment", "rejected"],
          "fulfillment": ["done"],
          "done": [],
          "rejected": []
        }
      }
```

```
    }
  ]
}
```

3.3 SoD mínimo: aprovador ≠ solicitante

```
{
  "policyId": "POLICY_SOD_MIN_V1",
  "version": "1.0.0",
  "rules": [
    {
      "id": "BLOCK_SELF_APPROVAL",
      "when": { "eventType": "REQUEST_APPROVED",
"actorUid_eq_createdByUid": true },
      "then": { "action": "BLOCK", "reason":
"SOD_SELF_APPROVAL_NOT_ALLOWED" }
    }
  ]
}
```

3.4 Evidência anti-duplicação (hash)

```
{
  "policyId": "POLICY_EVIDENCE_ANTIREPLAY_V1",
  "version": "1.0.0",
  "rules": [
    {
      "id": "BLOCK_DUPLICATE_EVIDENCE_HASH",
      "when": { "evidenceSha256_alreadyExists": true },
      "then": { "action": "BLOCK", "reason": "DUPLICATE_EVIDENCE" }
    }
  ]
}
```

---

4) Onde isso entra no seu NestJS (bem direto)

4.1 Fluxo do endpoint `POST /api/drafts/{id}/apply`

1. `FirebaseAuthGuard` + `TenantGuard` + `RolesGuard`
2. `RulesService.preValidateDraftApply(draft, actor)`
3. `StockService.applyIssueFromDraft()` (transação)
4. `AuditService.appendEvent()` (ideal: na mesma transação)
5. retorna `{ moveId, balanceAfter }`

4.2 Fluxo do endpoint `POST /api/requests/{id}/approve`

1. Guards
2. `FsmService.assertTransition(pending -> approved)`
3. `SoDService.assertApproverNotRequester()`
4. grava status + audit event

---

5) O que você já tem e o que falta implementar (MVP)

Você já descreveu quase tudo. Para o MVP “Ciclo+ padrão” faltam só estes pontos críticos:

- ✅ tenantId em tudo + rules e guards
- ✅ moves append-only com balanceBefore/After
- ✅ request FSM
- ❗ **AuditModule unificado + hashchain**
- ❗ **EvidenceModule com hash anti-duplicação + OCR status**
- ❗ **Drafts + apply por backend** (para suportar offline e evitar fraude)

Se você quiser, eu já te passo na próxima mensagem um **esqueleto de Controllers/Services NestJS** (arquivos e assinaturas) para:

- `AuditService`
- `EvidenceService`
- `DraftsService`
- `StockMovementsService`
- `RequestsService`

…no estilo Nest (providers + inject Firestore Admin).

Estrutura sugerida de pastas

```
src/
  common/
    firestore.provider.ts
    guards/
      firebase-auth.guard.ts
      tenant.guard.ts
      roles.guard.ts
    decorators/
      actor.decorator.ts
    utils/
      hash.util.ts
      id.util.ts
  audit/
    audit.module.ts
    audit.service.ts
  evidence/
    evidence.module.ts
    evidence.controller.ts
    evidence.service.ts
    dto/
      evidence.dto.ts
  drafts/
    drafts.module.ts
    drafts.controller.ts
    drafts.service.ts
    dto/
      drafts.dto.ts
  stock/
    stock.module.ts
    stock.controller.ts
    stock.service.ts
    dto/
      movements.dto.ts
  requests/
```

```
    requests.module.ts
    requests.controller.ts
    requests.service.ts
    dto/
      requests.dto.ts
    fsm/
      request-fsm.service.ts
  app.module.ts
```

---

1) Provider Firestore (Admin SDK)

```
// src/common/firestore.provider.ts
import { Provider } from '@nestjs/common';
import * as admin from 'firebase-admin';

export const FIRESTORE = 'FIRESTORE';

export const firestoreProvider: Provider = {
  provide: FIRESTORE,
  useFactory: () => {
    // Assumindo que admin.initializeApp() já foi chamado em algum
lugar
    // Caso não, chame aqui com credenciais.
    return admin.firestore();
  },
};
```

---

2) Utils: hash determinístico + IDs

```
// src/common/utils/hash.util.ts
import * as crypto from 'crypto';

export function canonicalJson(obj: any): string {
  const sorter = (x: any): any => {
    if (Array.isArray(x)) return x.map(sorter);
    if (x && typeof x === 'object') {
      return Object.keys(x).sort().reduce((acc, k) => {
        acc[k] = sorter(x[k]);
        return acc;
      }, {} as any);
    }
    return x;
  };
  return JSON.stringify(sorter(obj));
}

export function sha256Hex(s: string): string {
  return crypto.createHash('sha256').update(s).digest('hex');
}
// src/common/utils/id.util.ts
export function makeId(prefix: string) {
  return
`${prefix}_${Date.now()}_${Math.random().toString(16).slice(2)}`;
}
```

---

3) Decorator Actor (pega uid/role/permissions/tenantId do request)

```
// src/common/decorators/actor.decorator.ts
import { createParamDecorator, ExecutionContext } from
'@nestjs/common';
```

```
export const Actor = createParamDecorator((_, ctx: ExecutionContext)
=> {
  const req = ctx.switchToHttp().getRequest();
  const token = req.user?.token || {}; // dependerá do seu
FirebaseAuthGuard
  return {
    uid: req.user?.uid,
    role: token.role,
    permissions: token.permissions || [],
    tenantId: token.tenantId,
  };
});
```

---

4) AuditService (hashchain por tenant) + módulo

4.1 AuditService

```
// src/audit/audit.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { canonicalJson, sha256Hex } from '../common/utils/hash.util';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class AuditService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  /**
   * Append-only + hashchain por tenant.
   * Recomendação: chamar dentro da MESMA transação do domínio quando
crítico.
   */
  async appendEvent(tenantId: string, event: Omit<any, 'hash' |
'previousHash'>, tx?: FirebaseFirestore.Transaction) {
    const tenantRef = this.db.collection('tenants').doc(tenantId);
    const eventId = event.eventId ?? makeId('evt');
    const eventRef = this.db.collection('audit_events').doc(eventId);

    const runner = async (t: FirebaseFirestore.Transaction) => {
      const tenantSnap = await t.get(tenantRef);
      if (!tenantSnap.exists) throw new Error('TENANT_NOT_FOUND');

      const head = tenantSnap.get('auditHead') || {};
      const previousHash = head.lastHash || 'GENESIS';

      const eventToHash = { ...event, tenantId, eventId, previousHash
};
      const hash = sha256Hex(canonicalJson(eventToHash));

      // create-only (imutável)
      t.create(eventRef, { ...eventToHash, hash });

      t.update(tenantRef, { auditHead: { lastEventId: eventId,
lastHash: hash } });

      return { eventId, hash, previousHash };
    };
```

```
    if (tx) return runner(tx);
    return this.db.runTransaction(runner);
  }
}
```

4.2 AuditModule

```
// src/audit/audit.module.ts
import { Module } from '@nestjs/common';
import { AuditService } from './audit.service';
import { firestoreProvider } from '../common/firestore.provider';

@Module({
  providers: [firestoreProvider, AuditService],
  exports: [AuditService],
})
export class AuditModule {}
```

---

5) EvidenceModule (hash anti-duplicação) + controller

5.1 DTO

```
// src/evidence/dto/evidence.dto.ts
import { IsArray, IsEnum, IsNotEmpty, IsObject, IsOptional, IsString }
from 'class-validator';

export enum EvidenceType { PHOTO='PHOTO', AUDIO='AUDIO', DOC='DOC' }
export enum EvidenceRisk { A='A', B='B', DOC='DOC', B_LAB='B_LAB' }

export class CreateEvidenceDto {
  @IsEnum(EvidenceType) type: EvidenceType;
  @IsEnum(EvidenceRisk) riskType: EvidenceRisk;

  @IsString() @IsNotEmpty() storageRef: string;
  @IsString() @IsNotEmpty() sha256: string;

  @IsOptional() @IsObject() ocr?: {
    status?: 'PENDING'|'DONE'|'FAILED';
    extracted?: Record<string, any>;
    confidence?: number;
  };

  @IsOptional() @IsArray() tags?: string[];
}
```

5.2 EvidenceService

```
// src/evidence/evidence.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class EvidenceService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}
```

```
  async createEvidence(tenantId: string, actor: any, dto: any) {
    // Anti-duplicação por sha256 (por tenant)
    const dupQuery = await this.db.collection('evidences')
      .where('tenantId', '==', tenantId)
      .where('sha256', '==', dto.sha256)
      .limit(1)
      .get();

    if (!dupQuery.empty) {
      throw new Error('DUPLICATE_EVIDENCE');
    }

    const evidenceId = makeId('ev');
    const ref = this.db.collection('evidences').doc(evidenceId);

    const doc = {
      tenantId,
      evidenceId,
      ...dto,
      createdBy: { uid: actor.uid },
      createdAt: Date.now(),
    };

    await ref.create(doc); // create-only
    return doc;
  }

  async getEvidence(tenantId: string, evidenceId: string) {
    const snap = await
this.db.collection('evidences').doc(evidenceId).get();
    if (!snap.exists) throw new Error('EVIDENCE_NOT_FOUND');
    const data = snap.data()!;
    if (data.tenantId !== tenantId) throw new
Error('TENANT_MISMATCH');
    return data;
  }
}
```

5.3 EvidenceController

```
// src/evidence/evidence.controller.ts
import { Body, Controller, Get, Param, Post, UseGuards } from
'@nestjs/common';
import { EvidenceService } from './evidence.service';
import { CreateEvidenceDto } from './dto/evidence.dto';
import { Actor } from '../common/decorators/actor.decorator';
// Seus guards
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';

@Controller('/api/evidences')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class EvidenceController {
  constructor(private readonly evidence: EvidenceService) {}

  @Post()
  async create(@Actor() actor: any, @Body() dto: CreateEvidenceDto) {
```

```
    return this.evidence.createEvidence(actor.tenantId, actor, dto);
  }

  @Get(':id')
  async get(@Actor() actor: any, @Param('id') id: string) {
    return this.evidence.getEvidence(actor.tenantId, id);
  }
}
```

5.4 EvidenceModule

```
// src/evidence/evidence.module.ts
import { Module } from '@nestjs/common';
import { firestoreProvider } from '../common/firestore.provider';
import { EvidenceController } from './evidence.controller';
import { EvidenceService } from './evidence.service';

@Module({
  providers: [firestoreProvider, EvidenceService],
  controllers: [EvidenceController],
  exports: [EvidenceService],
})
export class EvidenceModule {}
```

---

6) DraftsModule (offline) + apply draft → move/balance/audit

6.1 DTO

```
// src/drafts/dto/drafts.dto.ts
import { IsArray, IsEnum, IsNotEmpty, IsObject, IsOptional, IsString,
ValidateNested } from 'class-validator';
import { Type } from 'class-transformer';

export enum DraftType { STOCK_ISSUE='STOCK_ISSUE',
STOCK_MOVE='STOCK_MOVE', STOCK_LOSS='STOCK_LOSS' }

export class ProposedStockOpDto {
  @IsString() @IsNotEmpty() itemId: string;
  @IsOptional() @IsString() stockLotId?: string;
  @IsString() @IsNotEmpty() locationId: string;

  @IsNotEmpty() qty: number;
  @IsString() @IsNotEmpty() uom: string;

  @IsOptional() @IsString() toLocationId?: string;
  @IsOptional() @IsString() reason?: string;
}

export class CreateDraftDto {
  @IsEnum(DraftType) draftType: DraftType;

  @ValidateNested() @Type(() => ProposedStockOpDto)
  proposed: ProposedStockOpDto;

  @IsArray() evidenceIds: string[];

  @IsOptional() @IsString() workId?: string;
  @IsOptional() @IsObject() clientMeta?: Record<string, any>;
```

```
}

export class ApplyDraftDto {
  @IsOptional() @IsString() reviewerComment?: string;
  @IsOptional() @IsString() overrideStockLotId?: string;
}
```

6.2 DraftsService (inclui transação e chama AuditService)

```
// src/drafts/drafts.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';
import { AuditService } from '../audit/audit.service';

@Injectable()
export class DraftsService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly audit: AuditService,
  ) {}

  async createDraft(tenantId: string, actor: any, dto: any) {
    const draftId = makeId('dr');
    const ref = this.db.collection('event_drafts').doc(draftId);
    const doc = {
      tenantId,
      draftId,
      ...dto,
      status: 'PENDING_VALIDATION',
      createdBy: { uid: actor.uid },
      createdAt: Date.now(),
    };
    await ref.create(doc);
    return doc;
  }

  async applyDraft(tenantId: string, actor: any, draftId: string, dto:
any) {
    const draftRef = this.db.collection('event_drafts').doc(draftId);

    return this.db.runTransaction(async (tx) => {
      const snap = await tx.get(draftRef);
      if (!snap.exists) throw new Error('DRAFT_NOT_FOUND');

      const draft = snap.data()!;
      if (draft.tenantId !== tenantId) throw new
Error('TENANT_MISMATCH');
      if (draft.status !== 'PENDING_VALIDATION') throw new
Error('DRAFT_NOT_APPLICABLE');

      // Apenas MVP: tratar STOCK_ISSUE. (MOVE/LOSS são análogos)
      if (draft.draftType !== 'STOCK_ISSUE') throw new
Error('DRAFT_TYPE_NOT_SUPPORTED_MVP');

      const proposed = draft.proposed || {};
      const itemId = proposed.itemId;
      const locationId = proposed.locationId;
      const qty = Number(proposed.qty);
```

```
      const uom = proposed.uom;
      const stockLotId = dto?.overrideStockLotId ??
proposed.stockLotId ?? null;

      if (!itemId || !locationId || !uom || !Number.isFinite(qty))
throw new Error('INVALID_DRAFT_FIELDS');
      if (qty <= 0) throw new Error('INVALID_QTY');

      // Balance
      const balRef =
this.db.collection('inventory_items').doc(itemId).collection('balances
').doc(locationId);
      const balSnap = await tx.get(balRef);
      const qtyOnHand = balSnap.exists ? (balSnap.data()!.qtyOnHand ||
0) : 0;
      if (qty > qtyOnHand) throw new Error('INSUFFICIENT_STOCK');

      // Move append-only
      const moveId = makeId('mv');
      const moveRef =
this.db.collection('inventory_items').doc(itemId).collection('moves').
doc(moveId);

      const balanceBefore = qtyOnHand;
      const balanceAfter = qtyOnHand - qty;

      const moveDoc = {
        tenantId,
        moveId,
        itemId,
        stockLotId,
        type: 'ISSUE',
        qty,
        uom,
        locationId,
        balanceBefore,
        balanceAfter,
        performedBy: { uid: actor.uid, role: actor.role },
        performedAt: Date.now(),
        evidenceIds: draft.evidenceIds || [],
        workId: draft.workId ?? null,
      };

      tx.create(moveRef, moveDoc);

      // Update balance (derived)
      if (balSnap.exists) {
        tx.update(balRef, { qtyOnHand: balanceAfter, updatedAt:
Date.now(), lastMoveId: moveId });
      } else {
        tx.create(balRef, { tenantId, itemId, locationId, qtyOnHand:
balanceAfter, updatedAt: Date.now(), lastMoveId: moveId });
      }

      // Update draft
      tx.update(draftRef, {
        status: 'APPLIED',
        appliedAt: Date.now(),
        moveId,
        reviewerComment: dto?.reviewerComment ?? null,
      });
```

```
      // AUDIT EVENT (ideal: dentro da mesma transação)
      await this.audit.appendEvent(tenantId, {
        eventId: makeId('evt'),
        eventType: 'STOCK_ISSUE_CONFIRMED',
        entityType: 'inventory_move',
        entityId: moveId,
        actor: { uid: actor.uid, role: actor.role, permissions:
actor.permissions || [] },
        timestamp: Date.now(),
        evidenceIds: draft.evidenceIds || [],
        payload: { draftId, itemId, stockLotId, qty, uom, locationId,
balanceBefore, balanceAfter },
        stateBefore: null,
        stateAfter: null,
      }, tx);

      return { moveId, balanceAfter };
    });
  }
}
```

6.3 DraftsController + Module

```
// src/drafts/drafts.controller.ts
import { Body, Controller, Param, Post, UseGuards } from
'@nestjs/common';
import { DraftsService } from './drafts.service';
import { CreateDraftDto, ApplyDraftDto } from './dto/drafts.dto';
import { Actor } from '../common/decorators/actor.decorator';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';

@Controller('/api/drafts')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class DraftsController {
  constructor(private readonly drafts: DraftsService) {}

  @Post()
  async create(@Actor() actor: any, @Body() dto: CreateDraftDto) {
    return this.drafts.createDraft(actor.tenantId, actor, dto);
  }

  @Post(':id/apply')
  async apply(@Actor() actor: any, @Param('id') id: string, @Body()
dto: ApplyDraftDto) {
    return this.drafts.applyDraft(actor.tenantId, actor, id, dto);
  }
}
// src/drafts/drafts.module.ts
import { Module } from '@nestjs/common';
import { firestoreProvider } from '../common/firestore.provider';
import { DraftsController } from './drafts.controller';
import { DraftsService } from './drafts.service';
import { AuditModule } from '../audit/audit.module';

@Module({
  imports: [AuditModule],
```

```
  providers: [firestoreProvider, DraftsService],
  controllers: [DraftsController],
})
export class DraftsModule {}
```

---

7) StockModule (endpoint direto para movimentos — opcional no MVP)

```
// src/stock/dto/movements.dto.ts
import { IsArray, IsEnum, IsNotEmpty, IsOptional, IsString } from
'class-validator';

export enum MovementType { ISSUE='ISSUE', MOVE='MOVE', LOSS='LOSS' }

export class CreateMovementDto {
  @IsEnum(MovementType) type: MovementType;
  @IsString() @IsNotEmpty() itemId: string;
  @IsOptional() @IsString() stockLotId?: string;
  @IsString() @IsNotEmpty() locationId: string;
  @IsOptional() @IsString() toLocationId?: string;
  @IsNotEmpty() qty: number;
  @IsString() @IsNotEmpty() uom: string;
  @IsArray() evidenceIds: string[];
}
// src/stock/stock.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { AuditService } from '../audit/audit.service';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class StockService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly audit: AuditService,
  ) {}

  async createMovement(tenantId: string, actor: any, dto: any) {
    // Para MVP, sugerir usar drafts; este endpoint é "atalho".
    // A lógica é similar ao applyDraft.
    const { itemId, locationId, qty, uom } = dto;
    if (qty <= 0) throw new Error('INVALID_QTY');

    return this.db.runTransaction(async (tx) => {
      const balRef =
this.db.collection('inventory_items').doc(itemId).collection('balances
').doc(locationId);
      const balSnap = await tx.get(balRef);
      const qtyOnHand = balSnap.exists ? (balSnap.data()!.qtyOnHand ||
0) : 0;
      if (dto.type !== 'INBOUND' && qty > qtyOnHand) throw new
Error('INSUFFICIENT_STOCK');

      const moveId = makeId('mv');
      const moveRef =
this.db.collection('inventory_items').doc(itemId).collection('moves').
doc(moveId);

      const balanceBefore = qtyOnHand;
      const balanceAfter = dto.type === 'INBOUND' ? qtyOnHand + qty :
qtyOnHand - qty;
```

```
      tx.create(moveRef, {
        tenantId, moveId, itemId,
        stockLotId: dto.stockLotId ?? null,
        type: dto.type,
        qty, uom,
        locationId,
        toLocationId: dto.toLocationId ?? null,
        balanceBefore, balanceAfter,
        performedBy: { uid: actor.uid, role: actor.role },
        performedAt: Date.now(),
        evidenceIds: dto.evidenceIds || []
      });

      if (balSnap.exists) {
        tx.update(balRef, { qtyOnHand: balanceAfter, updatedAt:
Date.now(), lastMoveId: moveId });
      } else {
        tx.create(balRef, { tenantId, itemId, locationId, qtyOnHand:
balanceAfter, updatedAt: Date.now(), lastMoveId: moveId });
      }

      await this.audit.appendEvent(tenantId, {
        eventId: makeId('evt'),
        eventType: `STOCK_${dto.type}_RECORDED`,
        entityType: 'inventory_move',
        entityId: moveId,
        actor: { uid: actor.uid, role: actor.role, permissions:
actor.permissions || [] },
        timestamp: Date.now(),
        evidenceIds: dto.evidenceIds || [],
        payload: { ...dto, balanceBefore, balanceAfter },
        stateBefore: null,
        stateAfter: null,
      }, tx);

      return { moveId, balanceAfter };
    });
  }
}
// src/stock/stock.controller.ts
import { Body, Controller, Post, UseGuards } from '@nestjs/common';
import { StockService } from './stock.service';
import { CreateMovementDto } from './dto/movements.dto';
import { Actor } from '../common/decorators/actor.decorator';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';

@Controller('/api/stock')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class StockController {
  constructor(private readonly stock: StockService) {}

  @Post('/movements')
  async createMovement(@Actor() actor: any, @Body() dto:
CreateMovementDto) {
    return this.stock.createMovement(actor.tenantId, actor, dto);
  }
}
```

```
// src/stock/stock.module.ts
import { Module } from '@nestjs/common';
import { firestoreProvider } from '../common/firestore.provider';
import { StockController } from './stock.controller';
import { StockService } from './stock.service';
import { AuditModule } from '../audit/audit.module';

@Module({
  imports: [AuditModule],
  providers: [firestoreProvider, StockService],
  controllers: [StockController],
})
export class StockModule {}
```

---

8) RequestsModule (FSM + SoD mínimo)

8.1 FSM service

```
// src/requests/fsm/request-fsm.service.ts
import { Injectable } from '@nestjs/common';

@Injectable()
export class RequestFsmService {
  private allowed: Record<string, string[]> = {
    pending: ['approved', 'rejected'],
    approved: ['fulfillment', 'rejected'],
    fulfillment: ['done'],
    done: [],
    rejected: [],
  };

  assertTransition(from: string, to: string) {
    const allowedTo = this.allowed[from] || [];
    if (!allowedTo.includes(to)) throw new
Error('FSM_INVALID_TRANSITION');
  }
}
```

8.2 RequestsService + Controller

```
// src/requests/requests.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { AuditService } from '../audit/audit.service';
import { makeId } from '../common/utils/id.util';
import { RequestFsmService } from './fsm/request-fsm.service';

@Injectable()
export class RequestsService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly audit: AuditService,
    private readonly fsm: RequestFsmService,
  ) {}

  async create(tenantId: string, actor: any, dto: any) {
    const requestId = makeId('req');
```

```
    const ref = this.db.collection('requests').doc(requestId);

    const doc = {
      tenantId,
      requestId,
      type: dto.type,
      status: 'pending',
      items: dto.items,
      notes: dto.notes ?? null,
      workId: dto.workId ?? null,
      createdBy: { uid: actor.uid },
      createdAt: Date.now(),
      updatedAt: Date.now(),
      approvedBy: null,
    };

    await ref.create(doc);

    await this.audit.appendEvent(tenantId, {
      eventId: makeId('evt'),
      eventType: 'REQUEST_CREATED',
      entityType: 'request',
      entityId: requestId,
      actor: { uid: actor.uid, role: actor.role, permissions:
actor.permissions || [] },
      timestamp: Date.now(),
      evidenceIds: [],
      payload: { requestId, type: dto.type, items: dto.items },
      stateBefore: null,
      stateAfter: 'pending',
    });

    return doc;
  }

  async approve(tenantId: string, actor: any, requestId: string, dto:
any) {
    const ref = this.db.collection('requests').doc(requestId);

    await this.db.runTransaction(async (tx) => {
      const snap = await tx.get(ref);
      if (!snap.exists) throw new Error('REQUEST_NOT_FOUND');

      const req = snap.data()!;
      if (req.tenantId !== tenantId) throw new
Error('TENANT_MISMATCH');

      // SoD: aprovador != criador
      if (req.createdBy?.uid === actor.uid) throw new
Error('SOD_SELF_APPROVAL_NOT_ALLOWED');

      this.fsm.assertTransition(req.status, 'approved');

      tx.update(ref, {
        status: 'approved',
        approvedBy: { uid: actor.uid },
        approvedAt: Date.now(),
        updatedAt: Date.now(),
      });

      await this.audit.appendEvent(tenantId, {
```

```
        eventId: makeId('evt'),
        eventType: 'REQUEST_APPROVED',
        entityType: 'request',
        entityId: requestId,
        actor: { uid: actor.uid, role: actor.role, permissions:
actor.permissions || [] },
        timestamp: Date.now(),
        evidenceIds: [],
        payload: { requestId, comment: dto?.comment ?? null },
        stateBefore: req.status,
        stateAfter: 'approved',
      }, tx);
    });

    return { requestId, status: 'approved' };
  }
}
// src/requests/requests.controller.ts
import { Body, Controller, Param, Post, UseGuards } from
'@nestjs/common';
import { RequestsService } from './requests.service';
import { CreateRequestDto, ApproveRequestDto } from
'./dto/requests.dto';
import { Actor } from '../common/decorators/actor.decorator';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';

@Controller('/api/requests')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class RequestsController {
  constructor(private readonly requests: RequestsService) {}

  @Post()
  async create(@Actor() actor: any, @Body() dto: CreateRequestDto) {
    return this.requests.create(actor.tenantId, actor, dto);
  }

  @Post(':id/approve')
  async approve(@Actor() actor: any, @Param('id') id: string, @Body()
dto: ApproveRequestDto) {
    return this.requests.approve(actor.tenantId, actor, id, dto);
  }
}
// src/requests/requests.module.ts
import { Module } from '@nestjs/common';
import { firestoreProvider } from '../common/firestore.provider';
import { AuditModule } from '../audit/audit.module';
import { RequestsController } from './requests.controller';
import { RequestsService } from './requests.service';
import { RequestFsmService } from './fsm/request-fsm.service';

@Module({
  imports: [AuditModule],
  providers: [firestoreProvider, RequestsService, RequestFsmService],
  controllers: [RequestsController],
})
export class RequestsModule {}
```

---

9) AppModule (juntando tudo)

```
// src/app.module.ts
import { Module } from '@nestjs/common';
import { AuditModule } from './audit/audit.module';
import { EvidenceModule } from './evidence/evidence.module';
import { DraftsModule } from './drafts/drafts.module';
import { StockModule } from './stock/stock.module';
import { RequestsModule } from './requests/requests.module';

@Module({
  imports: [
    AuditModule,
    EvidenceModule,
    DraftsModule,
    StockModule,
    RequestsModule,
  ],
})
export class AppModule {}
```

---

10) Checklist rápido para o DEV não errar

1. **moves** e **audit_events**: create-only (sem update/delete para cliente)
2. **Audit hashchain**: `tenants/{tenantId}.auditHead.lastHash` atualizado em transação
3. **Drafts**: app cria draft, backend aplica e promove (imutável)
4. **Idempotência** (próximo passo): webhooks/requests offline devem ter `clientRequestId` para não duplicar
5. **SoD mínimo**: aprovador ≠ solicitante (já incluído)

) Guards NestJS prontos (Firebase Auth + Tenant + Roles)

1.1 `FirebaseAuthGuard` (verifica ID token e injeta `req.user`)

```
// src/common/guards/firebase-auth.guard.ts
import { CanActivate, ExecutionContext, Injectable,
UnauthorizedException } from '@nestjs/common';
import * as admin from 'firebase-admin';

@Injectable()
export class FirebaseAuthGuard implements CanActivate {
  async canActivate(ctx: ExecutionContext): Promise<boolean> {
    const req = ctx.switchToHttp().getRequest();
    const auth = req.headers['authorization'] || '';
    const token = auth.startsWith('Bearer ') ? auth.slice(7) : null;

    if (!token) throw new
UnauthorizedException('MISSING_BEARER_TOKEN');

    try {
      const decoded = await admin.auth().verifyIdToken(token, true);
      // decoded: { uid, tenantId, role, permissions, ... }
      req.user = { uid: decoded.uid, token: decoded };
      return true;
    } catch (e) {
```

```
      throw new UnauthorizedException('INVALID_TOKEN');
    }
  }
}
```

1.2 `TenantGuard` (garante que o token tem `tenantId`)

Opcionalmente, você pode exigir que o `tenantId` venha no header `x-tenant-id` e comparar. Eu recomendo **não depender do client**: use sempre o `tenantId` do token como verdade.

```
// src/common/guards/tenant.guard.ts
import { CanActivate, ExecutionContext, Injectable, ForbiddenException
} from '@nestjs/common';

@Injectable()
export class TenantGuard implements CanActivate {
  canActivate(ctx: ExecutionContext): boolean {
    const req = ctx.switchToHttp().getRequest();
    const tenantId = req.user?.token?.tenantId;
    if (!tenantId) throw new
ForbiddenException('TENANT_ID_MISSING_IN_TOKEN');

    // Se quiser validar header opcional:
    const headerTenant = req.headers['x-tenant-id'];
    if (headerTenant && headerTenant !== tenantId) {
      throw new ForbiddenException('TENANT_ID_HEADER_MISMATCH');
    }

    return true;
  }
}
```

1.3 `RolesGuard` + decorator `@Roles()` + `@Permissions()`

```
// src/common/decorators/roles.decorator.ts
import { SetMetadata } from '@nestjs/common';
export const ROLES_KEY = 'roles';
export const Roles = (...roles: string[]) => SetMetadata(ROLES_KEY,
roles);
// src/common/decorators/permissions.decorator.ts
import { SetMetadata } from '@nestjs/common';
export const PERMS_KEY = 'permissions';
export const Permissions = (...perms: string[]) =>
SetMetadata(PERMS_KEY, perms);
// src/common/guards/roles.guard.ts
import { CanActivate, ExecutionContext, Injectable, ForbiddenException
} from '@nestjs/common';
import { Reflector } from '@nestjs/core';
import { ROLES_KEY } from '../decorators/roles.decorator';
import { PERMS_KEY } from '../decorators/permissions.decorator';

@Injectable()
export class RolesGuard implements CanActivate {
  constructor(private reflector: Reflector) {}

  canActivate(ctx: ExecutionContext): boolean {
    const req = ctx.switchToHttp().getRequest();
    const token = req.user?.token || {};
    const role = token.role;
    const perms: string[] = token.permissions || [];
```

```
    const requiredRoles =
this.reflector.getAllAndOverride<string[]>(ROLES_KEY, [
      ctx.getHandler(),
      ctx.getClass(),
    ]);

    const requiredPerms =
this.reflector.getAllAndOverride<string[]>(PERMS_KEY, [
      ctx.getHandler(),
      ctx.getClass(),
    ]);

    if (requiredRoles?.length) {
      if (!role || !requiredRoles.includes(role)) throw new
ForbiddenException('ROLE_NOT_ALLOWED');
    }

    if (requiredPerms?.length) {
      const hasAll = requiredPerms.every(p => perms.includes(p));
      if (!hasAll) throw new
ForbiddenException('MISSING_PERMISSIONS');
    }

    return true;
  }
}
```

Exemplo de uso no controller

```
import { UseGuards, Post } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';

@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
@Roles('MANAGER', 'AUDITOR')
@Permissions('stock.issue')
@Post('/alguma-rota')
async run() { /* ... */ }
```

---

2) Firestore Security Rules completas (MVP Sprints 1–4)

✅ Objetivo:

- Multi-tenant: `request.auth.token.tenantId == resource.data.tenantId`
- Imutabilidade para cliente:
  - `audit_events` e `moves` = **read-only**
- Cliente pode:
  - criar `evidences` (create-only)
  - criar/atualizar `event_drafts` (editável, mas controlado)

- criar `requests` e atualizar somente alguns campos conforme estado (MVP simples)
- ler catálogos/estoque conforme role (você pode endurecer depois)

Observação: **Admin SDK ignora rules**, então backend sempre consegue escrever nos imutáveis.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

    function isAuthed() { return request.auth != null; }
    function tenantId() { return request.auth.token.tenantId; }
    function role() { return request.auth.token.role; }
    function perms() { return request.auth.token.permissions; }

    function hasTenantField(data) { return data.tenantId is string; }
    function tenantMatches(data) { return isAuthed() &&
hasTenantField(data) && data.tenantId == tenantId(); }

    function hasRole(r) { return isAuthed() && role() == r; }
    function hasAnyRole(rs) { return isAuthed() && role() in rs; }
    function hasPerm(p) { return isAuthed() && (perms() is list) && (p
in perms()); }

    function isCreateOnly() { return request.method == 'create'; }

    // =========================
    // Tenants (read-only para cliente)
    // =========================
    match /tenants/{tid} {
      allow read: if isAuthed() && tid == tenantId();
      allow create, update, delete: if false;
    }

    // =========================
    // Catálogo e Itens (MVP: leitura geral no tenant; escrita só por
role)
    // =========================
    match /catalog_items/{id} {
      allow read: if tenantMatches(resource.data);
      allow create: if tenantMatches(request.resource.data) &&
hasAnyRole(['MANAGER','ADMIN']);
      allow update, delete: if false; // recomendado: só servidor (ou
permitir update por ADMIN)
    }

    match /inventory_items/{itemId} {
      allow read: if tenantMatches(resource.data);
      allow create: if tenantMatches(request.resource.data) &&
hasAnyRole(['MANAGER','ADMIN']);
      allow update, delete: if false; // preferir só servidor
    }

    // Lotes e saldos: leitura para cliente, escrita só servidor
    match /inventory_items/{itemId}/lots/{lotId} {
      allow read: if tenantMatches(resource.data);
      allow create, update, delete: if false;
    }
```

```
    match /inventory_items/{itemId}/balances/{locationId} {
      allow read: if tenantMatches(resource.data);
      allow create, update, delete: if false;
    }

    // Moves: imutável para cliente
    match /inventory_items/{itemId}/moves/{moveId} {
      allow read: if tenantMatches(resource.data);
      allow create, update, delete: if false;
    }

    // ========================
    // Evidências: create-only para cliente
    // ========================
    match /evidences/{evidenceId} {
      allow read: if tenantMatches(resource.data);

      // Cliente pode criar evidência do próprio tenant
      allow create: if tenantMatches(request.resource.data)
        && request.resource.data.createdBy.uid == request.auth.uid
        && request.resource.data.sha256 is string
        && request.resource.data.storageRef is string
        && request.resource.data.type in ['PHOTO','AUDIO','DOC']
        && request.resource.data.riskType in ['A','B','DOC','B_LAB'];

      // Atualização de OCR etc. -> somente servidor (Admin SDK)
      allow update, delete: if false;
    }

    // ========================
    // Drafts: cliente cria/atualiza enquanto pendente
    // ========================
    match /event_drafts/{draftId} {
      allow read: if tenantMatches(resource.data);

      allow create: if tenantMatches(request.resource.data)
        && request.resource.data.createdBy.uid == request.auth.uid
        && request.resource.data.status == 'PENDING_VALIDATION'
        && request.resource.data.draftType in
['STOCK_ISSUE','STOCK_MOVE','STOCK_LOSS'];

      // MVP: permitir update apenas se ainda PENDING_VALIDATION e
criado pelo mesmo uid
      allow update: if tenantMatches(resource.data)
        && resource.data.createdBy.uid == request.auth.uid
        && resource.data.status == 'PENDING_VALIDATION'
        && request.resource.data.status == resource.data.status;

      allow delete: if false;
    }

    // ========================
    // Requests: cliente cria; updates limitados (MVP simples)
    // ========================
    match /requests/{requestId} {
      allow read: if tenantMatches(resource.data);

      // Criar request (pendente)
      allow create: if tenantMatches(request.resource.data)
        && request.resource.data.createdBy.uid == request.auth.uid
        && request.resource.data.status == 'pending'
```

```
        && request.resource.data.items is list;

      // Para MVP: bloquear updates do cliente e deixar transições via
backend
      allow update, delete: if false;
    }

    // ========================
    // Audit events: imutável e só leitura para cliente
    // ========================
    match /audit_events/{eventId} {
      allow read: if tenantMatches(resource.data) &&
hasAnyRole(['MANAGER','AUDITOR','ADMIN']);
      allow create, update, delete: if false;
    }
  }
}
```

Ponto importante: Requests eu deixei **update=false** para cliente, garantindo que transições FSM só acontecem pelo backend. É o modo mais seguro.

---

3) Swagger / OpenAPI no NestJS (padrão rápido)

3.1 Configuração no `main.ts`

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { DocumentBuilder, SwaggerModule } from '@nestjs/swagger';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  const config = new DocumentBuilder()
    .setTitle('Ciclo+ API')
    .setDescription('MVP Sprints 1-4: Audit, Evidence, Drafts, Stock,
Requests')
    .setVersion('1.0.0')
    .addBearerAuth(
      { type: 'http', scheme: 'bearer', bearerFormat: 'JWT', name:
'Authorization', in: 'header' },
      'bearerAuth',
    )
    .addApiKey({ type: 'apiKey', name: 'x-tenant-id', in: 'header' },
'tenantHeader') // opcional
    .build();

  const document = SwaggerModule.createDocument(app, config);
  SwaggerModule.setup('/docs', app, document);

  await app.listen(3000);
}
bootstrap();
```

3.2 Decorar controllers com Swagger

EvidenceController (exemplo)

```
import { ApiBearerAuth, ApiTags, ApiOperation, ApiResponse } from
'@nestjs/swagger';

@ApiTags('Evidence')
@ApiBearerAuth('bearerAuth')
@Controller('/api/evidences')
export class EvidenceController {
  @Post()
  @ApiOperation({ summary: 'Create evidence metadata (file already
uploaded to Storage)' })
  @ApiResponse({ status: 201, description: 'Evidence created' })
  async create(...) {}
}
```

DraftsController (exemplo)

```
@ApiTags('Drafts')
@ApiBearerAuth('bearerAuth')
@Controller('/api/drafts')
export class DraftsController {
  @Post()
  @ApiOperation({ summary: 'Create an event draft (offline-friendly)'
})
  async create(...) {}

  @Post(':id/apply')
  @ApiOperation({ summary: 'Apply a draft (promote to immutable move +
audit hashchain)' })
  async apply(...) {}
}
```

RequestsController (exemplo)

```
@ApiTags('Requests')
@ApiBearerAuth('bearerAuth')
@Controller('/api/requests')
export class RequestsController {
  @Post()
  @ApiOperation({ summary: 'Create request (FSM starts at pending)' })
  async create(...) {}

  @Post(':id/approve')
  @ApiOperation({ summary: 'Approve request (SoD enforced)' })
  async approve(...) {}
}
```

---

4) Pequenos ajustes recomendados antes de codar “de verdade”

1. **Idempotência**: inclua `clientRequestId` em `CreateDraftDto` e `CreateEvidenceDto` para impedir duplicatas em offline/retry.
2. **Anti-duplicação forte**: além de `sha256`, considere `sha256 + tenantId` indexado e `create` com “unique check” no backend.
3. **Audit read-only**: eu limitei leitura do audit a roles (MANAGER/AUDITOR/ADMIN).

4. **Draft update restrito**: se quiser mais seguro, permita update só em campos específicos (ex.: `proposed.qty`) — Firestore Rules não é ótimo nisso, então o ideal é **não permitir update** e fazer "novo draft" (mais limpo).

1) Permissions (padrão sugerido)

Use permissions curtas e "por domínio". Ex.:

Core

- `audit.read`
- `evidence.create`
- `evidence.read`
- `draft.create`
- `draft.update` (se vocês realmente quiserem permitir update)
- `draft.apply` (normalmente só backend/manager)

Catálogo / Itens

- `catalog.read`
- `catalog.write`
- `inventory.read`
- `inventory.write`

Estoque

- `stock.read`
- `stock.issue` (baixa/consumo)
- `stock.move` (transferência)
- `stock.loss` (perda)
- `stock.inbound` (entrada via NF)
- `stock.adjust` (ajuste/inventário — normalmente mais restrito)

Requests (FSM)

- `request.create`
- `request.approve`
- `request.fulfill`
- `request.done`
- `request.reject`

(Fase 2/3 depois)

- `contract.create`, `contract.sign`, `escrow.create`, `escrow.release`, `dispute.open`, `dispute.resolve`, `milk.settlement`, etc.

---

2) Matriz RBAC (por Role → Permissions)

ADMIN

Tudo. Inclui gerenciar tenant, roles, políticas, auditoria, etc.

MANAGER

- `catalog.read`, `catalog.write`
- `inventory.read`, `inventory.write`
- `stock.read`, `stock.issue`, `stock.move`, `stock.loss`, `stock.inbound`, `stock.adjust`
- `request.create`, `request.approve`, `request.fulfill`, `request.done`, `request.reject`
- `audit.read`
- `evidence.read`

OPERATOR

Foco operação em campo/loja:

- `catalog.read`, `inventory.read`
- `stock.read`, `stock.issue`, `stock.move`, `stock.loss` *(loss pode exigir evidência B)*
- `request.create`, `request.fulfill`, `request.done`
- `evidence.create`, `evidence.read`
- `draft.create` *(e opcional `draft.update`)*

Não deve: `request.approve`, `stock.inbound`, `stock.adjust`, `audit.read` (ou só o mínimo do que ele próprio fez).

PRODUCER (Produtor Rural)

Foco "dono da produção / estoque próprio / solicitações":

- `catalog.read`, `inventory.read`
- `stock.read` *(do próprio tenant)*
- `request.create`
- `evidence.create`, `evidence.read`
- `draft.create`

Não deve: aprovar, ajustar estoque, ler auditoria completa.

TECHNICIAN

Foco em vistoria/parecer/evidência:

- `catalog.read`, `inventory.read`, `stock.read`
- `evidence.create`, `evidence.read`

- `request.create` *(para solicitar ações)*
- `audit.read` *(limitado: read-only para validação e parecer)*

SUPPLIER (Fornecedor / Indústria)

No MVP (Sprints 1–4) você pode deixar como:

- `catalog.read` *(ou catálogo público do tenant)*
- `evidence.create` *(para anexar NF/laudo/entrega quando virar fase 2)*
- `request.create` *(se vocês usarem requests para "pedido")*
- **sem** acesso a estoque interno do produtor, a menos que seja o mesmo tenant.

Importante: SUPPLIER normalmente é **outro tenant**. No seu modelo atual, tudo é por tenantId; então supplier "externo" não vê dados internos do producer. Isso é bom.

AUDITOR (Jurídico / Compliance)

- `audit.read` (principal)
- `inventory.read`, `stock.read`, `request.read` (leitura)
- `evidence.read`
- **sem escrita** em módulos críticos

LOGISTICS

No MVP:

- `request.read` (somente requests atribuídas / se houver atribuição)
- `evidence.create` (foto/GPS da coleta/entrega — fase 2)
- `evidence.read` (do que ele enviou)

INVESTOR

No MVP:

- `inventory.read` (apenas relatórios agregados; idealmente via endpoints que retornam "views")
- `audit.read` (não recomendado no MVP; melhor "analytics" read-only depois)

---

3) SoD mínima (regras essenciais)

Estas 4 regras evitam fraude já no MVP:

1. **Quem cria Request não aprova**
   - `createdBy.uid != approvedBy.uid`
2. **Ajuste de estoque (`stock.adjust`) não pode ser feito por OPERATOR/PRODUCER**

    - só `MANAGER/ADMIN`
3. **Entrada fiscal (NF) só MANAGER/ADMIN**
    - OPERATOR pode anexar evidência, mas não "criar lote fiscal"
4. **Auditor é read-only**
    - AUDITOR não altera `requests`, `moves`, `balances`, `lots`

---

4) Firestore Rules ajustadas (com roles)

Vou te dar uma versão **mais alinhada** ao RBAC.
Como Firestore Rules têm limitações (não é ótimo para checar arrays complexos), o ideal é:

- **regras enxutas no Firestore**
- **validações finas no backend** com Guards + RulesEngine

Ainda assim dá para aplicar bom controle.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

    function isAuthed() { return request.auth != null; }
    function tenantId() { return request.auth.token.tenantId; }
    function role() { return request.auth.token.role; }
    function perms() { return request.auth.token.permissions; }

    function tenantMatches(data) {
      return isAuthed() && (data.tenantId is string) && data.tenantId
== tenantId();
    }

    function hasAnyRole(rs) { return isAuthed() && (role() in rs); }

    // ===== Tenants =====
    match /tenants/{tid} {
      allow read: if isAuthed() && tid == tenantId() &&
hasAnyRole(['ADMIN','MANAGER','AUDITOR']);
      allow create, update, delete: if false;
    }

    // ===== Catalog =====
    match /catalog_items/{id} {
      allow read: if tenantMatches(resource.data) &&
hasAnyRole(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDI
TOR']);
      allow create: if tenantMatches(request.resource.data) &&
hasAnyRole(['ADMIN','MANAGER']);
      allow update, delete: if false;
    }

    // ===== Inventory Items =====
    match /inventory_items/{itemId} {
```

```
      allow read: if tenantMatches(resource.data) &&
hasAnyRole(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDI
TOR']);
      allow create: if tenantMatches(request.resource.data) &&
hasAnyRole(['ADMIN','MANAGER']);
      allow update, delete: if false;
    }

    // Lots/Balances: read-only for clients
    match /inventory_items/{itemId}/lots/{lotId} {
      allow read: if tenantMatches(resource.data) &&
hasAnyRole(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDI
TOR']);
      allow create, update, delete: if false;
    }
    match /inventory_items/{itemId}/balances/{locationId} {
      allow read: if tenantMatches(resource.data) &&
hasAnyRole(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDI
TOR']);
      allow create, update, delete: if false;
    }

    // Moves: immutable (read-only)
    match /inventory_items/{itemId}/moves/{moveId} {
      allow read: if tenantMatches(resource.data) &&
hasAnyRole(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDI
TOR']);
      allow create, update, delete: if false;
    }

    // ===== Evidences =====
    match /evidences/{evidenceId} {
      allow read: if tenantMatches(resource.data) &&
hasAnyRole(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDI
TOR','LOGISTICS']);

      // create-only for operational roles
      allow create: if tenantMatches(request.resource.data)
        && request.resource.data.createdBy.uid == request.auth.uid
        && request.resource.data.sha256 is string
        && request.resource.data.storageRef is string
        && request.resource.data.type in ['PHOTO','AUDIO','DOC']
        && request.resource.data.riskType in ['A','B','DOC','B_LAB']
        &&
hasAnyRole(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','LOGI
STICS','SUPPLIER']);

      allow update, delete: if false;
    }

    // ===== Drafts =====
    match /event_drafts/{draftId} {
      allow read: if tenantMatches(resource.data) &&
hasAnyRole(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDI
TOR']);

      allow create: if tenantMatches(request.resource.data)
        && request.resource.data.createdBy.uid == request.auth.uid
        && request.resource.data.status == 'PENDING_VALIDATION'
        && request.resource.data.draftType in
['STOCK_ISSUE','STOCK_MOVE','STOCK_LOSS']
```

```
        &&
hasAnyRole(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN']);

      // MVP: permitir update apenas pelo criador, enquanto PENDING
(opcional)
      allow update: if tenantMatches(resource.data)
        && resource.data.createdBy.uid == request.auth.uid
        && resource.data.status == 'PENDING_VALIDATION'
        && request.resource.data.status == 'PENDING_VALIDATION'
        &&
hasAnyRole(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN']);

      allow delete: if false;
    }

    // ===== Requests =====
    match /requests/{requestId} {
      allow read: if tenantMatches(resource.data) &&
hasAnyRole(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDI
TOR']);

      // create pending
      allow create: if tenantMatches(request.resource.data)
        && request.resource.data.createdBy.uid == request.auth.uid
        && request.resource.data.status == 'pending'
        && request.resource.data.items is list
        &&
hasAnyRole(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN']);

      // updates via backend only
      allow update, delete: if false;
    }

    // ===== Audit events =====
    match /audit_events/{eventId} {
      allow read: if tenantMatches(resource.data) &&
hasAnyRole(['ADMIN','MANAGER','AUDITOR']);
      allow create, update, delete: if false;
    }
  }
}
```

---

5) Claims no token (recomendado)

No Firebase Custom Claims para cada usuário:

```
{
  "tenantId": "t_abc",
  "role": "OPERATOR",
  "permissions":
["stock.issue","draft.create","evidence.create","request.create","requ
est.fulfill"]
}
```

Regra prática:

- O backend usa `role` para macro-autorizações

- E `permissions` para granularidade (mais seguro)

---

6) Matriz por endpoint (MVP)

Você pode colar isso no README do projeto:

Evidence

- `POST /api/evidences` → ADMIN/MANAGER/OPERATOR/PRODUCER/TECHNICIAN/LOGISTICS/SUPPLIER (`evidence.create`)
- `GET /api/evidences/:id` → ADMIN/MANAGER/OPERATOR/PRODUCER/TECHNICIAN/AUDITOR/LOGISTICS (`evidence.read`)

Drafts

- `POST /api/drafts` → ADMIN/MANAGER/OPERATOR/PRODUCER/TECHNICIAN (`draft.create`)
- `POST /api/drafts/:id/apply` → **backend only** (mas endpoint protegido): ADMIN/MANAGER (`draft.apply`)
  (OPERATOR só cria draft; o apply pode ser chamado pelo app, mas deve exigir role maior ou virar "queue worker" no backend.)

Stock

- `POST /api/stock/movements` (se existir) → ADMIN/MANAGER (`stock.adjust`/`stock.inbound`) ou OPERATOR (`stock.issue/move/loss`) conforme tipo
- `GET /api/balance/current` → roles operacionais (read)

Requests

- `POST /api/requests` → ADMIN/MANAGER/OPERATOR/PRODUCER/TECHNICIAN (`request.create`)
- `POST /api/requests/:id/approve` → ADMIN/MANAGER (`request.approve`) + SoD enforced
- (depois) `start-fulfillment` → OPERATOR/MANAGER
- (depois) `done` → OPERATOR/MANAGER

A seguir estão **(1) JSON de permissions** (catálogo + descrição), **(2) mapeamento Role → Permissions**, e **(3) seeds** (exemplos de tenants/usuários/claims) prontos para você usar.

---

1) `permissions.json` (catálogo de permissões)

```
{
  "version": "1.0.0",
  "permissions": [
    { "key": "audit.read", "desc": "Ler auditoria imutável e histórico
de eventos" },

    { "key": "evidence.create", "desc": "Criar evidência
(foto/áudio/documento) e anexar metadata" },
    { "key": "evidence.read", "desc": "Ler evidências do tenant" },

    { "key": "draft.create", "desc": "Criar rascunho (draft) de evento
operacional (offline-friendly)" },
    { "key": "draft.update", "desc": "Atualizar draft ainda pendente
(opcional no MVP)" },
    { "key": "draft.apply", "desc": "Aplicar draft (promover para
movimento imutável + auditoria hashchain)" },

    { "key": "catalog.read", "desc": "Ler catálogo de itens" },
    { "key": "catalog.write", "desc": "Criar/alterar itens de catálogo
(normalmente só admin/manager)" },

    { "key": "inventory.read", "desc": "Ler itens operacionais de
inventário (inventory_items)" },
    { "key": "inventory.write", "desc": "Criar itens operacionais de
inventário" },

    { "key": "stock.read", "desc": "Ler saldos, lotes e movimentações
(read-only)" },
    { "key": "stock.inbound", "desc": "Entrada fiscal / criação de
lote de estoque (NF)" },
    { "key": "stock.issue", "desc": "Baixa/consumo/saída de estoque"
},
    { "key": "stock.move", "desc": "Transferência de estoque entre
locais" },
    { "key": "stock.loss", "desc": "Registrar perda/quebra/ajuste por
perda com evidência" },
    { "key": "stock.adjust", "desc": "Ajuste de inventário
(contagem/correção) - restrito" },

    { "key": "request.read", "desc": "Ler solicitações (requests)" },
    { "key": "request.create", "desc": "Criar solicitação (request)"
},
    { "key": "request.approve", "desc": "Aprovar solicitação (SoD
aplica)" },
    { "key": "request.fulfill", "desc": "Iniciar atendimento/execução
(fulfillment)" },
    { "key": "request.done", "desc": "Finalizar solicitação (done)" },
    { "key": "request.reject", "desc": "Rejeitar solicitação" }
  ]
}
```

---

2) `role_permissions.json` (Role → Permissions)

Observação: `draft.update` é opcional; se quiser máxima segurança, remova e force "novo draft" em vez de editar.

```
{
  "version": "1.0.0",
  "roles": {
    "ADMIN": [
      "audit.read",
      "evidence.create", "evidence.read",
      "draft.create", "draft.update", "draft.apply",
      "catalog.read", "catalog.write",
      "inventory.read", "inventory.write",
      "stock.read", "stock.inbound", "stock.issue", "stock.move",
"stock.loss", "stock.adjust",
      "request.read", "request.create", "request.approve",
"request.fulfill", "request.done", "request.reject"
    ],
    "MANAGER": [
      "audit.read",
      "evidence.read",
      "draft.apply",
      "catalog.read", "catalog.write",
      "inventory.read", "inventory.write",
      "stock.read", "stock.inbound", "stock.issue", "stock.move",
"stock.loss", "stock.adjust",
      "request.read", "request.create", "request.approve",
"request.fulfill", "request.done", "request.reject"
    ],
    "OPERATOR": [
      "evidence.create", "evidence.read",
      "draft.create", "draft.update",
      "catalog.read",
      "inventory.read",
      "stock.read", "stock.issue", "stock.move", "stock.loss",
      "request.read", "request.create", "request.fulfill",
"request.done"
    ],
    "PRODUCER": [
      "evidence.create", "evidence.read",
      "draft.create", "draft.update",
      "catalog.read",
      "inventory.read",
      "stock.read", "stock.issue",
      "request.read", "request.create"
    ],
    "TECHNICIAN": [
      "audit.read",
      "evidence.create", "evidence.read",
      "draft.create",
      "catalog.read",
      "inventory.read",
      "stock.read",
      "request.read", "request.create"
    ],
    "SUPPLIER": [
      "evidence.create", "evidence.read",
      "catalog.read",
      "request.read", "request.create"
    ],
    "AUDITOR": [
      "audit.read",
      "evidence.read",
      "catalog.read",
      "inventory.read",
```

```
      "stock.read",
      "request.read"
    ],
    "LOGISTICS": [
      "evidence.create", "evidence.read",
      "request.read"
    ],
    "INVESTOR": [
      "catalog.read",
      "inventory.read",
      "stock.read",
      "request.read"
    ]
  }
}
```

---

3) Seeds (tenants + usuários + claims + exemplo de docs Firestore)

3.1 `seeds/tenants.json`

```
{
  "tenants": [
    {
      "tenantId": "t_demo_agro",
      "name": "Ciclo+ Demo Agro",
      "createdAt": 1730000000000,
      "auditHead": { "lastEventId": null, "lastHash": "GENESIS" }
    }
  ]
}
```

3.2 `seeds/users_claims.json`

Estes são "usuários exemplo". Na prática, o `uid` vem do Firebase Auth. Depois você aplica os custom claims com Admin SDK.

```
{
  "tenantId": "t_demo_agro",
  "users": [
    {
      "uid": "uid_admin_001",
      "displayName": "Admin Ciclo+",
      "email": "admin@demo.local",
      "claims": {
        "tenantId": "t_demo_agro",
        "role": "ADMIN",
        "permissions": [
          "audit.read",
          "evidence.create", "evidence.read",
          "draft.create", "draft.update", "draft.apply",
          "catalog.read", "catalog.write",
          "inventory.read", "inventory.write",
          "stock.read", "stock.inbound", "stock.issue", "stock.move",
"stock.loss", "stock.adjust",
          "request.read", "request.create", "request.approve",
"request.fulfill", "request.done", "request.reject"
        ]
      }
```

```
    },
    {
      "uid": "uid_manager_001",
      "displayName": "Gerente Operacional",
      "email": "manager@demo.local",
      "claims": {
        "tenantId": "t_demo_agro",
        "role": "MANAGER",
        "permissions": [
          "audit.read",
          "evidence.read",
          "draft.apply",
          "catalog.read", "catalog.write",
          "inventory.read", "inventory.write",
          "stock.read", "stock.inbound", "stock.issue", "stock.move",
"stock.loss", "stock.adjust",
          "request.read", "request.create", "request.approve",
"request.fulfill", "request.done", "request.reject"
        ]
      }
    },
    {
      "uid": "uid_operator_001",
      "displayName": "Operador Campo",
      "email": "operator@demo.local",
      "claims": {
        "tenantId": "t_demo_agro",
        "role": "OPERATOR",
        "permissions": [
          "evidence.create", "evidence.read",
          "draft.create", "draft.update",
          "catalog.read",
          "inventory.read",
          "stock.read", "stock.issue", "stock.move", "stock.loss",
          "request.read", "request.create", "request.fulfill",
"request.done"
        ]
      }
    },
    {
      "uid": "uid_producer_001",
      "displayName": "Produtor Rural",
      "email": "producer@demo.local",
      "claims": {
        "tenantId": "t_demo_agro",
        "role": "PRODUCER",
        "permissions": [
          "evidence.create", "evidence.read",
          "draft.create", "draft.update",
          "catalog.read",
          "inventory.read",
          "stock.read", "stock.issue",
          "request.read", "request.create"
        ]
      }
    },
    {
      "uid": "uid_tech_001",
      "displayName": "Técnico Assistência",
      "email": "technician@demo.local",
      "claims": {
```

```
    "tenantId": "t_demo_agro",
    "role": "TECHNICIAN",
    "permissions": [
      "audit.read",
      "evidence.create", "evidence.read",
      "draft.create",
      "catalog.read",
      "inventory.read",
      "stock.read",
      "request.read", "request.create"
    ]
  }
},
{
  "uid": "uid_auditor_001",
  "displayName": "Auditor Compliance",
  "email": "auditor@demo.local",
  "claims": {
    "tenantId": "t_demo_agro",
    "role": "AUDITOR",
    "permissions": [
      "audit.read",
      "evidence.read",
      "catalog.read",
      "inventory.read",
      "stock.read",
      "request.read"
    ]
  }
},
{
  "uid": "uid_logistics_001",
  "displayName": "Transportador",
  "email": "logistics@demo.local",
  "claims": {
    "tenantId": "t_demo_agro",
    "role": "LOGISTICS",
    "permissions": [
      "evidence.create", "evidence.read",
      "request.read"
    ]
  }
},
{
  "uid": "uid_supplier_001",
  "displayName": "Fornecedor/Indústria",
  "email": "supplier@demo.local",
  "claims": {
    "tenantId": "t_demo_agro",
    "role": "SUPPLIER",
    "permissions": [
      "evidence.create", "evidence.read",
      "catalog.read",
      "request.read", "request.create"
    ]
  }
},
{
  "uid": "uid_investor_001",
  "displayName": "Investidor",
  "email": "investor@demo.local",
```

```
      "claims": {
        "tenantId": "t_demo_agro",
        "role": "INVESTOR",
        "permissions": [
          "catalog.read",
          "inventory.read",
          "stock.read",
          "request.read"
        ]
      }
    }
  ]
}
```

3.3 `seeds/firestore_bootstrap.json` (exemplos de docs no Firestore)

```
{
  "catalog_items": [
    {
      "catalogItemId": "cat_salt_25kg",
      "tenantId": "t_demo_agro",
      "name": "Sal mineral 25kg",
      "sku": "SAL25",
      "uom": "KG",
      "identityMode": "LOT",
      "riskClass": "LOW",
      "active": true,
      "createdAt": 1730000000000
    }
  ],
  "inventory_items": [
    {
      "itemId": "inv_salt",
      "tenantId": "t_demo_agro",
      "catalogItemId": "cat_salt_25kg",
      "defaultLocationId": "loc_main",
      "policy": { "issueEvidenceRequired": "A",
"lossEvidenceRequired": "B" },
      "createdAt": 1730000000000
    }
  ],
  "requests": [
    {
      "requestId": "req_demo_1",
      "tenantId": "t_demo_agro",
      "type": "STOCK_REQUEST",
      "status": "pending",
      "items": [{ "itemId": "inv_salt", "qty": 25, "uom": "KG" }],
      "createdBy": { "uid": "uid_operator_001" },
      "createdAt": 1730000000000,
      "updatedAt": 1730000000000
    }
  ]
}
```

---

4) Script exemplo para aplicar custom claims (Node/Admin SDK)

Coloque isso em `scripts/applyClaims.ts` e rode localmente (com credenciais do service account).

```
import * as admin from 'firebase-admin';
import * as fs from 'fs';

admin.initializeApp({
  credential: admin.credential.applicationDefault(),
});

async function main() {
  const data = JSON.parse(fs.readFileSync('seeds/users_claims.json',
'utf8'));
  for (const u of data.users) {
    await admin.auth().setCustomUserClaims(u.uid, u.claims);
    console.log(`Claims set for ${u.uid} (${u.claims.role})`);
  }
}

main().catch(err => {
  console.error(err);
  process.exit(1);
});
```

Recomendações finais (bem objetivas)

- **MVP**: mantenha `draft.apply` só para `ADMIN/MANAGER` (ou só backend worker).
- `AUDITOR` nunca escreve em coleções críticas.
- `SUPPLIER` geralmente é outro tenant; se for "usuário do mesmo tenant", tudo ok.

Se quiser, eu também te mando um **JSON de policies** (RiskMatrix + SoD + FSM) já pronto para subir como `policies/` no Firestore e o backend ler versionado.

Segue um **PoliciesService completo (NestJS)** com:

- cache TTL (60s por padrão)
- carregamento de policies do Firestore (`policies/{policyId}`)
- helpers:
  - `getEvidenceRequirement(eventType, ctx)`
  - `assertEvidenceSatisfies(eventType, ctx, providedEvidence)`
  - `assertSod(eventType, ctx)`
  - `assertRequestTransition(req, targetState)`
  - `assertRequiredFieldsForState(entity, state)`
  - `assertStockRules(eventType, ctx)` (qty>0, saldo não negativo, locais etc.)

E um **RulesEngineService** que orquestra tudo (pré-validação padrão).

---

1) `policies.types.ts` (tipos básicos)

```
// src/policies/policies.types.ts
export type PolicyBase = {
  policyId: string;
  version: string;
  type: 'RISK_MATRIX' | 'SOD' | 'FSM' | 'DOMAIN_RULES' | 'DRAFTS' |
'EVIDENCE' | 'AUDIT';
```

```
  enabled: boolean;
  updatedAt?: number;
};

export type RiskMatrixPolicy = PolicyBase & {
  type: 'RISK_MATRIX';
  matrix: Record<string, any>;
};

export type SodPolicy = PolicyBase & {
  type: 'SOD';
  rules: any[];
};

export type FsmPolicy = PolicyBase & {
  type: 'FSM';
  entityType: string;
  states: string[];
  allowedTransitions: Record<string, string[]>;
  requiredFieldsByState: Record<string, string[]>;
  transitionEvents?: Record<string, string>;
};

export type DomainRulesPolicy = PolicyBase & {
  type: 'DOMAIN_RULES';
  domain: string;
  rules: any[];
};

export type DraftsPolicy = PolicyBase & {
  type: 'DRAFTS';
  rules: any[];
};

export type EvidencePolicy = PolicyBase & {
  type: 'EVIDENCE';
  rules: any[];
};

export type AuditPolicy = PolicyBase & {
  type: 'AUDIT';
  rules: any[];
};

export type AnyPolicy =
  | RiskMatrixPolicy
  | SodPolicy
  | FsmPolicy
  | DomainRulesPolicy
  | DraftsPolicy
  | EvidencePolicy
  | AuditPolicy;

export type PolicyContext = {
  tenantId: string;
  actor: { uid: string; role: string; permissions: string[] };
  payload?: any;
  computed?: any; // ex: balanceAfter
  entity?: any;   // ex: request doc
};
```

---

2) policies.module.ts

```
// src/policies/policies.module.ts
import { Module } from '@nestjs/common';
import { firestoreProvider } from '../common/firestore.provider';
import { PoliciesService } from './policies.service';
import { RulesEngineService } from './rules-engine.service';

@Module({
  providers: [firestoreProvider, PoliciesService, RulesEngineService],
  exports: [PoliciesService, RulesEngineService],
})
export class PoliciesModule {}
```

---

3) policies.service.ts (carrega, cacheia e fornece helpers)

```
// src/policies/policies.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { AnyPolicy, FsmPolicy, PolicyContext, RiskMatrixPolicy,
SodPolicy, DomainRulesPolicy } from './policies.types';

@Injectable()
export class PoliciesService {
  private cache: { loadedAt: number; policies: Record<string,
AnyPolicy> } | null = null;
  private ttlMs = 60_000;

  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  setTtl(ms: number) { this.ttlMs = ms; }

  async loadAll(force = false): Promise<Record<string, AnyPolicy>> {
    const now = Date.now();
    if (!force && this.cache && (now - this.cache.loadedAt) <
this.ttlMs) {
      return this.cache.policies;
    }

    const snap = await this.db.collection('policies').get();
    const policies: Record<string, AnyPolicy> = {};
    snap.forEach(d => {
      const p = d.data() as AnyPolicy;
      if (p?.policyId) policies[p.policyId] = p;
    });

    this.cache = { loadedAt: now, policies };
    return policies;
  }

  async getPolicy<T extends AnyPolicy = AnyPolicy>(policyId: string):
Promise<T | null> {
    const all = await this.loadAll();
    const p = all[policyId] as T | undefined;
    if (!p || p.enabled === false) return null;
    return p;
  }

  // ----------------------------
```

```
  // Risk Matrix helpers
  // ----------------------------

  /**
   * Retorna evidência requerida: 'A' | 'B' | 'DOC' | 'B_LAB'
   */
  async getEvidenceRequirement(eventType: string, ctx: {
itemRiskClass?: string; qty?: number }): Promise<string | null> {
    const p = await
this.getPolicy<RiskMatrixPolicy>('POLICY_RISK_MATRIX_V1');
    if (!p) return null;

    const entry = p.matrix?.[eventType];
    if (!entry) return null;

    let required = entry.default ?? null;

    // regras adicionais (se houver)
    const rules = entry.rules || [];
    for (const r of rules) {
      const when = r.when || {};
      const matchRisk = when.itemRiskClass ? (when.itemRiskClass ===
ctx.itemRiskClass) : true;
      const matchQty = typeof when.qtyGte === 'number' ? ((ctx.qty ??
0) >= when.qtyGte) : true;
      if (matchRisk && matchQty) {
        required = r.require;
      }
    }

    return required;
  }

  async assertEvidenceSatisfies(eventType: string, ctx: {
itemRiskClass?: string; qty?: number }, provided: { riskTypes:
string[] }) {
    const required = await this.getEvidenceRequirement(eventType,
ctx);
    if (!required) return; // sem exigência explícita
    if (!provided?.riskTypes?.includes(required)) {
      throw new Error(`EVIDENCE_REQUIRED_${required}`);
    }
  }

  // ----------------------------
  // SoD helpers
  // ----------------------------

  async assertSod(eventType: string, ctx: PolicyContext) {
    const p = await this.getPolicy<SodPolicy>('POLICY_SOD_MIN_V1');
    if (!p) return;

    for (const rule of p.rules || []) {
      // Rule 1: self approval
      if (rule.id === 'SOD_BLOCK_SELF_APPROVAL' && eventType ===
'REQUEST_APPROVED') {
        const actorUid = ctx.actor?.uid;
        const creatorUid = ctx.entity?.createdBy?.uid;
        if (actorUid && creatorUid && actorUid === creatorUid) {
          throw new Error(rule.onFail?.reason ||
'SOD_SELF_APPROVAL_NOT_ALLOWED');
```

```
        }
      }

      // Rule 2: restrict adjust/inbound handled elsewhere by RBAC,
      // but we enforce again at policy level (defense in depth)
      if (rule.id === 'SOD_RESTRICT_STOCK_ADJUST' && eventType ===
'STOCK_ADJUST_RECORDED') {
        if (!['ADMIN','MANAGER'].includes(ctx.actor?.role)) throw new
Error(rule.onFail?.reason || 'STOCK_ADJUST_RESTRICTED');
      }

      if (rule.id === 'SOD_RESTRICT_STOCK_INBOUND' &&
['INBOUND_NF_IMPORTED','STOCK_LOT_CREATED'].includes(eventType)) {
        if (!['ADMIN','MANAGER'].includes(ctx.actor?.role)) throw new
Error(rule.onFail?.reason || 'STOCK_INBOUND_RESTRICTED');
      }

      if (rule.id === 'SOD_BLOCK_AUDITOR_WRITES') {
        // se AUDITOR tentar executar evento de escrita operacional
        if (ctx.actor?.role === 'AUDITOR') {
          const writeEvents = new Set([

'STOCK_ISSUE_CONFIRMED','STOCK_MOVED','STOCK_LOSS_RECORDED','STOCK_ADJ
UST_RECORDED',

'REQUEST_APPROVED','REQUEST_REJECTED','REQUEST_FULFILLMENT_STARTED','R
EQUEST_DONE'
          ]);
          if (writeEvents.has(eventType)) throw new
Error(rule.onFail?.reason || 'AUDITOR_READ_ONLY');
        }
      }
    }
  }

  // ----------------------------
  // FSM (Requests) helpers
  // ----------------------------

  async getRequestFsm(): Promise<FsmPolicy | null> {
    return this.getPolicy<FsmPolicy>('POLICY_REQUEST_FSM_V1');
  }

  async assertRequestTransition(req: any, targetState: string) {
    const fsm = await this.getRequestFsm();
    if (!fsm) return;

    const from = req?.status;
    if (!from) throw new Error('FSM_MISSING_CURRENT_STATE');

    const allowed = fsm.allowedTransitions?.[from] || [];
    if (!allowed.includes(targetState)) throw new
Error('FSM_INVALID_TRANSITION');
  }

  async assertRequiredFieldsForState(entity: any, state: string) {
    const fsm = await this.getRequestFsm();
    if (!fsm) return;

    const reqFields = fsm.requiredFieldsByState?.[state] || [];
    for (const path of reqFields) {
```

```
      if (!this.hasPath(entity, path)) {
        throw new Error(`FSM_MISSING_FIELD_${path}`);
      }
    }
  }

  private hasPath(obj: any, path: string): boolean {
    const parts = path.split('.');
    let cur = obj;
    for (const p of parts) {
      if (cur == null || !(p in cur)) return false;
      cur = cur[p];
    }
    return cur !== undefined && cur !== null;
  }

  // ----------------------------
  // Stock core rules helpers
  // ----------------------------

  async assertStockRules(eventType: string, ctx: PolicyContext) {
    const p = await
this.getPolicy<DomainRulesPolicy>('POLICY_STOCK_CORE_V1');
    if (!p) return;

    const payload = ctx.payload || {};
    const computed = ctx.computed || {};

    // qty > 0
    if
(['STOCK_ISSUE_CONFIRMED','STOCK_MOVED','STOCK_LOSS_RECORDED','STOCK_A
DJUST_RECORDED'].includes(eventType)) {
      if (!(Number(payload.qty) > 0)) throw new Error('INVALID_QTY');
    }

    // location required
    if
(['STOCK_ISSUE_CONFIRMED','STOCK_MOVED','STOCK_LOSS_RECORDED'].include
s(eventType)) {
      if (!payload.locationId) throw new Error('LOCATION_REQUIRED');
    }

    // toLocation required for move
    if (eventType === 'STOCK_MOVED') {
      if (!payload.toLocationId) throw new
Error('TO_LOCATION_REQUIRED');
    }

    // balanceAfter >= 0 for operations that subtract
    if
(['STOCK_ISSUE_CONFIRMED','STOCK_MOVED','STOCK_LOSS_RECORDED'].include
s(eventType)) {
      if (typeof computed.balanceAfter === 'number' &&
computed.balanceAfter < 0) throw new Error('INSUFFICIENT_STOCK');
    }
  }
}
```

---

4) `rules-engine.service.ts` (orquestrador padrão)

```
// src/policies/rules-engine.service.ts
import { Injectable } from '@nestjs/common';
import { PoliciesService } from './policies.service';
import { PolicyContext } from './policies.types';

@Injectable()
export class RulesEngineService {
  constructor(private readonly policies: PoliciesService) {}

  /**
   * Pré-validação padrão para eventos operacionais.
   * Você chama isso ANTES de executar a transação do domínio.
   */
  async preValidate(eventType: string, ctx: PolicyContext, opts?: {
    itemRiskClass?: string;
    qty?: number;
    providedEvidenceRiskTypes?: string[]; // ex: ['A','B']
    isRequestTransition?: { targetState: string };
  }) {
    // SoD
    await this.policies.assertSod(eventType, ctx);

    // Risk/evidence
    if (opts?.providedEvidenceRiskTypes) {
      await this.policies.assertEvidenceSatisfies(
        eventType,
        { itemRiskClass: opts.itemRiskClass, qty: opts.qty },
        { riskTypes: opts.providedEvidenceRiskTypes }
      );
    }

    // Stock rules (se aplicável)
    await this.policies.assertStockRules(eventType, ctx);

    // Request FSM (se aplicável)
    if (opts?.isRequestTransition?.targetState) {
      await this.policies.assertRequestTransition(ctx.entity,
opts.isRequestTransition.targetState);
    }
  }

  /**
   * Pós-validação: exigir campos obrigatórios após mudança de estado
(ex.: request approved precisa approvedBy/approvedAt)
   */
  async postValidateRequestState(entity: any, state: string) {
    await this.policies.assertRequiredFieldsForState(entity, state);
  }
}
```

---

5) Como integrar isso no fluxo real (exemplo)

5.1 No `RequestsService.approve(...)`

Antes de atualizar:

- `preValidate('REQUEST_APPROVED', ctx, { isRequestTransition: { targetState: 'approved' } })`

Depois de atualizar e antes de commit (ou logo após, se você ler o doc):

- `postValidateRequestState(updatedReq, 'approved')`

Exemplo (trecho):

```
await this.rules.preValidate('REQUEST_APPROVED', {
  tenantId,
  actor,
  entity: req,
  payload: { requestId }
}, { isRequestTransition: { targetState: 'approved' } });
```

5.2 No `DraftsService.applyDraft(...)`

Antes de aplicar:

- determine `itemRiskClass` e `qty`
- busque evidências e monte `providedEvidenceRiskTypes` (ex.: ['B'] se foto)
- `preValidate('STOCK_ISSUE_CONFIRMED', ctx, { itemRiskClass, qty, providedEvidenceRiskTypes })`

---

6) Pequeno “glossário” de como pegar `providedEvidenceRiskTypes`

No MVP, o jeito mais simples:

- ler cada `evidenceId` e coletar `riskType` do doc (`A`, `B`, `DOC`, `B_LAB`)
- montar array único.

---

7) Ajuste no seu AppModule

Inclua o módulo:

```
import { PoliciesModule } from './policies/policies.module';

@Module({
  imports: [
    PoliciesModule,
    // ...outros
  ],
})
export class AppModule {}
```

Se você quiser, eu já monto o patch nos serviços que criamos:

- `RequestsService.approve` chamando `RulesEngineService`
- `DraftsService.applyDraft` chamando `RulesEngineService` e coletando riskTypes das evidências

Basta dizer: **"aplicar RulesEngine nos services"**.

Perfeito — abaixo estão os **patches (prontos para colar)** aplicando o `RulesEngineService` nos serviços que já criamos:

- `RequestsService.approve` (FSM + SoD)
- `DraftsService.applyDraft` (RiskMatrix + evidência + regras de estoque)
- (opcional) `StockService.createMovement` (mesma lógica de estoque)

Vou assumir que você já adicionou o `PoliciesModule` no `AppModule` e exporta o `RulesEngineService`.

---

1) Patch: `RequestsModule` e `RequestsService`

1.1 `requests.module.ts` — importar `PoliciesModule`

```
// src/requests/requests.module.ts
import { Module } from '@nestjs/common';
import { firestoreProvider } from '../common/firestore.provider';
import { AuditModule } from '../audit/audit.module';
import { PoliciesModule } from '../policies/policies.module'; // ✅
add
import { RequestsController } from './requests.controller';
import { RequestsService } from './requests.service';
import { RequestFsmService } from './fsm/request-fsm.service';

@Module({
  imports: [AuditModule, PoliciesModule], // ✅ add
  providers: [firestoreProvider, RequestsService, RequestFsmService],
  controllers: [RequestsController],
})
export class RequestsModule {}
```

1.2 `requests.service.ts` — injetar `RulesEngineService` e aplicar validações

```
// src/requests/requests.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { AuditService } from '../audit/audit.service';
import { makeId } from '../common/utils/id.util';
import { RequestFsmService } from './fsm/request-fsm.service';
import { RulesEngineService } from '../policies/rules-engine.service';
// ✅ add

@Injectable()
export class RequestsService {
```

```
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly audit: AuditService,
    private readonly fsm: RequestFsmService,
    private readonly rules: RulesEngineService, // ✅ add
  ) {}

  // ... create() pode ficar como está (opcional aplicar rules depois)

  async approve(tenantId: string, actor: any, requestId: string, dto:
any) {
    const ref = this.db.collection('requests').doc(requestId);

    await this.db.runTransaction(async (tx) => {
      const snap = await tx.get(ref);
      if (!snap.exists) throw new Error('REQUEST_NOT_FOUND');

      const req = snap.data()!;
      if (req.tenantId !== tenantId) throw new
Error('TENANT_MISMATCH');

      // ✅ PRE-VALIDAÇÃO via RulesEngine (SoD + FSM)
      await this.rules.preValidate(
        'REQUEST_APPROVED',
        { tenantId, actor, entity: req, payload: { requestId, comment:
dto?.comment ?? null } },
        { isRequestTransition: { targetState: 'approved' } }
      );

      // (opcional) mantém seu FSM local (defesa extra)
      this.fsm.assertTransition(req.status, 'approved');

      // Atualiza request
      const updated = {
        status: 'approved',
        approvedBy: { uid: actor.uid },
        approvedAt: Date.now(),
        updatedAt: Date.now(),
      };
      tx.update(ref, updated);

      // ✅ POST-VALIDAÇÃO: campos obrigatórios no estado approved
      await this.rules.postValidateRequestState({ ...req, ...updated
}, 'approved');

      // Auditoria em transação (já estava)
      await this.audit.appendEvent(tenantId, {
        eventId: makeId('evt'),
        eventType: 'REQUEST_APPROVED',
        entityType: 'request',
        entityId: requestId,
        actor: { uid: actor.uid, role: actor.role, permissions:
actor.permissions || [] },
        timestamp: Date.now(),
        evidenceIds: [],
        payload: { requestId, comment: dto?.comment ?? null },
        stateBefore: req.status,
        stateAfter: 'approved',
      }, tx);
    });
```

```
    return { requestId, status: 'approved' };
  }
}
```

Se você também tiver endpoints `reject`, `start-fulfillment`, `done`, é só repetir o padrão mudando:

- `eventType` (ex.: `REQUEST_REJECTED`)
- `targetState` (ex.: `rejected`)
- `postValidateRequestState(..., 'rejected')`

---

2) Patch: `DraftsModule` e `DraftsService`

2.1 `drafts.module.ts` — importar `PoliciesModule`

```
// src/drafts/drafts.module.ts
import { Module } from '@nestjs/common';
import { firestoreProvider } from '../common/firestore.provider';
import { DraftsController } from './drafts.controller';
import { DraftsService } from './drafts.service';
import { AuditModule } from '../audit/audit.module';
import { PoliciesModule } from '../policies/policies.module'; // ✅
add

@Module({
  imports: [AuditModule, PoliciesModule], // ✅ add
  providers: [firestoreProvider, DraftsService],
  controllers: [DraftsController],
})
export class DraftsModule {}
```

2.2 `drafts.service.ts` — coletar riskTypes das evidências + aplicar RiskMatrix/Stock rules

```
// src/drafts/drafts.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';
import { AuditService } from '../audit/audit.service';
import { RulesEngineService } from '../policies/rules-engine.service';
// ✅ add

@Injectable()
export class DraftsService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly audit: AuditService,
    private readonly rules: RulesEngineService, // ✅ add
  ) {}

  // helper: ler evidências e extrair riskTypes
```

```
  private async getEvidenceRiskTypes(tx:
FirebaseFirestore.Transaction, tenantId: string, evidenceIds:
string[]) {
    const riskTypes = new Set<string>();
    for (const id of evidenceIds || []) {
      const evRef = this.db.collection('evidences').doc(id);
      const evSnap = await tx.get(evRef);
      if (!evSnap.exists) continue; // ou erro
      const ev = evSnap.data()!;
      if (ev.tenantId !== tenantId) throw new
Error('TENANT_MISMATCH_EVIDENCE');
      if (ev.riskType) riskTypes.add(ev.riskType);
    }
    return Array.from(riskTypes);
  }

  // helper: obter itemRiskClass a partir do catalog (ou do
inventory_item)
  private async getItemRiskClass(tx: FirebaseFirestore.Transaction,
tenantId: string, itemId: string) {
    const invRef = this.db.collection('inventory_items').doc(itemId);
    const invSnap = await tx.get(invRef);
    if (!invSnap.exists) return 'LOW';

    const inv = invSnap.data()!;
    if (inv.tenantId !== tenantId) throw new
Error('TENANT_MISMATCH_ITEM');

    const catalogItemId = inv.catalogItemId;
    if (!catalogItemId) return 'LOW';

    const catRef =
this.db.collection('catalog_items').doc(catalogItemId);
    const catSnap = await tx.get(catRef);
    if (!catSnap.exists) return 'LOW';

    const cat = catSnap.data()!;
    if (cat.tenantId !== tenantId) throw new
Error('TENANT_MISMATCH_CATALOG');

    return cat.riskClass || 'LOW';
  }

  async applyDraft(tenantId: string, actor: any, draftId: string, dto:
any) {
    const draftRef = this.db.collection('event_drafts').doc(draftId);

    return this.db.runTransaction(async (tx) => {
      const snap = await tx.get(draftRef);
      if (!snap.exists) throw new Error('DRAFT_NOT_FOUND');

      const draft = snap.data()!;
      if (draft.tenantId !== tenantId) throw new
Error('TENANT_MISMATCH');
      if (draft.status !== 'PENDING_VALIDATION') throw new
Error('DRAFT_NOT_APPLICABLE');
      if (draft.draftType !== 'STOCK_ISSUE') throw new
Error('DRAFT_TYPE_NOT_SUPPORTED_MVP');

      const proposed = draft.proposed || {};
      const itemId = proposed.itemId;
```

```
      const locationId = proposed.locationId;
      const qty = Number(proposed.qty);
      const uom = proposed.uom;
      const stockLotId = dto?.overrideStockLotId ??
proposed.stockLotId ?? null;

      if (!itemId || !locationId || !uom || !Number.isFinite(qty))
throw new Error('INVALID_DRAFT_FIELDS');

      // Balance
      const balRef =
this.db.collection('inventory_items').doc(itemId).collection('balances
').doc(locationId);
      const balSnap = await tx.get(balRef);
      const qtyOnHand = balSnap.exists ? (balSnap.data()!.qtyOnHand ||
0) : 0;

      const balanceBefore = qtyOnHand;
      const balanceAfter = qtyOnHand - qty;

      // ✅ coletar contexto para RulesEngine
      const itemRiskClass = await this.getItemRiskClass(tx, tenantId,
itemId);
      const providedEvidenceRiskTypes = await
this.getEvidenceRiskTypes(tx, tenantId, draft.evidenceIds || []);

      // ✅ PRE-VALIDAÇÃO (SoD + RiskMatrix + Stock rules)
      await this.rules.preValidate(
        'STOCK_ISSUE_CONFIRMED',
        {
          tenantId,
          actor,
          payload: { draftId, itemId, stockLotId, qty, uom, locationId
},
          computed: { balanceBefore, balanceAfter },
          entity: draft
        },
        {
          itemRiskClass,
          qty,
          providedEvidenceRiskTypes
        }
      );

      // Move append-only
      const moveId = makeId('mv');
      const moveRef =
this.db.collection('inventory_items').doc(itemId).collection('moves').
doc(moveId);

      const moveDoc = {
        tenantId,
        moveId,
        itemId,
        stockLotId,
        type: 'ISSUE',
        qty,
        uom,
        locationId,
        balanceBefore,
        balanceAfter,
```

```
        performedBy: { uid: actor.uid, role: actor.role },
        performedAt: Date.now(),
        evidenceIds: draft.evidenceIds || [],
        workId: draft.workId ?? null
      };

      tx.create(moveRef, moveDoc);

      // Update balance
      if (balSnap.exists) {
        tx.update(balRef, { qtyOnHand: balanceAfter, updatedAt:
Date.now(), lastMoveId: moveId });
      } else {
        tx.create(balRef, { tenantId, itemId, locationId, qtyOnHand:
balanceAfter, updatedAt: Date.now(), lastMoveId: moveId });
      }

      // Update draft
      tx.update(draftRef, {
        status: 'APPLIED',
        appliedAt: Date.now(),
        moveId,
        reviewerComment: dto?.reviewerComment ?? null
      });

      // Audit event (em transação)
      await this.audit.appendEvent(tenantId, {
        eventId: makeId('evt'),
        eventType: 'STOCK_ISSUE_CONFIRMED',
        entityType: 'inventory_move',
        entityId: moveId,
        actor: { uid: actor.uid, role: actor.role, permissions:
actor.permissions || [] },
        timestamp: Date.now(),
        evidenceIds: draft.evidenceIds || [],
        payload: { draftId, itemId, stockLotId, qty, uom, locationId,
balanceBefore, balanceAfter, itemRiskClass },
        stateBefore: null,
        stateAfter: null
      }, tx);

      return { moveId, balanceAfter };
    });
  }
}
```

✅ Resultado: agora `applyDraft` só executa se:

- evidência exigida pelo RiskMatrix estiver presente (A/B/DOC)
- qty for válida e saldo não ficar negativo
- SoD/role restrições forem violadas → bloqueia

---

3) (Opcional) Patch: `StockService.createMovement` com RulesEngine

Se você ainda usar o endpoint direto `/api/stock/movements`, recomendo aplicar também:

3.1 `stock.module.ts` — importar `PoliciesModule`

```
// src/stock/stock.module.ts
import { Module } from '@nestjs/common';
import { firestoreProvider } from '../common/firestore.provider';
import { StockController } from './stock.controller';
import { StockService } from './stock.service';
import { AuditModule } from '../audit/audit.module';
import { PoliciesModule } from '../policies/policies.module'; // ✅
add

@Module({
  imports: [AuditModule, PoliciesModule], // ✅ add
  providers: [firestoreProvider, StockService],
  controllers: [StockController],
})
export class StockModule {}
```

3.2 `stock.service.ts` — injetar `RulesEngineService` e validar

No topo:

```
import { RulesEngineService } from '../policies/rules-engine.service';
```

No constructor:

```
constructor(
  @Inject(FIRESTORE) private readonly db: FirebaseFirestore.Firestore,
  private readonly audit: AuditService,
  private readonly rules: RulesEngineService, // ✅ add
) {}
```

Antes de criar o move (dentro da transação), compute `balanceAfter` e chame:

```
await this.rules.preValidate(
  `STOCK_${dto.type}_RECORDED`,
  { tenantId, actor, payload: dto, computed: { balanceBefore,
balanceAfter } },
  {
    itemRiskClass: 'LOW', // ou busque igual fizemos no DraftsService
    qty: dto.qty,
    providedEvidenceRiskTypes: (dto.evidenceRiskTypes || []) // se
você passar
  }
);
```

Para o endpoint direto, como você só recebe `evidenceIds`, o ideal é repetir a leitura das evidências (como no drafts) para montar `providedEvidenceRiskTypes`.

---

4) Recomendação prática (bem importante)

No MVP eu recomendo:

- **OPERATOR/PRODUCER/TECHNICIAN**: só `draft.create` (e evidência)
- **MANAGER/ADMIN**: faz `draft.apply` (ou um worker do backend que aplica automaticamente conforme regras)

Isso reduz fraude e simplifica suporte.

Se você quiser, eu também te mando o padrão de **registro "OPERATION_BLOCKED" em auditoria** (com motivo + payload + tentativa de transição) sempre que o RulesEngine levantar erro — fica bem alinhado ao Ciclo+.

Segue um padrão completo para **registrar `OPERATION_BLOCKED` na auditoria** sempre que o `RulesEngine` bloquear uma operação — com motivo, contexto, evidências, tentativa e rastreio. Já deixo:

1. **Modelo do evento OPERATION_BLOCKED**
2. **Helper `withAuditBlock()`** (wrapper) para usar em qualquer service
3. **Filter global NestJS** (opcional) para centralizar
4. **Como aplicar em RequestsService e DraftsService**
5. **Cuidados para não vazar dados sensíveis (LGPD)**

---

1) Modelo do evento `OPERATION_BLOCKED`

Gravar em `audit_events` (hashchain), com payload padronizado:

```
{
  "tenantId": "t_...",
  "eventId": "evt_...",
  "eventType": "OPERATION_BLOCKED",
  "entityType": "request|inventory_move|draft|evidence|stock",
  "entityId": "req_...|mv_...|dr_...",
  "actor": { "uid": "u_...", "role": "OPERATOR", "permissions":
["..."] },
  "timestamp": 1730000000000,
  "evidenceIds": ["ev_..."],
  "payload": {
    "blockedReason": "INSUFFICIENT_STOCK",
    "blockedCode": "INSUFFICIENT_STOCK",
    "message": "human-friendly opcional",
    "attempt": {
      "action": "APPLY_DRAFT|REQUEST_APPROVE|STOCK_MOVE",
      "eventType": "STOCK_ISSUE_CONFIRMED",
      "targetState": "approved",
      "requestId": "req_...",
      "draftId": "dr_...",
      "itemId": "inv_...",
      "qty": 25,
      "locationId": "loc_01"
    },
    "context": {
      "clientMeta": { "deviceId": "x", "offline": true },
```

```
      "ip": "opcional",
      "appVersion": "opcional"
    }
  }
}
```

---

2) Helper `withAuditBlock()` (wrapper) — use dentro dos services

✅ Vantagem: você chama o RulesEngine normalmente; se der erro, ele **audita automaticamente** e re-lança o erro.

```
// src/common/utils/with-audit-block.util.ts
import { makeId } from './id.util';
import { AuditService } from '../../audit/audit.service';

type BlockContext = {
  tenantId: string;
  actor: { uid: string; role: string; permissions: string[] };
  entityType: string;
  entityId?: string | null;
  evidenceIds?: string[];
  attempt: Record<string, any>;
  clientMeta?: Record<string, any>;
};

export async function withAuditBlock<T>(
  audit: AuditService,
  ctx: BlockContext,
  fn: () => Promise<T>,
  tx?: FirebaseFirestore.Transaction
): Promise<T> {
  try {
    return await fn();
  } catch (err: any) {
    const reason = (err?.message || 'UNKNOWN_BLOCK').toString();

    // Se já estivermos numa transação e ela falhou antes do commit,
    // ainda dá para registrar o audit event dentro da mesma tx (se
você passou tx).
    // Se não passou tx, registramos fora.
    await audit.appendEvent(ctx.tenantId, {
      eventId: makeId('evt'),
      eventType: 'OPERATION_BLOCKED',
      entityType: ctx.entityType,
      entityId: ctx.entityId ?? null,
      actor: ctx.actor,
      timestamp: Date.now(),
      evidenceIds: ctx.evidenceIds || [],
      payload: {
        blockedReason: reason,
        blockedCode: reason,
        attempt: ctx.attempt,
        context: { clientMeta: ctx.clientMeta || null }
      },
      stateBefore: null,
      stateAfter: null
    }, tx);
```

```
    throw err;
  }
}
```

Observação: Para auditar **dentro da transação** você precisa passar `tx`.
Se o erro acontece antes de iniciar `runTransaction`, você pode auditar fora — também funciona.

---

3) (Opcional) Filter global NestJS para padronizar erros

Isso não substitui o audit, mas ajuda a padronizar resposta.

```
// src/common/filters/domain-exception.filter.ts
import { ExceptionFilter, Catch, ArgumentsHost, HttpStatus } from
'@nestjs/common';

@Catch(Error)
export class DomainExceptionFilter implements ExceptionFilter {
  catch(exception: Error, host: ArgumentsHost) {
    const ctx = host.switchToHttp();
    const res = ctx.getResponse();

    const msg = exception.message || 'UNKNOWN_ERROR';

    // map simples
    const map: Record<string, number> = {
      INSUFFICIENT_STOCK: HttpStatus.CONFLICT,
      INVALID_QTY: HttpStatus.BAD_REQUEST,
      FSM_INVALID_TRANSITION: HttpStatus.CONFLICT,
      SOD_SELF_APPROVAL_NOT_ALLOWED: HttpStatus.FORBIDDEN,
      EVIDENCE_REQUIRED_A: HttpStatus.BAD_REQUEST,
      EVIDENCE_REQUIRED_B: HttpStatus.BAD_REQUEST,
      EVIDENCE_REQUIRED_DOC: HttpStatus.BAD_REQUEST,
      TENANT_MISMATCH: HttpStatus.FORBIDDEN
    };

    res.status(map[msg] ?? HttpStatus.BAD_REQUEST).json({
      ok: false,
      error: msg
    });
  }
}
```

No `main.ts`:

```
app.useGlobalFilters(new DomainExceptionFilter());
```

---

4) Aplicando nos services (exemplos práticos)

4.1 RequestsService.approve — auditar se falhar

Você já está dentro de `runTransaction`. Aqui a melhor prática é auditar **na mesma tx** quando o bloqueio acontecer antes do commit.

Patch (trecho do approve):

```
import { withAuditBlock } from '../common/utils/with-audit-
block.util';

// dentro de runTransaction(async (tx) => { ... })

await withAuditBlock(
  this.audit,
  {
    tenantId,
    actor: { uid: actor.uid, role: actor.role, permissions:
actor.permissions || [] },
    entityType: 'request',
    entityId: requestId,
    evidenceIds: [],
    attempt: { action: 'REQUEST_APPROVE', eventType:
'REQUEST_APPROVED', requestId, targetState: 'approved' }
  },
  async () => {
    await this.rules.preValidate(
      'REQUEST_APPROVED',
      { tenantId, actor, entity: req, payload: { requestId } },
      { isRequestTransition: { targetState: 'approved' } }
    );
  },
  tx
);
```

Se o RulesEngine bloquear, ele grava `OPERATION_BLOCKED` e depois lança o erro.

4.2 DraftsService.applyDraft — auditar se falhar

Dentro da `runTransaction`, após ler draft/balance/evidences:

```
await withAuditBlock(
  this.audit,
  {
    tenantId,
    actor: { uid: actor.uid, role: actor.role, permissions:
actor.permissions || [] },
    entityType: 'draft',
    entityId: draftId,
    evidenceIds: draft.evidenceIds || [],
    attempt: {
      action: 'APPLY_DRAFT',
      eventType: 'STOCK_ISSUE_CONFIRMED',
      draftId,
      itemId,
      qty,
      locationId
    },
    clientMeta: draft.clientMeta || null
  },
  async () => {
    await this.rules.preValidate(
      'STOCK_ISSUE_CONFIRMED',
```

```
      {
        tenantId,
        actor,
        entity: draft,
        payload: { draftId, itemId, stockLotId, qty, uom, locationId
},
        computed: { balanceBefore, balanceAfter }
      },
      { itemRiskClass, qty, providedEvidenceRiskTypes }
    );
  },
  tx
);
```

---

5) Cuidados LGPD / “não vazar”

No `attempt` do OPERATION_BLOCKED:

- **não** salve foto/áudio “inline” (só `evidenceIds`)
- evite salvar dados pessoais em claro
- preferir IDs e metadados mínimos (itemId, qty, locationId, requestId)

Se precisar salvar algo sensível, crie uma “versão mascarada” do payload.

Resultado prático

A partir daqui, toda tentativa inválida:

- fica registrada em auditoria imutável
- com hashchain
- com motivo rastreável
- e você consegue gerar relatórios de fraude, erros operacionais e conformidade.

Se você quiser, eu também te mando um **endpoint de auditoria**:

- `GET /api/audit/events?entityType=&entityId=&from=&to=`
  com paginação e filtros por role (AUDITOR/MANAGER/ADMIN).

Perfeito — vou fechar o “pacote mínimo completo” para **rodar bem e com segurança** (MVP pronto para produção inicial), sem deixar pontas soltas. Abaixo está tudo o que faltava para “funcionar perfeitamente” dentro do seu padrão Ciclo+.

---

1) Auditoria: endpoint de consulta + paginação + filtros

1.1 DTO de query

```
// src/audit/dto/audit-query.dto.ts
import { IsIn, IsInt, IsOptional, IsString, Max, Min } from 'class-
validator';
```

```
import { Type } from 'class-transformer';

export class AuditQueryDto {
  @IsOptional() @IsString() entityType?: string;
  @IsOptional() @IsString() entityId?: string;
  @IsOptional() @IsString() eventType?: string;

  @IsOptional() @Type(() => Number) @IsInt() @Min(0)
  from?: number; // timestamp ms

  @IsOptional() @Type(() => Number) @IsInt() @Min(0)
  to?: number; // timestamp ms

  @IsOptional() @Type(() => Number) @IsInt() @Min(1) @Max(200)
  limit?: number = 50;

  @IsOptional() @IsString()
  cursor?: string; // base64 json: { "timestamp": 123, "eventId":
"evt_x" }

  @IsOptional() @IsIn(['desc','asc'])
  order?: 'desc' | 'asc' = 'desc';
}
```

1.2 AuditController (somente ADMIN/MANAGER/AUDITOR)

```
// src/audit/audit.controller.ts
import { Controller, Get, Query, UseGuards } from '@nestjs/common';
import { ApiBearerAuth, ApiTags } from '@nestjs/swagger';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { AuditQueryDto } from './dto/audit-query.dto';
import { AuditReadService } from './audit.read.service';

@ApiTags('Audit')
@ApiBearerAuth('bearerAuth')
@Controller('/api/audit')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
@Roles('ADMIN', 'MANAGER', 'AUDITOR')
export class AuditController {
  constructor(private readonly read: AuditReadService) {}

  @Get('/events')
  async list(@Actor() actor: any, @Query() q: AuditQueryDto) {
    return this.read.listEvents(actor.tenantId, q);
  }
}
```

1.3 AuditReadService (query eficiente + cursor)

```
// src/audit/audit.read.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { AuditQueryDto } from './dto/audit-query.dto';

function encodeCursor(obj: any) {
  return Buffer.from(JSON.stringify(obj), 'utf8').toString('base64');
}
function decodeCursor(s: string) {
  return JSON.parse(Buffer.from(s, 'base64').toString('utf8'));
```

```
}

@Injectable()
export class AuditReadService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async listEvents(tenantId: string, q: AuditQueryDto) {
    const limit = q.limit ?? 50;
    const order = q.order ?? 'desc';

    let ref: FirebaseFirestore.Query =
this.db.collection('audit_events')
      .where('tenantId', '==', tenantId);

    if (q.entityType) ref = ref.where('entityType', '==',
q.entityType);
    if (q.entityId) ref = ref.where('entityId', '==', q.entityId);
    if (q.eventType) ref = ref.where('eventType', '==', q.eventType);

    if (typeof q.from === 'number') ref = ref.where('timestamp', '>=',
q.from);
    if (typeof q.to === 'number') ref = ref.where('timestamp', '<=',
q.to);

    // Ordenação + tie-break (eventId) para paginação estável
    ref = ref.orderBy('timestamp', order).orderBy('eventId', order);

    if (q.cursor) {
      const c = decodeCursor(q.cursor);
      ref = ref.startAfter(c.timestamp, c.eventId);
    }

    const snap = await ref.limit(limit).get();
    const items = snap.docs.map(d => d.data());

    const last = snap.docs[snap.docs.length - 1];
    const nextCursor = last ? encodeCursor({ timestamp:
last.get('timestamp'), eventId: last.get('eventId') }) : null;

    return { items, nextCursor };
  }
}
```

1.4 Atualizar AuditModule

```
// src/audit/audit.module.ts
import { Module } from '@nestjs/common';
import { firestoreProvider } from '../common/firestore.provider';
import { AuditService } from './audit.service';
import { AuditReadService } from './audit.read.service';
import { AuditController } from './audit.controller';

@Module({
  providers: [firestoreProvider, AuditService, AuditReadService],
  controllers: [AuditController],
  exports: [AuditService, AuditReadService],
})
export class AuditModule {}
```

---

2) “Operation Blocked” aplicado de forma padrão em TODO service

2.1 Wrapper já pronto (o que te passei) + dica crucial

Use o wrapper **sempre antes de gravar** (pré-validação). Para ficar “perfeito”, faça duas coisas:

1. Se falhar regra → registrar `OPERATION_BLOCKED` **com attempt + evidenceIds + clientMeta**
2. Sempre que possível, registrar **na mesma transação** (`tx`)

Você já tem `withAuditBlock()`. O que falta é **padronizar a resposta do erro** (o Filter global que enviei ajuda).

---

3) Idempotência “de verdade” (evita duplicidade no offline/retry)

Isso é essencial para “funcionar perfeitamente” no campo com rede ruim.

3.1 Adicione `clientRequestId` em Draft e Evidence

- `CreateDraftDto`: `clientRequestId: string`
- `CreateEvidenceDto`: `clientRequestId: string`

3.2 Coleção de idempotência: `idempotency/{tenantId}_{key}`

Quando receber uma requisição que cria algo:

- se existir `idempotencyDoc` → retorna o resultado salvo
- se não existir → cria idempotencyDoc **na mesma transação** do efeito

Exemplo (padrão em service):

```
const idemKey = `${tenantId}:${actor.uid}:${dto.clientRequestId}`;
const idemRef = db.collection('idempotency').doc(idemKey);

const idemSnap = await tx.get(idemRef);
if (idemSnap.exists) return idemSnap.data().result;

...executa operação...
tx.create(idemRef, { tenantId, key: idemKey, createdAt: Date.now(),
result: { moveId, balanceAfter } });
```

---

4) Índices Firestore necessários (para não quebrar em produção)

Sem isso, suas queries vão falhar/ficar lentas.

4.1 Audit events

- `tenantId + timestamp(desc) + eventId(desc)`
- se usar filtros:
  - `tenantId + entityType + timestamp + eventId`
  - `tenantId + entityType + entityId + timestamp + eventId`
  - `tenantId + eventType + timestamp + eventId`

4.2 Balances / moves

- Se consultar moves por item:
  - `tenantId + performedAt(desc)` (dentro de subcoleção por item pode bastar orderBy)
- Requests:
  - `tenantId + status + createdAt(desc)`

Dica: quando o Firestore reclamar do índice, copie o link que ele dá e crie.

---

5) Segurança: "quem pode aplicar draft"

Para ficar perfeito e antifraude, recomendo:

- **OPERATOR/PRODUCER/TECHNICIAN**: só cria draft + evidência
- **MANAGER/ADMIN**: aplica draft
  (ou um **worker** do backend que aplica automaticamente quando regras permitem)

Implementação simples

No `DraftsController.apply`, adicione:

```
@Roles('ADMIN','MANAGER')
```

ou

```
@Permissions('draft.apply')
```

---

6) Worker automático de aplicação (opcional, mas deixa perfeito)

Se você quiser "sem intervenção", faça um job que:

- busca drafts `PENDING_VALIDATION`
- tenta aplicar
- se falhar, grava `OPERATION_BLOCKED` e marca draft como `PENDING_REVIEW`

Coleção draft pode ter:

- `status: PENDING_VALIDATION | PENDING_REVIEW | APPLIED | REJECTED`
- `lastError`

---

7) Logging, métricas e rate limit (para ficar estável)

Para "funcionar perfeitamente" você precisa blindar a API:

- **Rate limit** por IP/uid (ex.: `@nestjs/throttler`)
- Log estruturado (pino/winston):
  - requestId
  - tenantId
  - uid
  - endpoint
  - errorCode
- Métricas (opcional): contagem de `OPERATION_BLOCKED` por razão

---

8) Ajustes finais nas Rules do Firestore (produção)

Para consistência total:

- `requests`: **cliente não atualiza** (já deixamos assim)
- `audit_events` e `moves`: **somente read** para cliente
- `evidences`: create-only, update false (OCR só servidor)
- `event_drafts`: update opcional (se quiser ainda mais seguro, torne create-only e "novo draft")

---

9) Checklist final de "pronto para rodar"

✅ Auth + Claims
✅ Guards Tenant/Roles/Permissions
✅ Firestore Rules multi-tenant + imutáveis protegidos
✅ Audit hashchain + endpoint /audit/events com cursor
✅ RulesEngine aplicado nos services
✅ OPERATION_BLOCKED auditado sempre
✅ Idempotência (clientRequestId)
✅ Índices Firestore criados
✅ Rate limit + logs

Se você quiser, eu também posso te entregar agora (em texto) os 3 endpoints que faltam do FSM:

- `POST /api/requests/:id/reject`
- `POST /api/requests/:id/start-fulfillment`
- `POST /api/requests/:id/done` (registrando `moveIds`)

Quer que eu gere esses 3 já no mesmo padrão (RulesEngine + OPERATION_BLOCKED + audit hashchain)?

A seguir estão os **3 endpoints faltantes do FSM** já no padrão "Ciclo+ perfeito":

- `reject`
- `start-fulfillment`
- `done`

Com:

- `RulesEngine.preValidate(...)` (FSM + SoD)
- `withAuditBlock(...)` (gera `OPERATION_BLOCKED` se bloquear)
- `audit.appendEvent(...)` na **mesma transação**
- `postValidateRequestState(...)` garantindo campos obrigatórios do estado

Observação: vou manter **update do request somente pelo backend** (Firestore Rules já bloqueiam update pelo cliente).
Para roles, recomendo:

- reject: `ADMIN/MANAGER` (ou `request.reject`)
- start-fulfillment: `ADMIN/MANAGER/OPERATOR`
- done: `ADMIN/MANAGER/OPERATOR`

---

1) DTOs adicionais (Requests)

```
// src/requests/dto/requests-actions.dto.ts
import { IsArray, IsNotEmpty, IsOptional, IsString } from 'class-
validator';

export class RejectRequestDto {
  @IsString() @IsNotEmpty() reason: string;
}

export class StartFulfillmentDto {
  @IsOptional() @IsString() note?: string;
}

export class DoneRequestDto {
  @IsArray() moveIds: string[]; // movimentos (mv_...) usados para
atender
  @IsOptional() @IsString() note?: string;
}
```

---

2) Controller: novas rotas

```
// src/requests/requests.controller.ts
import { Body, Controller, Param, Post, UseGuards } from
'@nestjs/common';
import { ApiBearerAuth, ApiTags, ApiOperation } from
'@nestjs/swagger';
import { RequestsService } from './requests.service';
import { Actor } from '../common/decorators/actor.decorator';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { RejectRequestDto, StartFulfillmentDto, DoneRequestDto } from
'./dto/requests-actions.dto';

@ApiTags('Requests')
@ApiBearerAuth('bearerAuth')
@Controller('/api/requests')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class RequestsController {
  constructor(private readonly requests: RequestsService) {}

  // ... create/approve já existentes

  @Post(':id/reject')
  @ApiOperation({ summary: 'Reject request (FSM enforced + audit)' })
  @Roles('ADMIN','MANAGER') // ou @Permissions('request.reject')
  @Permissions('request.reject')
  async reject(@Actor() actor: any, @Param('id') id: string, @Body()
dto: RejectRequestDto) {
    return this.requests.reject(actor.tenantId, actor, id, dto);
  }

  @Post(':id/start-fulfillment')
  @ApiOperation({ summary: 'Start fulfillment (FSM enforced + audit)'
})
  @Roles('ADMIN','MANAGER','OPERATOR')
  @Permissions('request.fulfill')
  async startFulfillment(@Actor() actor: any, @Param('id') id: string,
@Body() dto: StartFulfillmentDto) {
    return this.requests.startFulfillment(actor.tenantId, actor, id,
dto);
  }

  @Post(':id/done')
  @ApiOperation({ summary: 'Mark request done (requires moveIds) (FSM
enforced + audit)' })
  @Roles('ADMIN','MANAGER','OPERATOR')
  @Permissions('request.done')
  async done(@Actor() actor: any, @Param('id') id: string, @Body()
dto: DoneRequestDto) {
    return this.requests.done(actor.tenantId, actor, id, dto);
  }
}
```

---

3) Service: implementações completas (com OPERATION_BLOCKED)

```
// src/requests/requests.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { AuditService } from '../audit/audit.service';
import { makeId } from '../common/utils/id.util';
import { RequestFsmService } from './fsm/request-fsm.service';
import { RulesEngineService } from '../policies/rules-engine.service';
import { withAuditBlock } from '../common/utils/with-audit-
block.util';

@Injectable()
export class RequestsService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly audit: AuditService,
    private readonly fsm: RequestFsmService,
    private readonly rules: RulesEngineService,
  ) {}

  async reject(tenantId: string, actor: any, requestId: string, dto: {
reason: string }) {
    const ref = this.db.collection('requests').doc(requestId);

    await this.db.runTransaction(async (tx) => {
      const snap = await tx.get(ref);
      if (!snap.exists) throw new Error('REQUEST_NOT_FOUND');

      const req = snap.data()!;
      if (req.tenantId !== tenantId) throw new
Error('TENANT_MISMATCH');

      // Permitir rejeitar de pending ou approved (como na policy)
      const targetState = 'rejected';

      await withAuditBlock(
        this.audit,
        {
          tenantId,
          actor: { uid: actor.uid, role: actor.role, permissions:
actor.permissions || [] },
          entityType: 'request',
          entityId: requestId,
          evidenceIds: [],
          attempt: { action: 'REQUEST_REJECT', eventType:
'REQUEST_REJECTED', requestId, targetState, reason: dto.reason }
        },
        async () => {
          await this.rules.preValidate(
            'REQUEST_REJECTED',
            { tenantId, actor, entity: req, payload: { requestId,
reason: dto.reason } },
            { isRequestTransition: { targetState } }
          );
        },
        tx
      );
```

```
      // Defesa extra
      this.fsm.assertTransition(req.status, 'rejected');

      const updated = {
        status: 'rejected',
        rejectedBy: { uid: actor.uid },
        rejectedAt: Date.now(),
        reason: dto.reason,
        updatedAt: Date.now(),
      };

      tx.update(ref, updated);

      await this.rules.postValidateRequestState({ ...req, ...updated
}, 'rejected');

      await this.audit.appendEvent(tenantId, {
        eventId: makeId('evt'),
        eventType: 'REQUEST_REJECTED',
        entityType: 'request',
        entityId: requestId,
        actor: { uid: actor.uid, role: actor.role, permissions:
actor.permissions || [] },
        timestamp: Date.now(),
        evidenceIds: [],
        payload: { requestId, reason: dto.reason },
        stateBefore: req.status,
        stateAfter: 'rejected',
      }, tx);
    });

    return { requestId, status: 'rejected' };
  }

  async startFulfillment(tenantId: string, actor: any, requestId:
string, dto: { note?: string }) {
    const ref = this.db.collection('requests').doc(requestId);

    await this.db.runTransaction(async (tx) => {
      const snap = await tx.get(ref);
      if (!snap.exists) throw new Error('REQUEST_NOT_FOUND');

      const req = snap.data()!;
      if (req.tenantId !== tenantId) throw new
Error('TENANT_MISMATCH');

      const targetState = 'fulfillment';

      await withAuditBlock(
        this.audit,
        {
          tenantId,
          actor: { uid: actor.uid, role: actor.role, permissions:
actor.permissions || [] },
          entityType: 'request',
          entityId: requestId,
          evidenceIds: [],
          attempt: { action: 'REQUEST_START_FULFILLMENT', eventType:
'REQUEST_FULFILLMENT_STARTED', requestId, targetState, note: dto.note
?? null }
        },
```

```
        async () => {
          await this.rules.preValidate(
            'REQUEST_FULFILLMENT_STARTED',
            { tenantId, actor, entity: req, payload: { requestId,
note: dto.note ?? null } },
            { isRequestTransition: { targetState } }
          );
        },
        tx
      );

      this.fsm.assertTransition(req.status, 'fulfillment');

      const updated = {
        status: 'fulfillment',
        fulfillmentStartedBy: { uid: actor.uid },
        fulfillmentStartedAt: Date.now(),
        fulfillmentNote: dto.note ?? null,
        updatedAt: Date.now(),
      };

      tx.update(ref, updated);

      await this.rules.postValidateRequestState({ ...req, ...updated
}, 'fulfillment');

      await this.audit.appendEvent(tenantId, {
        eventId: makeId('evt'),
        eventType: 'REQUEST_FULFILLMENT_STARTED',
        entityType: 'request',
        entityId: requestId,
        actor: { uid: actor.uid, role: actor.role, permissions:
actor.permissions || [] },
        timestamp: Date.now(),
        evidenceIds: [],
        payload: { requestId, note: dto.note ?? null },
        stateBefore: req.status,
        stateAfter: 'fulfillment',
      }, tx);
    });

    return { requestId, status: 'fulfillment' };
  }

  async done(tenantId: string, actor: any, requestId: string, dto: {
moveIds: string[]; note?: string }) {
    const ref = this.db.collection('requests').doc(requestId);

    await this.db.runTransaction(async (tx) => {
      const snap = await tx.get(ref);
      if (!snap.exists) throw new Error('REQUEST_NOT_FOUND');

      const req = snap.data()!;
      if (req.tenantId !== tenantId) throw new
Error('TENANT_MISMATCH');

      const targetState = 'done';
      const moveIds = dto.moveIds || [];

      await withAuditBlock(
        this.audit,
```

```
        {
          tenantId,
          actor: { uid: actor.uid, role: actor.role, permissions:
actor.permissions || [] },
          entityType: 'request',
          entityId: requestId,
          evidenceIds: [],
          attempt: { action: 'REQUEST_DONE', eventType:
'REQUEST_DONE', requestId, targetState, moveIds, note: dto.note ??
null }
        },
        async () => {
          await this.rules.preValidate(
            'REQUEST_DONE',
            { tenantId, actor, entity: req, payload: { requestId,
moveIds, note: dto.note ?? null } },
            { isRequestTransition: { targetState } }
          );

          if (!Array.isArray(moveIds) || moveIds.length === 0) throw
new Error('MOVE_IDS_REQUIRED');
        },
        tx
      );

      this.fsm.assertTransition(req.status, 'done');

      const updated = {
        status: 'done',
        doneBy: { uid: actor.uid },
        doneAt: Date.now(),
        doneNote: dto.note ?? null,
        moveIds,
        updatedAt: Date.now(),
      };

      tx.update(ref, updated);

      await this.rules.postValidateRequestState({ ...req, ...updated
}, 'done');

      await this.audit.appendEvent(tenantId, {
        eventId: makeId('evt'),
        eventType: 'REQUEST_DONE',
        entityType: 'request',
        entityId: requestId,
        actor: { uid: actor.uid, role: actor.role, permissions:
actor.permissions || [] },
        timestamp: Date.now(),
        evidenceIds: [],
        payload: { requestId, moveIds, note: dto.note ?? null },
        stateBefore: req.status,
        stateAfter: 'done',
      }, tx);
    });

    return { requestId, status: 'done' };
  }
}
```

4) Pontos que deixam “perfeito” mesmo (recomendados)

Para ficar 100% alinhado ao agro + operação:

1. **done deve verificar se moveIds pertencem ao mesmo tenant**
   - opcional: checar se os moves realmente existem e foram criados após `fulfillmentStartedAt`.
2. **fulfillment pode “reservar” estoque** (fase 2)
   - no MVP, não reserva: apenas controla fluxo; estoque é baixado via drafts/moves.
3. **OPERATION_BLOCKED** já fica gravado para qualquer tentativa inválida.

---

5) Ajuste de permissions (confirmação)

Garanta que:

- MANAGER/ADMIN tenham `request.reject`, `request.fulfill`, `request.done`
- OPERATOR tenha `request.fulfill`, `request.done`
- PRODUCER não tenha `request.approve/reject/done` (dependendo do seu fluxo)

Se você quiser, eu fecho também o **endpoint `GET /api/requests` com paginação e filtros por status** (muito necessário no app) e um `GET /api/requests/:id` completo.

A seguir estão os endpoints **necessários para o app funcionar bem**:

- `GET /api/requests` (lista com filtros + paginação por cursor)
- `GET /api/requests/:id` (detalhe)
- (extra recomendado) `GET /api/requests/:id/timeline` (opcional: puxa eventos de auditoria ligados ao request)

Tudo com:

- multi-tenant (via token)
- RBAC (quem pode ler request)
- paginação estável (createdAt + requestId)
- pronto para React Query / infinite scroll

---

1) DTO Query para listagem

```
// src/requests/dto/requests-query.dto.ts
import { IsIn, IsInt, IsOptional, IsString, Max, Min } from 'class-
validator';
```

```
import { Type } from 'class-transformer';

export class RequestsQueryDto {
  @IsOptional() @IsString()
  status?: 'pending' | 'approved' | 'fulfillment' | 'done' |
'rejected';

  @IsOptional() @Type(() => Number) @IsInt() @Min(0)
  from?: number; // createdAt >= from

  @IsOptional() @Type(() => Number) @IsInt() @Min(0)
  to?: number; // createdAt <= to

  @IsOptional() @Type(() => Number) @IsInt() @Min(1) @Max(200)
  limit?: number = 50;

  @IsOptional() @IsIn(['desc','asc'])
  order?: 'desc' | 'asc' = 'desc';

  @IsOptional() @IsString()
  cursor?: string; // base64 json: { createdAt, requestId }
}
```

---

2) RequestsReadService (lista + cursor + detalhe)

```
// src/requests/requests.read.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { RequestsQueryDto } from './dto/requests-query.dto';

function encodeCursor(obj: any) {
  return Buffer.from(JSON.stringify(obj), 'utf8').toString('base64');
}
function decodeCursor(s: string) {
  return JSON.parse(Buffer.from(s, 'base64').toString('utf8'));
}

@Injectable()
export class RequestsReadService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async list(tenantId: string, q: RequestsQueryDto) {
    const limit = q.limit ?? 50;
    const order = q.order ?? 'desc';

    let ref: FirebaseFirestore.Query = this.db.collection('requests')
      .where('tenantId', '==', tenantId);

    if (q.status) ref = ref.where('status', '==', q.status);
    if (typeof q.from === 'number') ref = ref.where('createdAt', '>=',
q.from);
    if (typeof q.to === 'number') ref = ref.where('createdAt', '<=',
q.to);

    ref = ref.orderBy('createdAt', order).orderBy('requestId', order);

    if (q.cursor) {
      const c = decodeCursor(q.cursor);
      ref = ref.startAfter(c.createdAt, c.requestId);
```

```
    }

    const snap = await ref.limit(limit).get();
    const items = snap.docs.map(d => d.data());

    const last = snap.docs[snap.docs.length - 1];
    const nextCursor = last ? encodeCursor({ createdAt:
last.get('createdAt'), requestId: last.get('requestId') }) : null;

    return { items, nextCursor };
  }

  async getById(tenantId: string, requestId: string) {
    const ref = this.db.collection('requests').doc(requestId);
    const snap = await ref.get();
    if (!snap.exists) throw new Error('REQUEST_NOT_FOUND');
    const data = snap.data()!;
    if (data.tenantId !== tenantId) throw new
Error('TENANT_MISMATCH');
    return data;
  }
}
```

---

3) Controller: `GET /api/requests` e `GET /api/requests/:id`

```
// src/requests/requests.controller.ts (adicionar)
import { Get, Query } from '@nestjs/common';
import { RequestsReadService } from './requests.read.service';
import { RequestsQueryDto } from './dto/requests-query.dto';
import { ApiOperation } from '@nestjs/swagger';

// no constructor do controller:
constructor(
  private readonly requests: RequestsService,
  private readonly read: RequestsReadService, // ✅ add
) {}

// add endpoints:

@Get()
@ApiOperation({ summary: 'List requests (cursor pagination)' })
@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDITOR')
@Permissions('request.read')
async list(@Actor() actor: any, @Query() q: RequestsQueryDto) {
  return this.read.list(actor.tenantId, q);
}

@Get(':id')
@ApiOperation({ summary: 'Get request details' })
@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDITOR')
@Permissions('request.read')
async get(@Actor() actor: any, @Param('id') id: string) {
  return this.read.getById(actor.tenantId, id);
}
```

Se você preferir não exigir `Permissions('request.read')` para todos, pode manter apenas roles. Mas permission é mais preciso.

---

4) RequestsModule: registrar o ReadService

```
// src/requests/requests.module.ts
import { RequestsReadService } from './requests.read.service';

@Module({
  imports: [AuditModule, PoliciesModule],
  providers: [firestoreProvider, RequestsService, RequestsReadService,
RequestFsmService],
  controllers: [RequestsController],
})
export class RequestsModule {}
```

---

5) (Extra recomendado) Timeline do request via auditoria

Isso é MUITO útil no app para "ver o que aconteceu".

5.1 Endpoint

```
GET /api/requests/:id/timeline
```

Implementação (simples)

```
// src/requests/requests.controller.ts (adicionar)
@Get(':id/timeline')
@ApiOperation({ summary: 'Request timeline (audit events filtered)' })
@Roles('ADMIN','MANAGER','AUDITOR')
@Permissions('audit.read')
async timeline(@Actor() actor: any, @Param('id') id: string) {
  // reusa AuditReadService via DI (se quiser)
  // aqui eu faço query direta por simplicidade:
  // melhor: injetar AuditReadService e chamar listEvents com filtro
entityType/entityId

  // ...vou deixar a versão "injeção" logo abaixo...
  return { ok: true };
}
```

Melhor (injeção do AuditReadService)

No controller, injete `AuditReadService` e use:

```
import { AuditReadService } from '../audit/audit.read.service';

constructor(
  private readonly requests: RequestsService,
  private readonly read: RequestsReadService,
  private readonly auditRead: AuditReadService
) {}

@Get(':id/timeline')
@Roles('ADMIN','MANAGER','AUDITOR')
@Permissions('audit.read')
async timeline(@Actor() actor: any, @Param('id') id: string, @Query()
q: any) {
  return this.auditRead.listEvents(actor.tenantId, {
```

```
    entityType: 'request',
    entityId: id,
    limit: q.limit ? Number(q.limit) : 100,
    order: 'asc'
  } as any);
}
```

---

6) Índices Firestore para Requests (não esquecer)

Para `GET /requests` com filtros:

- `tenantId + createdAt(desc) + requestId(desc)`
- `tenantId + status + createdAt(desc) + requestId(desc)`

---

7) Ajuste de permissions para leitura de requests

Garanta que os roles que vão listar requests tenham `request.read`:

- ADMIN ✅
- MANAGER ✅
- OPERATOR ✅
- PRODUCER ✅
- TECHNICIAN ✅
- AUDITOR ✅
- SUPPLIER/LOGISTICS/INVESTOR: depende do seu fluxo (eu deixaria fora no MVP)

Se você quiser, eu também monto o **módulo de "Locations"** (locais/galpões/fazendas) + endpoint `GET /api/stock/balance/current?itemId=&locationId=` e `GET /api/stock/moves` paginado — isso normalmente é o que fecha o app de estoque.

A seguir eu fecho o que falta para o app de estoque ficar **completo e usável**:

1. **LocationsModule** (locais: fazenda/galpão/depósito/loja)
2. **StockRead endpoints**:
   - `GET /api/stock/balance/current`
   - `GET /api/stock/moves`
3. **Índices Firestore** necessários
4. **Regras de segurança recomendadas** (ajuste mínimo nas rules)

Mantendo a regra de ouro: `balances` e `moves` são **read-only para o cliente** (somente backend escreve).

---

1) LocationsModule (CRUD mínimo)

1.1 DTO

```
// src/locations/dto/locations.dto.ts
import { IsNotEmpty, IsOptional, IsString } from 'class-validator';

export class CreateLocationDto {
  @IsString() @IsNotEmpty() name: string;
  @IsOptional() @IsString() type?: 'FARM' | 'WAREHOUSE' | 'STORE' |
'YARD' | 'SILO';
  @IsOptional() @IsString() carCode?: string; // vínculo CAR
(opcional)
  @IsOptional() @IsString() stateRegistration?: string; // inscrição
estadual (opcional)
  @IsOptional() geofence?: { lat: number; lng: number; radiusM: number
};
}
```

1.2 Service

```
// src/locations/locations.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class LocationsService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async create(tenantId: string, actor: any, dto: any) {
    const locationId = makeId('loc');
    const ref = this.db.collection('locations').doc(locationId);

    const doc = {
      tenantId,
      locationId,
      name: dto.name,
      type: dto.type ?? 'WAREHOUSE',
      carCode: dto.carCode ?? null,
      stateRegistration: dto.stateRegistration ?? null,
      geofence: dto.geofence ?? null,
      active: true,
      createdBy: { uid: actor.uid },
      createdAt: Date.now(),
      updatedAt: Date.now(),
    };

    await ref.create(doc);
    return doc;
  }

  async list(tenantId: string) {
    const snap = await this.db.collection('locations')
      .where('tenantId', '==', tenantId)
      .where('active', '==', true)
      .orderBy('name', 'asc')
      .get();

    return snap.docs.map(d => d.data());
  }
```

```
  async get(tenantId: string, locationId: string) {
    const snap = await
this.db.collection('locations').doc(locationId).get();
    if (!snap.exists) throw new Error('LOCATION_NOT_FOUND');
    const data = snap.data()!;
    if (data.tenantId !== tenantId) throw new
Error('TENANT_MISMATCH');
    return data;
  }
}
```

1.3 Controller

```
// src/locations/locations.controller.ts
import { Body, Controller, Get, Param, Post, UseGuards } from
'@nestjs/common';
import { ApiBearerAuth, ApiTags, ApiOperation } from
'@nestjs/swagger';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { LocationsService } from './locations.service';
import { CreateLocationDto } from './dto/locations.dto';

@ApiTags('Locations')
@ApiBearerAuth('bearerAuth')
@Controller('/api/locations')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class LocationsController {
  constructor(private readonly locations: LocationsService) {}

  @Post()
  @ApiOperation({ summary: 'Create location (farm/warehouse/store)' })
  @Roles('ADMIN','MANAGER')
  @Permissions('inventory.write')
  async create(@Actor() actor: any, @Body() dto: CreateLocationDto) {
    return this.locations.create(actor.tenantId, actor, dto);
  }

  @Get()
  @ApiOperation({ summary: 'List active locations' })

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDITOR')
  @Permissions('inventory.read')
  async list(@Actor() actor: any) {
    return this.locations.list(actor.tenantId);
  }

  @Get(':id')
  @ApiOperation({ summary: 'Get location details' })

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDITOR')
  @Permissions('inventory.read')
  async get(@Actor() actor: any, @Param('id') id: string) {
    return this.locations.get(actor.tenantId, id);
  }
}
```

1.4 Module

```
// src/locations/locations.module.ts
import { Module } from '@nestjs/common';
import { firestoreProvider } from '../common/firestore.provider';
import { LocationsController } from './locations.controller';
import { LocationsService } from './locations.service';

@Module({
  providers: [firestoreProvider, LocationsService],
  controllers: [LocationsController],
  exports: [LocationsService],
})
export class LocationsModule {}
```

Adicione `LocationsModule` no `AppModule`.

---

2) Stock Read: saldo atual e histórico de movimentos

2.1 DTO Query — balance

```
// src/stock/dto/balance-query.dto.ts
import { IsOptional, IsString } from 'class-validator';

export class BalanceQueryDto {
  @IsOptional() @IsString() itemId?: string;
  @IsOptional() @IsString() locationId?: string;
}
```

2.2 DTO Query — moves (paginado)

```
// src/stock/dto/moves-query.dto.ts
import { IsIn, IsInt, IsOptional, IsString, Max, Min } from 'class-
validator';
import { Type } from 'class-transformer';

export class MovesQueryDto {
  @IsString() itemId: string;
  @IsOptional() @IsString() locationId?: string;

  @IsOptional() @Type(() => Number) @IsInt() @Min(1) @Max(200)
  limit?: number = 50;

  @IsOptional() @IsIn(['desc','asc'])
  order?: 'desc' | 'asc' = 'desc';

  @IsOptional() @IsString()
  cursor?: string; // base64 json: { performedAt, moveId }
}
```

2.3 StockReadService

```
// src/stock/stock.read.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { BalanceQueryDto } from './dto/balance-query.dto';
import { MovesQueryDto } from './dto/moves-query.dto';

function encodeCursor(obj: any) {
  return Buffer.from(JSON.stringify(obj), 'utf8').toString('base64');
}
function decodeCursor(s: string) {
```

```
  return JSON.parse(Buffer.from(s, 'base64').toString('utf8'));
}

@Injectable()
export class StockReadService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async getBalanceCurrent(tenantId: string, q: BalanceQueryDto) {
    // balance por item e location:
inventory_items/{itemId}/balances/{locationId}
    // para listar tudo (sem itemId) você precisaria de uma coleção
materializada.
    // MVP: exige itemId (recomendado).
    if (!q.itemId) throw new Error('ITEM_ID_REQUIRED');

    if (q.locationId) {
      const ref =
this.db.collection('inventory_items').doc(q.itemId).collection('balanc
es').doc(q.locationId);
      const snap = await ref.get();
      if (!snap.exists) return { itemId: q.itemId, locationId:
q.locationId, qtyOnHand: 0 };
      const data = snap.data()!;
      if (data.tenantId !== tenantId) throw new
Error('TENANT_MISMATCH');
      return data;
    }

    const snap = await
this.db.collection('inventory_items').doc(q.itemId).collection('balanc
es')
      .where('tenantId', '==', tenantId)
      .get();

    return snap.docs.map(d => d.data());
  }

  async listMoves(tenantId: string, q: MovesQueryDto) {
    const limit = q.limit ?? 50;
    const order = q.order ?? 'desc';

    let ref: FirebaseFirestore.Query = this.db
      .collection('inventory_items').doc(q.itemId)
      .collection('moves')
      .where('tenantId', '==', tenantId);

    if (q.locationId) ref = ref.where('locationId', '==',
q.locationId);

    ref = ref.orderBy('performedAt', order).orderBy('moveId', order);

    if (q.cursor) {
      const c = decodeCursor(q.cursor);
      ref = ref.startAfter(c.performedAt, c.moveId);
    }

    const snap = await ref.limit(limit).get();
    const items = snap.docs.map(d => d.data());

    const last = snap.docs[snap.docs.length - 1];
```

```
    const nextCursor = last ? encodeCursor({ performedAt:
last.get('performedAt'), moveId: last.get('moveId') }) : null;

    return { items, nextCursor };
  }
}
```

2.4 StockController: endpoints read

```
// src/stock/stock.controller.ts
import { Controller, Get, Query, UseGuards } from '@nestjs/common';
import { ApiBearerAuth, ApiTags, ApiOperation } from
'@nestjs/swagger';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { StockReadService } from './stock.read.service';
import { BalanceQueryDto } from './dto/balance-query.dto';
import { MovesQueryDto } from './dto/moves-query.dto';

@ApiTags('Stock')
@ApiBearerAuth('bearerAuth')
@Controller('/api/stock')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class StockController {
  constructor(
    // seu StockService (write) pode continuar aqui...
    private readonly read: StockReadService, // ✅ add
  ) {}

  @Get('/balance/current')
  @ApiOperation({ summary: 'Get current balance (requires itemId)' })

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDITOR')
  @Permissions('stock.read')
  async balance(@Actor() actor: any, @Query() q: BalanceQueryDto) {
    return this.read.getBalanceCurrent(actor.tenantId, q);
  }

  @Get('/moves')
  @ApiOperation({ summary: 'List stock moves for itemId (cursor
pagination)' })

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDITOR')
  @Permissions('stock.read')
  async moves(@Actor() actor: any, @Query() q: MovesQueryDto) {
    return this.read.listMoves(actor.tenantId, q);
  }
}
```

2.5 StockModule: incluir StockReadService

```
// src/stock/stock.module.ts
import { Module } from '@nestjs/common';
import { firestoreProvider } from '../common/firestore.provider';
import { AuditModule } from '../audit/audit.module';
import { PoliciesModule } from '../policies/policies.module';
import { StockController } from './stock.controller';
import { StockService } from './stock.service';
```

```
import { StockReadService } from './stock.read.service';

@Module({
  imports: [AuditModule, PoliciesModule],
  providers: [firestoreProvider, StockService, StockReadService],
  controllers: [StockController],
})
export class StockModule {}
```

---

3) Índices Firestore (obrigatórios para produção)

3.1 Locations

- `tenantId + active + name(asc)`

3.2 Moves (subcoleção por item)

Se você filtrar por `locationId` + order:

- `tenantId + locationId + performedAt(desc) + moveId(desc)`

Caso não filtre por locationId, normalmente não precisa (mas dependendo da regra, pode pedir):

- `tenantId + performedAt(desc) + moveId(desc)`

---

4) Firestore Rules (ajuste mínimo para Locations)

Adicione:

```
match /locations/{locationId} {
  allow read: if tenantMatches(resource.data) &&
hasAnyRole(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDI
TOR']);
  allow create: if tenantMatches(request.resource.data) &&
hasAnyRole(['ADMIN','MANAGER']);
  allow update, delete: if false;
}
```

---

5) Observação importante (para ficar “perfeito” no app offline)

O endpoint `/balance/current` exige `itemId` para não depender de collection group e índices complexos.

Se você quiser **listar saldo de todos itens da fazenda** (tela “estoque geral”), o ideal é criar uma coleção materializada:

- `stock_balances/{balanceId}` com campos: `tenantId, itemId, locationId, qtyOnHand, updatedAt`
  e manter isso atualizado a cada movimento.
  Aí você tem:
- `GET /api/stock/balances?locationId=` rápido.

Se você disser **"quero estoque geral por local"**, eu já te monto esse materializador.

Se você quiser, eu também fecho o lado do app (Expo/React Native) com:

- hooks React Query para `requests` (infinite)
- hooks para `stock moves` (infinite)
- cache offline e revalidação
- envio de draft + evidência (foto/voz) com retry idempotente

Abaixo está o "lado do app" (Expo + React Native) **pronto para funcionar bem** com:

- AuthContext (Firebase Auth)
- API client com token + tenant header (opcional) + retry/backoff
- React Query com persistência offline (AsyncStorage)
- Hooks:
  - Requests: list (infinite), get, approve/reject/start/done
  - Stock: balance, moves (infinite)
  - Drafts: create, apply
  - Evidence: upload (Storage) + create metadata (API)
- Idempotência: `clientRequestId` em todas as mutações

Vou assumir que você usa **NestJS API** em `EXPO_PUBLIC_API_URL` e **Firebase Auth** no app.

---

0) Dependências (app)

Instale:

```
npm i @tanstack/react-query @tanstack/react-query-persist-client
npm i @react-native-async-storage/async-storage
npm i axios
npm i uuid
npm i firebase
```

---

1) Firebase init

```
// src/firebase/firebase.ts
import { initializeApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';
```

```
import { getStorage } from 'firebase/storage';

const firebaseConfig = {
  apiKey: process.env.EXPO_PUBLIC_FIREBASE_API_KEY!,
  authDomain: process.env.EXPO_PUBLIC_FIREBASE_AUTH_DOMAIN!,
  projectId: process.env.EXPO_PUBLIC_FIREBASE_PROJECT_ID!,
  storageBucket: process.env.EXPO_PUBLIC_FIREBASE_STORAGE_BUCKET!,
  messagingSenderId:
process.env.EXPO_PUBLIC_FIREBASE_MESSAGING_SENDER_ID!,
  appId: process.env.EXPO_PUBLIC_FIREBASE_APP_ID!,
};

const app = initializeApp(firebaseConfig);

export const auth = getAuth(app);
export const storage = getStorage(app);
```

---

2) AuthContext (token + claims)

```
// src/auth/AuthContext.tsx
import React, { createContext, useContext, useEffect, useMemo,
useState } from 'react';
import { onAuthStateChanged, getIdTokenResult, User } from
'firebase/auth';
import { auth } from '../firebase/firebase';

type Claims = {
  tenantId?: string;
  role?: string;
  permissions?: string[];
};

type AuthState = {
  user: User | null;
  token: string | null;
  claims: Claims | null;
  loading: boolean;
  refresh: () => Promise<void>;
};

const Ctx = createContext<AuthState | null>(null);

export function AuthProvider({ children }: { children: React.ReactNode
}) {
  const [user, setUser] = useState<User | null>(null);
  const [token, setToken] = useState<string | null>(null);
  const [claims, setClaims] = useState<Claims | null>(null);
  const [loading, setLoading] = useState(true);

  const refresh = async () => {
    if (!auth.currentUser) return;
    const idTokenResult = await getIdTokenResult(auth.currentUser,
true);
    setToken(idTokenResult.token);
    setClaims(idTokenResult.claims as Claims);
  };

  useEffect(() => {
    const unsub = onAuthStateChanged(auth, async (u) => {
      setUser(u);
```

```
      if (!u) {
        setToken(null);
        setClaims(null);
        setLoading(false);
        return;
      }
      const idTokenResult = await getIdTokenResult(u, true);
      setToken(idTokenResult.token);
      setClaims(idTokenResult.claims as Claims);
      setLoading(false);
    });
    return () => unsub();
  }, []);

  const value = useMemo(() => ({ user, token, claims, loading, refresh
}), [user, token, claims, loading]);
  return <Ctx.Provider value={value}>{children}</Ctx.Provider>;
}

export function useAuth() {
  const v = useContext(Ctx);
  if (!v) throw new Error('AuthContext missing');
  return v;
}
```

---

3) API Client (axios) com token + retry/backoff + idempotência

```
// src/api/client.ts
import axios from 'axios';

export const api = axios.create({
  baseURL: process.env.EXPO_PUBLIC_API_URL,
  timeout: 20000,
});

export function setAuthHeaders(token: string | null, tenantId?:
string) {
  api.defaults.headers.common['Authorization'] = token ? `Bearer
${token}` : '';
  if (tenantId) api.defaults.headers.common['x-tenant-id'] = tenantId;
// opcional
}

// retry simples
api.interceptors.response.use(undefined, async (error) => {
  const cfg = error.config;
  if (!cfg) throw error;

  cfg.__retryCount = cfg.__retryCount || 0;
  if (cfg.__retryCount >= 2) throw error;

  // retry apenas em erros de rede/timeout/5xx
  const status = error.response?.status;
  const shouldRetry = !status || status >= 500;

  if (!shouldRetry) throw error;

  cfg.__retryCount += 1;
  const delay = 500 * cfg.__retryCount;
  await new Promise((r) => setTimeout(r, delay));
```

```
  return api(cfg);
});
```

No `App.tsx`, após login, aplique `setAuthHeaders(token, claims.tenantId)`.

---

4) React Query + persistência offline (perfeito para campo)

```
// src/query/queryClient.ts
import { QueryClient } from '@tanstack/react-query';
import AsyncStorage from '@react-native-async-storage/async-storage';
import { persistQueryClient } from '@tanstack/react-query-persist-
client';
import { createAsyncStoragePersister } from '@tanstack/query-async-
storage-persister';

export const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      retry: 1,
      staleTime: 30_000,
      gcTime: 24 * 60 * 60 * 1000,
    },
    mutations: {
      retry: 0, // mutações: controladas por idempotência no backend
    },
  },
});

const persister = createAsyncStoragePersister({ storage: AsyncStorage
});

export function setupQueryPersistence() {
  persistQueryClient({
    queryClient,
    persister,
    buster: 'v1',
    maxAge: 7 * 24 * 60 * 60 * 1000,
  });
}
```

No `App.tsx`:

```
// App.tsx
import { QueryClientProvider } from '@tanstack/react-query';
import { queryClient, setupQueryPersistence } from
'./src/query/queryClient';
import { AuthProvider } from './src/auth/AuthContext';
import { useEffect } from 'react';

export default function App() {
  useEffect(() => setupQueryPersistence(), []);

  return (
    <AuthProvider>
      <QueryClientProvider client={queryClient}>
        {/* suas rotas */}
      </QueryClientProvider>
    </AuthProvider>
```

```
  );
}
```

---

5) Hooks: Requests (infinite + actions)

5.1 Tipos de cursor

O backend retorna `{ items, nextCursor }`. React Query infinite usa `getNextPageParam`.

```
// src/api/requests.ts
import { api } from './client';
import { v4 as uuid } from 'uuid';

export type RequestsListParams = {
  status?: string;
  limit?: number;
  order?: 'asc' | 'desc';
  cursor?: string | null;
};

export async function listRequests(params: RequestsListParams) {
  const res = await api.get('/api/requests', { params });
  return res.data as { items: any[]; nextCursor: string | null };
}

export async function getRequest(id: string) {
  const res = await api.get(`/api/requests/${id}`);
  return res.data;
}

export async function approveRequest(id: string) {
  const res = await api.post(`/api/requests/${id}/approve`, {
clientRequestId: uuid() });
  return res.data;
}

export async function rejectRequest(id: string, reason: string) {
  const res = await api.post(`/api/requests/${id}/reject`, { reason,
clientRequestId: uuid() });
  return res.data;
}

export async function startFulfillment(id: string, note?: string) {
  const res = await api.post(`/api/requests/${id}/start-fulfillment`,
{ note: note ?? null, clientRequestId: uuid() });
  return res.data;
}

export async function doneRequest(id: string, moveIds: string[],
note?: string) {
  const res = await api.post(`/api/requests/${id}/done`, { moveIds,
note: note ?? null, clientRequestId: uuid() });
  return res.data;
}
```

Hooks

```
// src/hooks/useRequests.ts
import { useInfiniteQuery, useMutation, useQuery, useQueryClient }
from '@tanstack/react-query';
import { listRequests, getRequest, approveRequest, rejectRequest,
startFulfillment, doneRequest } from '../api/requests';

export function useRequestsList(status?: string) {
  return useInfiniteQuery({
    queryKey: ['requests', 'list', status ?? 'all'],
    queryFn: ({ pageParam }) => listRequests({ status, limit: 30,
order: 'desc', cursor: pageParam ?? null }),
    initialPageParam: null as string | null,
    getNextPageParam: (lastPage) => lastPage.nextCursor ?? undefined,
  });
}

export function useRequest(id: string) {
  return useQuery({
    queryKey: ['requests', 'get', id],
    queryFn: () => getRequest(id),
    enabled: !!id,
  });
}

export function useRequestActions(id: string) {
  const qc = useQueryClient();

  const approve = useMutation({
    mutationFn: () => approveRequest(id),
    onSuccess: () => qc.invalidateQueries({ queryKey: ['requests'] }),
  });

  const reject = useMutation({
    mutationFn: (reason: string) => rejectRequest(id, reason),
    onSuccess: () => qc.invalidateQueries({ queryKey: ['requests'] }),
  });

  const fulfill = useMutation({
    mutationFn: (note?: string) => startFulfillment(id, note),
    onSuccess: () => qc.invalidateQueries({ queryKey: ['requests'] }),
  });

  const done = useMutation({
    mutationFn: (args: { moveIds: string[]; note?: string }) =>
doneRequest(id, args.moveIds, args.note),
    onSuccess: () => qc.invalidateQueries({ queryKey: ['requests'] }),
  });

  return { approve, reject, fulfill, done };
}
```

---

6) Hooks: Stock (balance + moves infinite)

```
// src/api/stock.ts
import { api } from './client';

export async function getBalanceCurrent(itemId: string, locationId?:
string) {
  const res = await api.get('/api/stock/balance/current', { params: {
itemId, locationId } });
```

```
  return res.data;
}

export async function listMoves(params: { itemId: string; locationId?:
string; limit?: number; order?: 'asc'|'desc'; cursor?: string | null
}) {
  const res = await api.get('/api/stock/moves', { params });
  return res.data as { items: any[]; nextCursor: string | null };
}
// src/hooks/useStock.ts
import { useInfiniteQuery, useQuery } from '@tanstack/react-query';
import { getBalanceCurrent, listMoves } from '../api/stock';

export function useBalance(itemId: string, locationId?: string) {
  return useQuery({
    queryKey: ['stock', 'balance', itemId, locationId ?? 'all'],
    queryFn: () => getBalanceCurrent(itemId, locationId),
    enabled: !!itemId,
  });
}

export function useMoves(itemId: string, locationId?: string) {
  return useInfiniteQuery({
    queryKey: ['stock', 'moves', itemId, locationId ?? 'all'],
    queryFn: ({ pageParam }) => listMoves({ itemId, locationId, limit:
30, order: 'desc', cursor: pageParam ?? null }),
    initialPageParam: null as string | null,
    enabled: !!itemId,
    getNextPageParam: (lastPage) => lastPage.nextCursor ?? undefined,
  });
}
```

---

7) Evidence: upload foto/voz para Storage + metadata na API

7.1 Upload helper

```
// src/api/evidence.ts
import { ref, uploadBytes, getDownloadURL } from 'firebase/storage';
import { storage } from '../firebase/firebase';
import { api } from './client';
import { v4 as uuid } from 'uuid';

export async function uploadEvidenceFile(tenantId: string, uri:
string, kind: 'PHOTO'|'AUDIO'|'DOC') {
  const id = `ev_${uuid()}`;
  const path = `tenants/${tenantId}/evidences/${id}`;

  // converte uri -> blob
  const resp = await fetch(uri);
  const blob = await resp.blob();

  const r = ref(storage, path);
  await uploadBytes(r, blob, { contentType: kind === 'PHOTO' ?
'image/jpeg' : undefined });

  const url = await getDownloadURL(r);
  return { evidenceId: id, storageRef: path, url };
}

// metadata na API (backend grava no Firestore)
```

```
export async function createEvidenceMetadata(payload: {
  evidenceId: string;
  tenantId: string;
  type: 'PHOTO'|'AUDIO'|'DOC';
  riskType: 'A'|'B'|'DOC'|'B_LAB';
  storageRef: string;
  sha256: string;
  clientRequestId: string;
  geo?: { lat: number; lng: number };
}) {
  const res = await api.post('/api/evidences', payload);
  return res.data;
}
```

SHA256 no RN: você pode usar uma lib (`expo-crypto` ou `react-native-quick-crypto`). Se quiser, eu te mando a implementação exata para Expo.

---

8) Drafts: create + apply

```
// src/api/drafts.ts
import { api } from './client';
import { v4 as uuid } from 'uuid';

export async function createDraft(payload: any) {
  const res = await api.post('/api/drafts', { ...payload,
clientRequestId: uuid() });
  return res.data;
}

export async function applyDraft(draftId: string, payload?: any) {
  const res = await api.post(`/api/drafts/${draftId}/apply`, {
...(payload ?? {}), clientRequestId: uuid() });
  return res.data;
}
// src/hooks/useDrafts.ts
import { useMutation, useQueryClient } from '@tanstack/react-query';
import { createDraft, applyDraft } from '../api/drafts';

export function useDraftMutations() {
  const qc = useQueryClient();

  const create = useMutation({
    mutationFn: (payload: any) => createDraft(payload),
    onSuccess: () => qc.invalidateQueries({ queryKey: ['requests'] }),
  });

  const apply = useMutation({
    mutationFn: (args: { draftId: string; payload?: any }) =>
applyDraft(args.draftId, args.payload),
    onSuccess: () => qc.invalidateQueries({ queryKey: ['stock'] }),
  });

  return { create, apply };
}
```

---

9) Tela exemplo (simples) — lista Requests com infinite scroll

```
// src/screens/RequestsListScreen.tsx
import React from 'react';
import { FlatList, Text, TouchableOpacity, View } from 'react-native';
import { useRequestsList } from '../hooks/useRequests';

export function RequestsListScreen({ status }: { status?: string }) {
  const q = useRequestsList(status);

  const items = q.data?.pages.flatMap(p => p.items) ?? [];

  return (
    <View style={{ flex: 1, padding: 12 }}>
      <FlatList
        data={items}
        keyExtractor={(it) => it.requestId}
        onEndReached={() => q.fetchNextPage()}
        onEndReachedThreshold={0.5}
        ListFooterComponent={() => q.isFetchingNextPage ?
<Text>Carregando...</Text> : null}
        renderItem={({ item }) => (
          <TouchableOpacity style={{ padding: 12, borderWidth: 1,
marginBottom: 8 }}>
            <Text style={{ fontWeight: '700' }}>{item.type} •
{item.status}</Text>
            <Text>ID: {item.requestId}</Text>
          </TouchableOpacity>
        )}
      />
    </View>
  );
}
```

---

10) MUITO importante para “funcionar perfeito”

10.1 Idempotência no backend

No app, toda mutação já manda `clientRequestId`.
No backend, implemente `idempotency` (doc lock) para:

- createEvidence
- createDraft
- applyDraft
- requests actions (approve/reject/fulfill/done)

10.2 Erros bem tratados

Quando vier `OPERATION_BLOCKED` (HTTP 409/400):

- mostrar motivo “amigável” no app
- sugerir ação (ex.: “estoque insuficiente” → “faça entrada NF ou ajuste”)

Se você quiser, eu posso te mandar um `errorMap.ts` com mensagens simples para operador.

---

11) O que falta (seu “estoque geral por local”)

Para o app ter tela “visão geral do estoque” sem exigir itemId, o ideal é a coleção materializada:

- `stock_balances/{tenantId}_{locationId}_{itemId}`

E um endpoint:

- `GET /api/stock/balances?locationId=loc_...`

Se você confirmar **“quero estoque geral”**, eu entrego:

- materializador no backend (atualiza sempre que move acontece)
- endpoint + hook no app + índices

Quer que eu inclua **estoque geral por local** agora?

Sim — vamos incluir **(A) “estoque geral por local”** com coleção materializada + endpoints + hooks, e **(B) Marketplace com API para vincular e visualizar estoque do fornecedor em “tempo real”** (sem quebrar o isolamento multi-tenant).

Vou te entregar um desenho que funciona de verdade em produção, com segurança.

A) Estoque geral por local (materializado)

1) Nova coleção materializada

Crie uma coleção **flat** (fora das subcoleções) para poder listar tudo rápido:

`stock_balances/{balanceId}`

- `balanceId = ${tenantId}_${locationId}_${itemId}`
- Campos:

```
{
  "tenantId": "t_demo_agro",
  "locationId": "loc_main",
  "itemId": "inv_salt",
  "catalogItemId": "cat_salt_25kg",
  "qtyOnHand": 125,
  "uom": "KG",
  "updatedAt": 1730000000000,
  "lastMoveId": "mv_..."
}
```

Isso evita Collection Group complexa e permite tela “estoque geral”.

2) Atualização automática (no backend)

Sempre que você **aplicar um move** (ISSUE/MOVE/LOSS/ADJUST/INBOUND), no mesmo transaction você atualiza:

1. `inventory_items/{itemId}/balances/{locationId}` (como já faz)
2. `stock_balances/{tenantId_locationId_itemId}`

Patch no `DraftsService.applyDraft` (depois do balance):

```
const matId = `${tenantId}_${locationId}_${itemId}`;
const matRef = this.db.collection('stock_balances').doc(matId);

tx.set(matRef, {
  tenantId,
  locationId,
  itemId,
  catalogItemId: draft.catalogItemId ?? null,
  qtyOnHand: balanceAfter,
  uom,
  updatedAt: Date.now(),
  lastMoveId: moveId,
}, { merge: true });
```

3) Endpoint: listar estoque geral

DTO

```
// src/stock/dto/balances-list.dto.ts
import { IsOptional, IsString, IsInt, Min, Max, IsIn } from 'class-
validator';
import { Type } from 'class-transformer';

export class BalancesListDto {
  @IsOptional() @IsString() locationId?: string;
  @IsOptional() @IsString() itemId?: string;

  @IsOptional() @Type(() => Number) @IsInt() @Min(1) @Max(200)
  limit?: number = 100;

  @IsOptional() @IsIn(['desc','asc'])
  order?: 'desc' | 'asc' = 'desc';

  @IsOptional() @IsString()
  cursor?: string; // base64: { updatedAt, balanceId }
}
```

Read service

```
// src/stock/stock.balances.read.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { BalancesListDto } from './dto/balances-list.dto';

function encodeCursor(obj: any) {
  return Buffer.from(JSON.stringify(obj), 'utf8').toString('base64');
}
function decodeCursor(s: string) {
  return JSON.parse(Buffer.from(s, 'base64').toString('utf8'));
}
```

```
@Injectable()
export class StockBalancesReadService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async list(tenantId: string, q: BalancesListDto) {
    const limit = q.limit ?? 100;
    const order = q.order ?? 'desc';

    let ref: FirebaseFirestore.Query =
this.db.collection('stock_balances')
      .where('tenantId', '==', tenantId);

    if (q.locationId) ref = ref.where('locationId', '==',
q.locationId);
    if (q.itemId) ref = ref.where('itemId', '==', q.itemId);

    ref = ref.orderBy('updatedAt', order).orderBy('__name__', order);

    if (q.cursor) {
      const c = decodeCursor(q.cursor);
      ref = ref.startAfter(c.updatedAt, c.balanceId);
    }

    const snap = await ref.limit(limit).get();
    const items = snap.docs.map(d => ({ balanceId: d.id, ...d.data()
}));

    const last = snap.docs[snap.docs.length - 1];
    const nextCursor = last ? encodeCursor({ updatedAt:
last.get('updatedAt'), balanceId: last.id }) : null;

    return { items, nextCursor };
  }
}
```

Controller

```
// src/stock/stock.controller.ts (adicionar)
@Get('/balances')
@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDITOR')
@Permissions('stock.read')
async balances(@Actor() actor: any, @Query() q: BalancesListDto) {
  return this.balancesRead.list(actor.tenantId, q);
}
```

4) Índices Firestore necessários

- tenantId + updatedAt(desc) + __name__(desc)
- tenantId + locationId + updatedAt(desc) + __name__(desc)
- tenantId + itemId + updatedAt(desc) + __name__(desc)

5) Firestore Rules (stock_balances read-only p/ cliente)

```
match /stock_balances/{id} {
  allow read: if tenantMatches(resource.data) &&
hasAnyRole(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDI
TOR']);
  allow create, update, delete: if false;
}
```

6) Hook no app (Expo)

```
// src/api/stockBalances.ts
import { api } from './client';
export async function listBalances(params: any) {
  const res = await api.get('/api/stock/balances', { params });
  return res.data;
}
// src/hooks/useStockBalances.ts
import { useInfiniteQuery } from '@tanstack/react-query';
import { listBalances } from '../api/stockBalances';

export function useStockBalances(locationId?: string) {
  return useInfiniteQuery({
    queryKey: ['stock', 'balances', locationId ?? 'all'],
    queryFn: ({ pageParam }) => listBalances({ locationId, limit: 50,
order: 'desc', cursor: pageParam ?? null }),
    initialPageParam: null as string | null,
    getNextPageParam: (last) => last.nextCursor ?? undefined,
  });
}
```

B) Marketplace: vincular e visualizar estoque do fornecedor "em tempo real"

Aqui está o ponto crítico: **fornecedor e comprador normalmente são tenants diferentes**.
Você **não pode** simplesmente "ler balances do tenant do fornecedor", porque isso quebraria o isolamento.

A solução correta é publicar um **"espelho autorizado" (view)** do estoque do fornecedor no contexto do marketplace.

Arquitetura recomendada (segura e real-time)

1. **Supplier** mantém seu estoque no próprio tenant (normal)
2. Quando ele decide vender no marketplace, ele cria/atualiza **uma oferta**
3. O backend sincroniza (espelha) o saldo do fornecedor para uma coleção pública/controlada do marketplace:

`market_offers/{offerId}` (cadastro da oferta)
`market_offer_stock/{offerId}_{skuOrItem}` (saldo publicado da oferta)

Estes docs são "real-time" para o comprador via API e/ou Firestore listener.

1) Coleções do Marketplace

```
market_offers/{offerId}
{
  "offerId": "off_123",
  "supplierTenantId": "t_supplier_01",
  "supplierId": "sup_abc",
  "catalogItemKey": "cat_salt_25kg",
  "title": "Sal mineral 25kg",
  "uom": "KG",
```

```
  "price": 120.50,
  "currency": "BRL",
  "visibility": "PUBLIC",
  "allowedBuyerTenants": ["t_buyer_01"],
  "createdAt": 1730000000000,
  "updatedAt": 1730000000000,
  "active": true
}
```

market_offer_stock/{id}

`id = ${offerId}_${locationId}` ou `${offerId}` (depende se você quer por local)

```
{
  "id": "off_123_loc_main",
  "offerId": "off_123",
  "supplierTenantId": "t_supplier_01",
  "qtyAvailable": 980,
  "updatedAt": 1730000000000
}
```

2) Como vincular o estoque do fornecedor à oferta

Você precisa de um **mapeamento** entre oferta e "fonte do estoque" do fornecedor:

- `supplierTenantId`
- `supplierItemId` (inventory_items do fornecedor)
- `locationId` (opcional)

Exemplo dentro da oferta:

```
"source": {
  "itemId": "inv_salt",
  "locationId": "loc_main"
}
```

3) Sincronização "tempo real"

Existem 2 jeitos bons:

Opção A (mais simples e robusta): Trigger/Worker do backend

- Sempre que atualizar `balances` ou `stock_balances` no tenant do fornecedor, o backend atualiza `market_offer_stock` relacionado.

Como você já materializa `stock_balances`, fica fácil:

- ouvir changes em `stock_balances` do tenant do fornecedor
- encontrar ofertas que apontam para aquele `itemId/locationId`
- atualizar `market_offer_stock.qtyAvailable`

Pode ser Cloud Functions, ou um worker NestJS rodando com listener (em produção eu prefiro Cloud Functions/Cloud Run job).

Opção B (API "pull" + cache curto)

- O comprador chama `GET /market/offers/:id/stock`
- O backend lê o estoque do fornecedor e responde.
- Para "real-time", colocar cache de 3–5s e usar SSE/WebSocket.

Funciona, mas custa mais e é mais complexo.

Eu recomendo Opção A: Firestore update automático → app vê em tempo real com listener.

4) Segurança de acesso (quem pode ver stock)

Regras de visibilidade:

- `PUBLIC`: qualquer tenant autenticado pode ver `market_offers`
- `PRIVATE`: só tenants listados em `allowedBuyerTenants` veem a oferta e o stock
- `PARTNER`: só buyers com contrato ativo (fase 2)

No Firestore Rules do marketplace:

- permitir read se `visibility == PUBLIC` ou `tenantId in allowedBuyerTenants`
- write de `market_offer_stock` **somente servidor**.

5) API do Marketplace (REST) no Nest

Endpoints principais

- `GET /api/market/offers` (listar ofertas)
- `GET /api/market/offers/:id` (detalhe)
- `GET /api/market/offers/:id/stock` (retorna `market_offer_stock`, já espelhado)
- `POST /api/market/offers` (SUPPLIER cria oferta)
- `PATCH /api/market/offers/:id` (SUPPLIER atualiza)
- (fase 2) `POST /api/market/orders` / `contracts` / escrow etc.

6) "Vincular fornecedor" (supplier linkage)

Para garantir que o fornecedor só publique o próprio estoque:

- o backend exige que `actor.tenantId == supplierTenantId` ao criar oferta
- e que `source.itemId` exista no tenant dele

7) Hook no app (comprador) real-time

Você pode usar:

- **API** para listar ofertas

- **Firestore listener** para `market_offer_stock` do offer selecionado (real-time perfeito)
  OU
- **API** com polling leve (a cada 5–10s) se não quiser listener

Se você quer real-time mesmo (como você pediu), use listener.

O que eu preciso decidir no seu modelo (mas eu já faço a melhor escolha padrão)

Para eu te entregar o código final do Marketplace, vou assumir:

1. Marketplace é **multi-tenant**, mas as ofertas ficam numa coleção global (com ACL por tenant)
2. Estoque exibido é um **espelho** (`market_offer_stock`) — nunca leitura direta do tenant do fornecedor
3. Sync via atualização em `stock_balances` do fornecedor (materializado)

Se isso está ok (é o melhor e mais seguro), eu já te mando na próxima mensagem:

- `MarketModule` completo (controller + service + DTOs)
- Firestore Rules para `market_offers` e `market_offer_stock`
- Índices Firestore
- Job/trigger lógico (pseudo-código) de sincronização
- Hooks do app (list offers + subscribe stock real-time)

Quer que o **estoque publicado** seja:

- **por oferta (total)**, ou
- **por local** (ex.: fornecedor tem 3 depósitos e você quer mostrar por região/local)?

Fechado: **por oferta (total)** e **só aparecem empresas/ofertas próximas** (pela localização do comprador). E, ao mesmo tempo, **estoque do fornecedor "em tempo real" via API**, sem quebrar o isolamento multi-tenant.

A melhor implementação (segura e que funciona em produção) é:

- **O app do comprador NÃO lê Firestore do fornecedor** (nunca).
- O comprador **consome a API do Marketplace**, que:
  1. filtra ofertas por **distância (lat/lng + raio)**
  2. retorna **estoque total disponível da oferta**
  3. fornece **atualização em tempo real** (SSE/WebSocket) ou fallback com polling curto.

Abaixo já vai o modelo completo.

---

1) Modelo de dados do Marketplace

1.1 `market_offers/{offerId}`

Representa a oferta publicada pelo fornecedor (indústria/loja/insumos).

```
{
  "offerId": "off_123",
  "active": true,

  "supplierTenantId": "t_supplier_01",
  "supplierName": "Insumos X",
  "supplierLocation": {
    "lat": -15.61,
    "lng": -56.09
  },

  "serviceRadiusKm": 120,
  "regionLabel": "Cuiabá e entorno",

  "catalogItemId": "cat_salt_25kg",
  "title": "Sal mineral 25kg",
  "uom": "KG",
  "price": 120.50,
  "currency": "BRL",

  "source": {
    "supplierItemId": "inv_salt",
    "aggregateMode": "TOTAL"
  },

  "visibility": "PUBLIC",
  "allowedBuyerTenants": [],

  "createdAt": 1730000000000,
  "updatedAt": 1730000000000
}
```

1.2 `market_offer_stock/{offerId}`

Espelho materializado do estoque total da oferta (não é o estoque bruto do fornecedor).

```
{
  "offerId": "off_123",
  "qtyAvailable": 980,
  "updatedAt": 1730000000000,
  "lastSyncReason": "STOCK_MOVE|INBOUND|ADJUST"
}
```

Por que assim? Porque o comprador só precisa do “disponível para compra”, não do estoque interno detalhado do fornecedor.

---

2) Como calcular “estoque total por oferta”

Você já vai ter `stock_balances` materializado no tenant do fornecedor (parte A que incluímos).

Então, para uma oferta `off_123` com `source.supplierItemId = inv_salt`:

- Somar `qtyOnHand` de **todas as locations** do fornecedor para aquele item (ou apenas locations “marketEnabled”, se quiser).
- Atualizar `market_offer_stock/off_123.qtyAvailable`.

Regra de negócio recomendada

- O fornecedor pode marcar quais locais entram no marketplace:
  - `locations/{id}.marketEnabled = true`
- O total exibido considera só esses locais (evita mostrar estoque “interno”).

---

3) API do Marketplace (com filtro por proximidade)

3.1 Listar ofertas próximas

GET `/api/market/offers?lat=-15.6&lng=-56.1&radiusKm=150&limit=50`

O backend faz:

- pega ofertas ativas
- calcula distância (Haversine)
- filtra por:
  - `distance <= min(radiusKm, offer.serviceRadiusKm)`
- retorna ordenado por distância
- inclui `qtyAvailable` do `market_offer_stock`

Resposta:

```
{
  "items": [
    {
      "offerId": "off_123",
      "title": "Sal mineral 25kg",
      "price": 120.5,
      "uom": "KG",
      "supplierName": "Insumos X",
      "distanceKm": 18.4,
      "qtyAvailable": 980,
      "updatedAt": 1730000000000
    }
  ]
}
```

3.2 Detalhe da oferta (validando proximidade)

GET `/api/market/offers/off_123?lat=...&lng=...`

- Se o comprador estiver fora do raio: `403 OUT_OF_RANGE`

3.3 Estoque "tempo real" por oferta (via API)

GET `/api/market/offers/off_123/stock?lat=...&lng=...`

- Retorna `{ qtyAvailable, updatedAt }`
- Valida proximidade também

---

4) Tempo real de estoque via API (recomendado: SSE)

Como você pediu "em tempo real", o mais eficiente é um stream:

4.1 SSE stream de ofertas próximas

GET `/api/market/stream?lat=...&lng=...&radiusKm=150`

- O backend:
  - envia snapshot inicial das ofertas em alcance
  - escuta alterações em `market_offer_stock` (listener server-side)
  - publica eventos apenas das ofertas que continuam dentro do raio

Eventos:

- `offer_stock_update` (quando qty muda)
- `offer_deactivated` (se oferta desativou)
- `offer_created` (nova oferta na região)

Se você preferir WebSocket, dá também, mas SSE já resolve bem.

4.2 Fallback simples (se não usar SSE agora)

No app, usar React Query com:

- `refetchInterval: 5000` para a lista
- `refetchInterval: 2000` na tela de detalhe (opcional)

---

5) Segurança e isolamento (multi-tenant)

Para garantir "perfeito":

- **Clientes NÃO leem** `market_offer_stock` direto do Firestore.
- Firestore Rules:

    - `market_offers`: read false (ou read só servidor)
    - `market_offer_stock`: read false
    - writes: somente servidor

Assim, **toda validação de proximidade fica no backend** (Firestore Rules não calculam distância).

Firestore Rules (market)

```
match /market_offers/{id} {
  allow read, write: if false;
}
match /market_offer_stock/{id} {
  allow read, write: if false;
}
```

Resultado: ninguém "fura" a regra de proximidade via leitura direta.

---

6) Implementação NestJS (estrutura dos módulos)

`MarketModule`

- `MarketOffersService`
- `MarketOffersController`
- `MarketStreamController` (SSE)
- `MarketSyncService` (sincroniza oferta ← estoque)

Endpoints

- `GET /api/market/offers`
- `GET /api/market/offers/:id`
- `GET /api/market/offers/:id/stock`
- `GET /api/market/stream` (SSE)
- `POST /api/market/offers` (SUPPLIER/MANAGER do tenant fornecedor)
- `PATCH /api/market/offers/:id` (SUPPLIER/MANAGER)

---

7) Como sincronizar stock → market_offer_stock

Você já atualiza `stock_balances` a cada movimento.

Agora, crie um sincronizador no backend:

Padrão robusto (recomendado)

- Quando aplicar movimento / inbound / adjust:
    1. atualiza `stock_balances` do fornecedor
    2. chama `MarketSyncService.syncOffersForItem(tenantId, itemId)`

3. esse sync:
    - busca offers do fornecedor com `source.supplierItemId == itemId`
    - recalcula total somando balances do fornecedor (marketEnabled)
    - atualiza `market_offer_stock/{offerId}`

Mesmo sem trigger do Firestore, isso garante "real-time" porque a atualização acontece no mesmo fluxo do backend.

---

8) App (Expo): hooks do marketplace "prontos"

8.1 Listar ofertas próximas (React Query)

```
export function useMarketOffersNearMe(lat: number, lng: number,
radiusKm = 150) {
  return useQuery({
    queryKey: ['market', 'offers', lat, lng, radiusKm],
    queryFn: async () => (await api.get('/api/market/offers', {
params: { lat, lng, radiusKm, limit: 50 } })).data,
    staleTime: 10_000,
    refetchInterval: 10_000, // fallback caso SSE não esteja ativo
  });
}
```

8.2 SSE (tempo real de verdade)

No React Native você pode usar uma lib tipo `react-native-sse` (ou WebSocket). A lógica:

- abre stream `/api/market/stream?lat&lng&radiusKm`
- ao receber `offer_stock_update`, atualiza o cache do React Query

---

9) "Empresas próximas que utiliza o sistema"

Isso vem naturalmente no `GET /market/offers`:

- cada oferta traz `supplierName`, `distanceKm`, `regionLabel`
- você pode ter endpoint adicional:
    - `GET /api/market/companies?lat&lng&radiusKm`
    - lista fornecedores ativos no marketplace

Próximo passo (eu já deixo pronto para você colar)

Se você confirmar que está ok, eu te mando o **código completo do MarketModule** (NestJS) com:

- DTOs
- Controller (offers + stock)
- Controller SSE
- Service com Haversine
- SyncService integrado no applyDraft/inbound
- Índices recomendados (para filtrar offers por active + region)
- Hooks de app (list + detalhe + SSE listener)

Só preciso de 1 decisão técnica:

- você quer o filtro de proximidade baseado em:
    1. **localização do fornecedor** (um ponto por empresa/oferta) ✅ *mais simples*
    2. **localizações múltiplas** (vários depósitos e o mais próximo determina a oferta) *(mais preciso, mais complexo)*

Como você pediu "por oferta total", eu recomendo **(1)**: **um ponto por oferta/empresa** + raio de serviço.

Perfeito — isso vira uma regra de **"raio adaptativo"**: se não houver empresa/oferta próxima, o sistema **expande automaticamente a área** até encontrar alguma (com limites para não ficar infinito).

Abaixo vai o padrão completo (API + app) do jeito mais robusto.

---

1) Regra de negócio: Raio adaptativo

Comportamento

1. O comprador informa `lat/lng` e (opcional) `radiusKm` inicial (ex.: 50 km).
2. A API busca ofertas dentro do raio efetivo.
3. Se **não achar nenhuma**, ela aumenta o raio e tenta de novo.
4. Para quando:
    - encontrar pelo menos 1 oferta **ou**
    - atingir `maxRadiusKm` (ex.: 800 km)
5. Retorna sempre:
    - `usedRadiusKm` (raio final utilizado)
    - `expanded` (se expandiu)
    - `attempts` (quantas iterações)

Recomendação de valores (bons no agro)

- `radiusKm` inicial: **50 km**
- multiplicador: **×2** por tentativa
- `maxRadiusKm`: **800 km** (ou 600 km, dependendo do estado)
- `maxAttempts`: **5** (50→100→200→400→800)

---

2) API: `GET /api/market/offers` com expansão automática

Query params

- `lat` (obrigatório)
- `lng` (obrigatório)
- `radiusKm` (opcional, default 50)
- `maxRadiusKm` (opcional, default 800)
- `limit` (opcional, default 50)
- `expand` (opcional, default true)

Resposta

```
{
  "items": [ ... ],
  "usedRadiusKm": 200,
  "expanded": true,
  "attempts": 3
}
```

---

3) Implementação NestJS (Service) — lógica de expansão

Observação: como o marketplace vai calcular distância no backend (Haversine), o "loop" roda **em memória** sobre as ofertas ativas. Para escalar melhor, depois dá pra migrar para Geohash / índice regional, mas isso já funciona muito bem no MVP.

```
// src/market/market-offers.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

function haversineKm(a: {lat:number; lng:number}, b: {lat:number;
lng:number}) {
  const R = 6371;
  const toRad = (x:number) => x * Math.PI / 180;
  const dLat = toRad(b.lat - a.lat);
  const dLng = toRad(b.lng - a.lng);
  const s1 = Math.sin(dLat/2) ** 2;
  const s2 = Math.cos(toRad(a.lat)) * Math.cos(toRad(b.lat)) *
(Math.sin(dLng/2) ** 2);
  return 2 * R * Math.asin(Math.sqrt(s1 + s2));
}

@Injectable()
export class MarketOffersService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async listNear(
    tenantId: string, // buyer tenant (pode usar para filtros futuros)
    params: { lat: number; lng: number; radiusKm?: number;
maxRadiusKm?: number; limit?: number; expand?: boolean }
  ) {
    const origin = { lat: params.lat, lng: params.lng };
    let radius = Math.max(1, params.radiusKm ?? 50);
    const maxRadius = Math.max(radius, params.maxRadiusKm ?? 800);
```

```
    const limit = Math.min(Math.max(params.limit ?? 50, 1), 100);
    const expand = params.expand !== false;

    // 1) carrega ofertas ativas (você pode futuramente paginar /
indexar por região)
    const snap = await this.db.collection('market_offers')
      .where('active', '==', true)
      .get();

    const offers = snap.docs.map(d => d.data());

    let attempts = 0;
    let expanded = false;

    while (true) {
      attempts += 1;

      // 2) filtra por proximidade e raio de serviço da própria oferta
      const filtered = offers
        .map((o:any) => {
          const supplierLoc = o.supplierLocation;
          if (!supplierLoc?.lat || !supplierLoc?.lng) return null;

          const distanceKm = haversineKm(origin, supplierLoc);
          const serviceRadiusKm = Number(o.serviceRadiusKm ?? 0) || 0;
          const effectiveOfferRadius = serviceRadiusKm > 0 ?
Math.min(serviceRadiusKm, radius) : radius;

          const inRange = distanceKm <= effectiveOfferRadius;
          return inRange ? { ...o, distanceKm } : null;
        })
        .filter(Boolean)
        .sort((a:any, b:any) => a.distanceKm - b.distanceKm)
        .slice(0, limit);

      if (filtered.length > 0) {
        // 3) junta estoque espelhado por offerId (em lote)
        const ids = filtered.map((o:any) => o.offerId);
        const stockSnaps = await Promise.all(ids.map((id:string) =>
this.db.collection('market_offer_stock').doc(id).get()));
        const stockMap = new Map<string, any>();
        stockSnaps.forEach(s => { if (s.exists) stockMap.set(s.id,
s.data()); });

        const items = filtered.map((o:any) => {
          const st = stockMap.get(o.offerId);
          return {
            offerId: o.offerId,
            title: o.title,
            price: o.price,
            uom: o.uom,
            supplierName: o.supplierName,
            distanceKm: Number(o.distanceKm.toFixed(1)),
            qtyAvailable: st?.qtyAvailable ?? 0,
            updatedAt: st?.updatedAt ?? null,
            regionLabel: o.regionLabel ?? null
          };
        });

        return { items, usedRadiusKm: radius, expanded, attempts };
      }
```

```
      // se não achou nada
      if (!expand || radius >= maxRadius || attempts >= 6) {
        return { items: [], usedRadiusKm: radius, expanded, attempts
};
      }

      // expande
      radius = Math.min(maxRadius, radius * 2);
      expanded = true;
    }
  }
}
```

---

4) Controller do endpoint

```
// src/market/market.controller.ts
import { Controller, Get, Query, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Actor } from '../common/decorators/actor.decorator';
import { MarketOffersService } from './market-offers.service';

@Controller('/api/market')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class MarketController {
  constructor(private readonly market: MarketOffersService) {}

  @Get('/offers')
  async offers(
    @Actor() actor: any,
    @Query('lat') lat: string,
    @Query('lng') lng: string,
    @Query('radiusKm') radiusKm?: string,
    @Query('maxRadiusKm') maxRadiusKm?: string,
    @Query('limit') limit?: string,
    @Query('expand') expand?: string
  ) {
    return this.market.listNear(actor.tenantId, {
      lat: Number(lat),
      lng: Number(lng),
      radiusKm: radiusKm ? Number(radiusKm) : undefined,
      maxRadiusKm: maxRadiusKm ? Number(maxRadiusKm) : undefined,
      limit: limit ? Number(limit) : undefined,
      expand: expand ? expand !== 'false' : true
    });
  }
}
```

---

5) App: UX “expandindo área” (sem confundir o usuário)

Tela de marketplace

- Você inicia com raio 50 km (ou o que o usuário escolher).
- Se a API responder `expanded: true`, mostre um aviso simples:

“Sem ofertas próximas. Expandimos a busca para 200 km.”

Hook React Query

```
export function useMarketOffersNearMe(lat: number, lng: number) {
  return useQuery({
    queryKey: ['market', 'offers', lat, lng],
    queryFn: async () => (await api.get('/api/market/offers', {
params: { lat, lng, radiusKm: 50, maxRadiusKm: 800, limit: 50, expand:
true } })).data,
    staleTime: 10_000,
    refetchInterval: 10_000
  });
}
```

---

6) “Empresas próximas que utiliza o sistema”

Na prática, isso já vem na listagem (cada oferta tem supplierName + distanceKm).
Se você quiser uma lista só de empresas, a API pode agrupar por fornecedor e retornar um `companies` separado.

---

7) Importante: performance (quando crescer)

Quando houver milhares de ofertas, a melhor evolução é:

- salvar `geohash` do fornecedor
- filtrar candidatos por geohash prefixos antes do Haversine
- manter “index por região” (ex.: estado/UF)

Mas para MVP e escala regional, o que te passei funciona muito bem.

Se você quiser, eu já te mando também:

1. **MarketSyncService** (atualiza `market_offer_stock` quando muda `stock_balances` do fornecedor)
2. **Endpoints de cadastro de oferta** (`POST/PATCH /api/market/offers`) com validação de tenant fornecedor + source itemId.

A seguir está o **MarketModule completo** (NestJS) com:

- `POST /api/market/offers` (fornecedor cria oferta)
- `PATCH /api/market/offers/:id` (fornecedor atualiza)
- `GET /api/market/offers` (comprador lista com **raio adaptativo**)
- `GET /api/market/offers/:id` (detalhe, validando proximidade)
- `GET /api/market/offers/:id/stock` (estoque total espelhado, validando proximidade)

- `MarketSyncService` (sincroniza `market_offer_stock` com base no `stock_balances` do fornecedor)
- Integração recomendada: chamar `syncOffersForSupplierItem(...)` sempre que movimentar estoque

Mantive a segurança: **market_offer_stock e market_offers não são lidos diretamente pelo app via Firestore**. Tudo passa pela API (onde validamos proximidade e regras).

---

1) DTOs

```
// src/market/dto/market.dto.ts
import { IsBoolean, IsInt, IsNotEmpty, IsNumber, IsOptional, IsString,
Max, Min } from 'class-validator';
import { Type } from 'class-transformer';

export class MarketListDto {
  @Type(() => Number) @IsNumber() lat: number;
  @Type(() => Number) @IsNumber() lng: number;

  @IsOptional() @Type(() => Number) @IsInt() @Min(1) @Max(800)
  radiusKm?: number = 50;

  @IsOptional() @Type(() => Number) @IsInt() @Min(1) @Max(1200)
  maxRadiusKm?: number = 800;

  @IsOptional() @Type(() => Number) @IsInt() @Min(1) @Max(100)
  limit?: number = 50;

  @IsOptional() @IsBoolean()
  expand?: boolean = true;
}

export class CreateOfferDto {
  @IsString() @IsNotEmpty() title: string;
  @IsString() @IsNotEmpty() catalogItemId: string;
  @IsString() @IsNotEmpty() uom: string;

  @Type(() => Number) @IsNumber()
  price: number;

  @IsOptional() @IsString()
  currency?: string = 'BRL';

  // localização do fornecedor (ponto da empresa/oferta)
  @Type(() => Number) @IsNumber()
  supplierLat: number;

  @Type(() => Number) @IsNumber()
  supplierLng: number;

  @IsOptional() @Type(() => Number) @IsInt() @Min(1) @Max(800)
  serviceRadiusKm?: number = 120;

  @IsOptional() @IsString()
  regionLabel?: string;

  // vínculo com estoque do fornecedor
```

```
  @IsString() @IsNotEmpty()
  supplierItemId: string; // inventory_items do tenant fornecedor

  @IsOptional() @IsString()
  supplierName?: string;

  @IsOptional() @IsString()
  visibility?: 'PUBLIC' | 'PRIVATE' = 'PUBLIC';
}

export class UpdateOfferDto {
  @IsOptional() @IsString() title?: string;
  @IsOptional() @IsString() uom?: string;
  @IsOptional() @Type(() => Number) @IsNumber() price?: number;

  @IsOptional() @Type(() => Number) @IsNumber() supplierLat?: number;
  @IsOptional() @Type(() => Number) @IsNumber() supplierLng?: number;

  @IsOptional() @Type(() => Number) @IsInt() @Min(1) @Max(800)
  serviceRadiusKm?: number;

  @IsOptional() @IsString()
  regionLabel?: string;

  @IsOptional() @IsBoolean()
  active?: boolean;
}
```

---

2) Utils (Haversine + validação range)

```
// src/market/market.utils.ts
export function haversineKm(a: { lat: number; lng: number }, b: { lat:
number; lng: number }) {
  const R = 6371;
  const toRad = (x: number) => (x * Math.PI) / 180;
  const dLat = toRad(b.lat - a.lat);
  const dLng = toRad(b.lng - a.lng);
  const s1 = Math.sin(dLat / 2) ** 2;
  const s2 = Math.cos(toRad(a.lat)) * Math.cos(toRad(b.lat)) *
(Math.sin(dLng / 2) ** 2);
  return 2 * R * Math.asin(Math.sqrt(s1 + s2));
}

export function assertInRange(distanceKm: number, radiusKm: number) {
  if (distanceKm > radiusKm) throw new Error('OUT_OF_RANGE');
}
```

---

3) MarketSyncService (espelha estoque total do fornecedor por oferta)

O que ele faz: para um `supplierTenantId + supplierItemId`:

- soma `stock_balances` do fornecedor daquele item (de locais marketEnabled, se você quiser)
- atualiza `market_offer_stock/{offerId}`

```
// src/market/market-sync.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

@Injectable()
export class MarketSyncService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async syncOfferStock(offerId: string) {
    const offerRef = this.db.collection('market_offers').doc(offerId);
    const offerSnap = await offerRef.get();
    if (!offerSnap.exists) throw new Error('OFFER_NOT_FOUND');

    const offer = offerSnap.data()!;
    if (!offer.active) {
      // se desativou, zera espelho
      await
this.db.collection('market_offer_stock').doc(offerId).set({
        offerId,
        qtyAvailable: 0,
        updatedAt: Date.now(),
        lastSyncReason: 'OFFER_INACTIVE'
      }, { merge: true });
      return;
    }

    const supplierTenantId = offer.supplierTenantId;
    const supplierItemId = offer.source?.supplierItemId;

    // soma saldos materializados do fornecedor
    // stock_balances tem: tenantId, locationId, itemId, qtyOnHand
    // itemId aqui = supplierItemId
    let q = this.db.collection('stock_balances')
      .where('tenantId', '==', supplierTenantId)
      .where('itemId', '==', supplierItemId);

    // opcional: só locais marketEnabled
    // Se você salvar marketEnabled em locations, pode filtrar via
join no backend (2 queries)
    // MVP: soma tudo

    const snap = await q.get();
    let total = 0;
    snap.forEach(d => { total += Number(d.get('qtyOnHand') || 0); });

    await this.db.collection('market_offer_stock').doc(offerId).set({
      offerId,
      qtyAvailable: total,
      updatedAt: Date.now(),
      lastSyncReason: 'STOCK_BALANCES_AGG'
    }, { merge: true });
  }

  async syncOffersForSupplierItem(supplierTenantId: string,
supplierItemId: string) {
    // buscar offers do fornecedor para esse item
    const snap = await this.db.collection('market_offers')
      .where('supplierTenantId', '==', supplierTenantId)
      .where('source.supplierItemId', '==', supplierItemId)
      .get();
```

```
    const offerIds = snap.docs.map(d => d.id);
    for (const offerId of offerIds) {
      await this.syncOfferStock(offerId);
    }
  }
}
```

---

4) MarketOffersService (CRUD + list near com expansão automática)

```
// src/market/market-offers.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';
import { haversineKm, assertInRange } from './market.utils';
import { MarketSyncService } from './market-sync.service';

@Injectable()
export class MarketOffersService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly sync: MarketSyncService
  ) {}

  async createOffer(supplierTenantId: string, actor: any, dto: any) {
    // valida que o item existe no tenant do fornecedor
    const invRef =
this.db.collection('inventory_items').doc(dto.supplierItemId);
    const invSnap = await invRef.get();
    if (!invSnap.exists) throw new Error('SUPPLIER_ITEM_NOT_FOUND');
    const inv = invSnap.data()!;
    if (inv.tenantId !== supplierTenantId) throw new
Error('TENANT_MISMATCH');

    const offerId = makeId('off');
    const ref = this.db.collection('market_offers').doc(offerId);

    const offer = {
      offerId,
      active: true,
      supplierTenantId,
      supplierName: dto.supplierName ?? 'Fornecedor',
      supplierLocation: { lat: dto.supplierLat, lng: dto.supplierLng
},
      serviceRadiusKm: dto.serviceRadiusKm ?? 120,
      regionLabel: dto.regionLabel ?? null,

      catalogItemId: dto.catalogItemId,
      title: dto.title,
      uom: dto.uom,
      price: dto.price,
      currency: dto.currency ?? 'BRL',

      source: { supplierItemId: dto.supplierItemId, aggregateMode:
'TOTAL' },

      visibility: dto.visibility ?? 'PUBLIC',
```

```
      allowedBuyerTenants: [],

      createdBy: { uid: actor.uid },
      createdAt: Date.now(),
      updatedAt: Date.now()
    };

    await ref.create(offer);

    // cria espelho inicial de estoque
    await this.sync.syncOfferStock(offerId);

    return offer;
  }

  async updateOffer(supplierTenantId: string, actor: any, offerId:
string, dto: any) {
    const ref = this.db.collection('market_offers').doc(offerId);
    const snap = await ref.get();
    if (!snap.exists) throw new Error('OFFER_NOT_FOUND');
    const offer = snap.data()!;
    if (offer.supplierTenantId !== supplierTenantId) throw new
Error('TENANT_MISMATCH');

    const patch: any = { updatedAt: Date.now() };
    if (dto.title !== undefined) patch.title = dto.title;
    if (dto.uom !== undefined) patch.uom = dto.uom;
    if (dto.price !== undefined) patch.price = dto.price;
    if (dto.active !== undefined) patch.active = dto.active;
    if (dto.serviceRadiusKm !== undefined) patch.serviceRadiusKm =
dto.serviceRadiusKm;
    if (dto.regionLabel !== undefined) patch.regionLabel =
dto.regionLabel;
    if (dto.supplierLat !== undefined || dto.supplierLng !==
undefined) {
      patch.supplierLocation = {
        lat: dto.supplierLat ?? offer.supplierLocation?.lat,
        lng: dto.supplierLng ?? offer.supplierLocation?.lng,
      };
    }

    await ref.set(patch, { merge: true });

    // se mudou algo relevante, re-sincroniza
    await this.sync.syncOfferStock(offerId);

    return { offerId, ...offer, ...patch };
  }

  async listNear(buyerTenantId: string, params: { lat:number;
lng:number; radiusKm?:number; maxRadiusKm?:number; limit?:number;
expand?:boolean }) {
    const origin = { lat: params.lat, lng: params.lng };
    let radius = Math.max(1, params.radiusKm ?? 50);
    const maxRadius = Math.max(radius, params.maxRadiusKm ?? 800);
    const limit = Math.min(Math.max(params.limit ?? 50, 1), 100);
    const expand = params.expand !== false;

    const snap = await this.db.collection('market_offers')
      .where('active', '==', true)
      .get();
```

```
    const offers = snap.docs.map(d => d.data());

    let attempts = 0;
    let expanded = false;

    while (true) {
      attempts += 1;

      const filtered = offers
        .map((o:any) => {
          // visibilidade (MVP: PUBLIC)
          if (o.visibility === 'PRIVATE') {
            const allowed = Array.isArray(o.allowedBuyerTenants) ?
o.allowedBuyerTenants : [];
            if (!allowed.includes(buyerTenantId)) return null;
          }

          const supplierLoc = o.supplierLocation;
          if (!supplierLoc?.lat || !supplierLoc?.lng) return null;

          const distanceKm = haversineKm(origin, supplierLoc);
          const serviceRadiusKm = Number(o.serviceRadiusKm ?? 0) || 0;

          // regra: tem que estar dentro do raio efetivo e também
dentro do raio de serviço do fornecedor
          const inUserRadius = distanceKm <= radius;
          const inServiceRadius = serviceRadiusKm > 0 ? distanceKm <=
serviceRadiusKm : true;

          if (!inUserRadius || !inServiceRadius) return null;
          return { ...o, distanceKm };
        })
        .filter(Boolean)
        .sort((a:any, b:any) => a.distanceKm - b.distanceKm)
        .slice(0, limit);

      if (filtered.length > 0) {
        // batch stock
        const ids = filtered.map((o:any) => o.offerId);
        const stockSnaps = await Promise.all(ids.map((id:string) =>
this.db.collection('market_offer_stock').doc(id).get()));
        const stockMap = new Map<string, any>();
        stockSnaps.forEach(s => { if (s.exists) stockMap.set(s.id,
s.data()); });

        const items = filtered.map((o:any) => {
          const st = stockMap.get(o.offerId);
          return {
            offerId: o.offerId,
            title: o.title,
            price: o.price,
            uom: o.uom,
            supplierName: o.supplierName,
            distanceKm: Number(o.distanceKm.toFixed(1)),
            qtyAvailable: st?.qtyAvailable ?? 0,
            updatedAt: st?.updatedAt ?? null,
            regionLabel: o.regionLabel ?? null
          };
        });
```

```
        return { items, usedRadiusKm: radius, expanded, attempts };
      }

      if (!expand || radius >= maxRadius || attempts >= 6) {
        return { items: [], usedRadiusKm: radius, expanded, attempts
};
      }

      radius = Math.min(maxRadius, radius * 2);
      expanded = true;
    }
  }

  async getOfferNear(buyerTenantId: string, offerId: string, origin:
{lat:number; lng:number}) {
    const ref = this.db.collection('market_offers').doc(offerId);
    const snap = await ref.get();
    if (!snap.exists) throw new Error('OFFER_NOT_FOUND');
    const o = snap.data()!;
    if (!o.active) throw new Error('OFFER_INACTIVE');

    // visibilidade
    if (o.visibility === 'PRIVATE') {
      const allowed = Array.isArray(o.allowedBuyerTenants) ?
o.allowedBuyerTenants : [];
      if (!allowed.includes(buyerTenantId)) throw new
Error('OFFER_FORBIDDEN');
    }

    const supplierLoc = o.supplierLocation;
    const distanceKm = haversineKm(origin, supplierLoc);
    const serviceRadiusKm = Number(o.serviceRadiusKm ?? 0) || 0;
    if (serviceRadiusKm > 0) assertInRange(distanceKm,
serviceRadiusKm);

    const stSnap = await
this.db.collection('market_offer_stock').doc(offerId).get();
    const st = stSnap.exists ? stSnap.data()! : { qtyAvailable: 0 };

    return {
      offerId: o.offerId,
      title: o.title,
      price: o.price,
      uom: o.uom,
      supplierName: o.supplierName,
      distanceKm: Number(distanceKm.toFixed(1)),
      qtyAvailable: st.qtyAvailable ?? 0,
      updatedAt: st.updatedAt ?? null,
      regionLabel: o.regionLabel ?? null
    };
  }

  async getOfferStockNear(buyerTenantId: string, offerId: string,
origin: {lat:number; lng:number}) {
    // valida range via offer (não expõe stock se fora do raio)
    await this.getOfferNear(buyerTenantId, offerId, origin);
    const stSnap = await
this.db.collection('market_offer_stock').doc(offerId).get();
    const st = stSnap.exists ? stSnap.data()! : null;
    return { offerId, qtyAvailable: st?.qtyAvailable ?? 0, updatedAt:
st?.updatedAt ?? null };
```

```
  }
}
```

---

5) Controller

```
// src/market/market.controller.ts
import { Body, Controller, Get, Param, Patch, Post, Query, UseGuards }
from '@nestjs/common';
import { ApiBearerAuth, ApiTags, ApiOperation } from
'@nestjs/swagger';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { MarketOffersService } from './market-offers.service';
import { CreateOfferDto, MarketListDto, UpdateOfferDto } from
'./dto/market.dto';

@ApiTags('Market')
@ApiBearerAuth('bearerAuth')
@Controller('/api/market')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class MarketController {
  constructor(private readonly market: MarketOffersService) {}

  @Get('/offers')
  @ApiOperation({ summary: 'List offers near buyer (adaptive radius)'
})

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDITOR',
'SUPPLIER','LOGISTICS','INVESTOR')
  async list(@Actor() actor: any, @Query() q: MarketListDto) {
    return this.market.listNear(actor.tenantId, {
      lat: q.lat, lng: q.lng,
      radiusKm: q.radiusKm, maxRadiusKm: q.maxRadiusKm,
      limit: q.limit, expand: q.expand
    });
  }

  @Get('/offers/:id')
  @ApiOperation({ summary: 'Get offer detail (must be within service
radius)' })

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDITOR',
'SUPPLIER','LOGISTICS','INVESTOR')
  async get(@Actor() actor: any, @Param('id') id: string,
@Query('lat') lat: string, @Query('lng') lng: string) {
    return this.market.getOfferNear(actor.tenantId, id, { lat:
Number(lat), lng: Number(lng) });
  }

  @Get('/offers/:id/stock')
  @ApiOperation({ summary: 'Get offer stock (real-time mirror), range-
protected' })
```

```
@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDITOR',
'SUPPLIER','LOGISTICS','INVESTOR')
  async stock(@Actor() actor: any, @Param('id') id: string,
@Query('lat') lat: string, @Query('lng') lng: string) {
    return this.market.getOfferStockNear(actor.tenantId, id, { lat:
Number(lat), lng: Number(lng) });
  }

  @Post('/offers')
  @ApiOperation({ summary: 'Supplier creates offer' })
  @Roles('SUPPLIER','MANAGER','ADMIN')
  @Permissions('catalog.read')
  async createOffer(@Actor() actor: any, @Body() dto: CreateOfferDto)
{
    // oferta pertence ao tenant do fornecedor (actor.tenantId)
    return this.market.createOffer(actor.tenantId, actor, dto);
  }

  @Patch('/offers/:id')
  @ApiOperation({ summary: 'Supplier updates offer' })
  @Roles('SUPPLIER','MANAGER','ADMIN')
  @Permissions('catalog.read')
  async updateOffer(@Actor() actor: any, @Param('id') id: string,
@Body() dto: UpdateOfferDto) {
    return this.market.updateOffer(actor.tenantId, actor, id, dto);
  }
}
```

---

6) MarketModule

```
// src/market/market.module.ts
import { Module } from '@nestjs/common';
import { firestoreProvider } from '../common/firestore.provider';
import { MarketController } from './market.controller';
import { MarketOffersService } from './market-offers.service';
import { MarketSyncService } from './market-sync.service';

@Module({
  providers: [firestoreProvider, MarketOffersService,
MarketSyncService],
  controllers: [MarketController],
  exports: [MarketSyncService],
})
export class MarketModule {}
```

Adicione no `AppModule`.

---

7) Firestore Rules (Marketplace só por API)

```
match /market_offers/{id} { allow read, write: if false; }
match /market_offer_stock/{id} { allow read, write: if false; }
```

---

8) Índices Firestore

- `market_offers`:
  - `active == true` (Firestore já ajuda; se você adicionar filtros futuros, cria índices)
- `market_offer_stock`:
  - doc direto por offerId (sem índice)

---

9) Integrar sincronização no estoque (para ficar "real-time" de verdade)

No ponto onde você confirma um movimento (ex.: `DraftsService.applyDraft`), após atualizar `stock_balances` do fornecedor, chame:

```
await this.marketSync.syncOffersForSupplierItem(tenantId, itemId);
```

Para isso:

- Importe `MarketModule` no `DraftsModule` e injete `MarketSyncService`.

Exemplo (DraftsModule):

```
imports: [AuditModule, PoliciesModule, MarketModule]
```

No DraftsService constructor:

```
constructor(..., private readonly marketSync: MarketSyncService) {}
```

E no final da transação (idealmente fora do tx, para não travar):

- Você pode coletar `supplierTenantId` e `itemId` e chamar depois do commit. MVP simples: chamar após `runTransaction` terminar.

---

10) App (Expo): hook marketplace com expansão automática

```
// src/api/market.ts
import { api } from './client';

export async function listMarketOffers(lat: number, lng: number) {
  const res = await api.get('/api/market/offers', {
    params: { lat, lng, radiusKm: 50, maxRadiusKm: 800, expand: true,
limit: 50 }
  });
  return res.data;
}

export async function getOffer(offerId: string, lat: number, lng:
number) {
```

```
  const res = await api.get(`/api/market/offers/${offerId}`, { params:
{ lat, lng } });
  return res.data;
}

export async function getOfferStock(offerId: string, lat: number, lng:
number) {
  const res = await api.get(`/api/market/offers/${offerId}/stock`, {
params: { lat, lng } });
  return res.data;
}
```

Resultado

✅ Se não houver empresa perto, a **busca expande** automaticamente até encontrar.
✅ O comprador vê o **estoque total** do fornecedor "em tempo real" (espelhado e atualizado sempre que estoque muda).
✅ Sem violar multi-tenant (ninguém lê estoque direto do fornecedor).

Se você quiser, eu também adiciono **"marketEnabled" por local** (para o fornecedor escolher quais depósitos entram no marketplace) + filtro no `MarketSyncService` para somar só esses locais.

Sim — vou colocar os dois pontos:

1. **`marketEnabled` por local** (o fornecedor escolhe quais depósitos entram no marketplace; o estoque total da oferta soma só esses locais).
2. **Marketplace B2C com acesso restrito**: **somente clientes cadastrados e ativos** conseguem acessar qualquer endpoint do marketplace.

Abaixo estão os patches completos (NestJS + Firestore Rules).

---

1) `marketEnabled` nos locais (Locations)

1.1 Atualize o DTO de Location

```
// src/locations/dto/locations.dto.ts
import { IsBoolean, IsNotEmpty, IsOptional, IsString } from 'class-
validator';

export class CreateLocationDto {
  @IsString() @IsNotEmpty() name: string;
  @IsOptional() @IsString() type?: 'FARM' | 'WAREHOUSE' | 'STORE' |
'YARD' | 'SILO';

  @IsOptional() @IsString() carCode?: string;
  @IsOptional() @IsString() stateRegistration?: string;

  @IsOptional() geofence?: { lat: number; lng: number; radiusM: number
};
```

```
  // ✅ novo
  @IsOptional() @IsBoolean() marketEnabled?: boolean; // default false
}

export class UpdateLocationDto {
  @IsOptional() @IsString() name?: string;
  @IsOptional() @IsString() type?: 'FARM' | 'WAREHOUSE' | 'STORE' |
'YARD' | 'SILO';

  @IsOptional() @IsString() carCode?: string;
  @IsOptional() @IsString() stateRegistration?: string;

  @IsOptional() geofence?: { lat: number; lng: number; radiusM: number
};

  // ✅ novo
  @IsOptional() @IsBoolean() marketEnabled?: boolean;
}
```

1.2 Atualize `LocationsService` (create + update)

```
// src/locations/locations.service.ts
async create(tenantId: string, actor: any, dto: any) {
  // ...
  const doc = {
    // ...
    marketEnabled: dto.marketEnabled ?? false, // ✅
    // ...
  };
  await ref.create(doc);
  return doc;
}

async update(tenantId: string, actor: any, locationId: string, dto:
any) {
  const ref = this.db.collection('locations').doc(locationId);
  const snap = await ref.get();
  if (!snap.exists) throw new Error('LOCATION_NOT_FOUND');
  const cur = snap.data()!;
  if (cur.tenantId !== tenantId) throw new Error('TENANT_MISMATCH');

  const patch: any = { updatedAt: Date.now() };
  if (dto.name !== undefined) patch.name = dto.name;
  if (dto.type !== undefined) patch.type = dto.type;
  if (dto.carCode !== undefined) patch.carCode = dto.carCode;
  if (dto.stateRegistration !== undefined) patch.stateRegistration =
dto.stateRegistration;
  if (dto.geofence !== undefined) patch.geofence = dto.geofence;
  if (dto.marketEnabled !== undefined) patch.marketEnabled =
dto.marketEnabled; // ✅

  await ref.set(patch, { merge: true });
  return { ...cur, ...patch };
}
```

1.3 Endpoint para atualizar location

```
// src/locations/locations.controller.ts
import { Patch } from '@nestjs/common';
```

```
import { UpdateLocationDto } from './dto/locations.dto';

@Patch(':id')
@Roles('ADMIN','MANAGER')
@Permissions('inventory.write')
async update(@Actor() actor: any, @Param('id') id: string, @Body()
dto: UpdateLocationDto) {
  return this.locations.update(actor.tenantId, actor, id, dto);
}
```

---

2) MarketSyncService somando apenas locais `marketEnabled=true`

2.1 Atualize o `MarketSyncService.syncOfferStock`

Ele vai:

1. buscar locations do fornecedor com `marketEnabled == true`
2. somar `stock_balances` apenas desses `locationId`

```
// src/market/market-sync.service.ts
async syncOfferStock(offerId: string) {
  const offerRef = this.db.collection('market_offers').doc(offerId);
  const offerSnap = await offerRef.get();
  if (!offerSnap.exists) throw new Error('OFFER_NOT_FOUND');

  const offer = offerSnap.data()!;
  if (!offer.active) {
    await this.db.collection('market_offer_stock').doc(offerId).set({
      offerId, qtyAvailable: 0, updatedAt: Date.now(), lastSyncReason:
'OFFER_INACTIVE'
    }, { merge: true });
    return;
  }

  const supplierTenantId = offer.supplierTenantId;
  const supplierItemId = offer.source?.supplierItemId;

  // ✅ 1) pegar locais habilitados para marketplace
  const locSnap = await this.db.collection('locations')
    .where('tenantId', '==', supplierTenantId)
    .where('active', '==', true)
    .where('marketEnabled', '==', true)
    .get();

  const enabledLocationIds = locSnap.docs.map(d => d.get('locationId')
|| d.id);
  if (enabledLocationIds.length === 0) {
    await this.db.collection('market_offer_stock').doc(offerId).set({
      offerId, qtyAvailable: 0, updatedAt: Date.now(), lastSyncReason:
'NO_MARKET_LOCATIONS'
    }, { merge: true });
    return;
  }

  // ✅ 2) somar balances apenas desses locais
  // Firestore "in" tem limite de 10 valores; então fazemos em
batches.
```

```
  const chunks: string[][] = [];
  for (let i = 0; i < enabledLocationIds.length; i += 10)
chunks.push(enabledLocationIds.slice(i, i + 10));

  let total = 0;
  for (const batch of chunks) {
    const balSnap = await this.db.collection('stock_balances')
      .where('tenantId', '==', supplierTenantId)
      .where('itemId', '==', supplierItemId)
      .where('locationId', 'in', batch)
      .get();

    balSnap.forEach(d => { total += Number(d.get('qtyOnHand') || 0);
});
  }

  await this.db.collection('market_offer_stock').doc(offerId).set({
    offerId,
    qtyAvailable: total,
    updatedAt: Date.now(),
    lastSyncReason: 'STOCK_BALANCES_AGG_MARKET_LOC'
  }, { merge: true });
}
```

✅ Agora o fornecedor controla o que entra no marketplace só marcando `marketEnabled`.

---

3) Marketplace B2C: somente clientes cadastrados e ativos

A forma mais sólida é colocar uma **regra de acesso no backend** (porque o marketplace já é somente por API).

3.1 Modelo no Firestore: `tenants/{tenantId}`

Adicione este bloco no doc do tenant (cadastro do cliente):

```
"marketplace": {
  "enabled": true,
  "tier": "B2C",
  "status": "ACTIVE"
}
```

Regras:

- `enabled=true`
- `tier='B2C'` (você pode ter B2B depois)
- `status='ACTIVE'` (suspenso/bloqueado não entra)

3.2 Guard de acesso ao marketplace

```
// src/market/guards/market-access.guard.ts
import { CanActivate, ExecutionContext, Injectable, Inject } from
'@nestjs/common';
import { FIRESTORE } from '../../common/firestore.provider';
```

```
@Injectable()
export class MarketAccessGuard implements CanActivate {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async canActivate(ctx: ExecutionContext) {
    const req = ctx.switchToHttp().getRequest();
    const actor = req.user; // seu FirebaseAuthGuard coloca aqui
    const tenantId = actor?.tenantId;

    if (!tenantId) throw new Error('TENANT_REQUIRED');

    const snap = await
this.db.collection('tenants').doc(tenantId).get();
    if (!snap.exists) throw new Error('TENANT_NOT_REGISTERED');

    const t = snap.data()!;
    const m = t.marketplace || {};

    if (m.enabled !== true) throw new Error('MARKETPLACE_DISABLED');
    if ((m.tier || 'B2C') !== 'B2C') throw new
Error('MARKETPLACE_TIER_NOT_ALLOWED');
    if ((m.status || 'ACTIVE') !== 'ACTIVE') throw new
Error('MARKETPLACE_NOT_ACTIVE');

    // opcional: restringir roles que podem acessar marketplace
    // ex.: PRODUCER/OPERATOR/TECHNICIAN/SUPPLIER/MANAGER/ADMIN
    const allowedRoles = new
Set(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','SUPPLIER'])
;
    if (!allowedRoles.has(actor.role)) throw new
Error('MARKET_ROLE_FORBIDDEN');

    return true;
  }
}
```

3.3 Aplique o guard em TODOS endpoints do marketplace

```
// src/market/market.controller.ts
import { MarketAccessGuard } from './guards/market-access.guard';

@Controller('/api/market')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard,
MarketAccessGuard) // ✅ add
export class MarketController { /* ... */ }
```

✅ Resultado: **mesmo autenticado**, se não tiver **tenant registrado e ativo**, não acessa marketplace.

---

4) Seed/Processo de “cliente cadastrado”

Para cadastrar um novo cliente (tenant), você pode:

- criar endpoint ADMIN `POST /api/admin/tenants`

- ou criar via painel interno

Doc mínimo:

```
{
  "tenantId": "t_xxx",
  "name": "Fazenda ABC",
  "marketplace": { "enabled": true, "tier": "B2C", "status": "ACTIVE"
},
  "createdAt": 1730000000000
}
```

---

5) Índices Firestore adicionais

Para o sync com `locations`:

- `locations`: `tenantId + active + marketEnabled`

Para `stock_balances` com filtro `locationId in` + itemId:

- `tenantId + itemId + locationId + updatedAt` (Firestore pode pedir; crie quando solicitar)

---

6) Regras Firestore (Locations update pelo backend apenas ou permitido?)

Você tinha "update false" antes. Para permitir **ativar marketEnabled** pelo painel/admin via app, você tem 2 opções:

Opção mais segura (recomendada)

- **Update de locations só via backend** (painel chama API)
- Firestore Rules mantém `update: false`

✅ Mantém lastro e controle total.

Se quiser permitir update pelo app direto, dá para permitir update somente `marketEnabled` e `name`, mas eu recomendo **API**.

---

7) O que isso garante

- Fornecedor escolhe exatamente quais depósitos entram no marketplace ✅
- Estoque mostrado é total "publicável" (soma desses depósitos) ✅
- Comprador só acessa marketplace se for cliente cadastrado e ativo ✅
- Proximidade + expansão de raio continua funcionando ✅

Se você quiser, eu também posso te mandar:

- endpoint `GET /api/market/companies` (lista empresas próximas, sem listar ofertas)
- endpoint `POST /api/market/offers/:id/reserve` (pré-reserva/escrow — fase seguinte)

A seguir estão os dois complementos que você pediu, já no padrão completo:

1. **`GET /api/market/companies`** → lista **empresas próximas** que usam o sistema (cadastradas e ativas no marketplace B2C), com **raio adaptativo** igual às ofertas.
2. **`POST /api/market/offers/:id/reserve`** → **pré-reserva** com "hold" de quantidade + expiração (base para escrow). (MVP seguro; não liquida pagamento ainda.)

Tudo continua protegido por `MarketAccessGuard` (somente clientes cadastrados e ativos).

---

1) Empresas próximas: `GET /api/market/companies`

1.1 Modelo de dados (empresas)

Reaproveitamos `market_offers` como fonte, mas agrupamos por fornecedor. Para ficar bem, sugiro também ter:

- `supplierTenantId` e `supplierName`
- `supplierLocation` (ponto da empresa)
- `serviceRadiusKm`

Já existe na offer.

1.2 DTO

```
// src/market/dto/market-companies.dto.ts
import { IsBoolean, IsInt, IsNumber, IsOptional, Max, Min } from
'class-validator';
import { Type } from 'class-transformer';

export class MarketCompaniesDto {
  @Type(() => Number) @IsNumber() lat: number;
  @Type(() => Number) @IsNumber() lng: number;

  @IsOptional() @Type(() => Number) @IsInt() @Min(1) @Max(800)
  radiusKm?: number = 50;

  @IsOptional() @Type(() => Number) @IsInt() @Min(1) @Max(1200)
  maxRadiusKm?: number = 800;

  @IsOptional() @Type(() => Number) @IsInt() @Min(1) @Max(200)
  limit?: number = 50;

  @IsOptional() @IsBoolean()
  expand?: boolean = true;
}
```

1.3 Service (agrupamento + raio adaptativo)

```
// src/market/market-companies.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { haversineKm } from './market.utils';

@Injectable()
export class MarketCompaniesService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async listCompaniesNear(params: { lat:number; lng:number;
radiusKm?:number; maxRadiusKm?:number; limit?:number; expand?:boolean
}) {
    const origin = { lat: params.lat, lng: params.lng };
    let radius = Math.max(1, params.radiusKm ?? 50);
    const maxRadius = Math.max(radius, params.maxRadiusKm ?? 800);
    const limit = Math.min(Math.max(params.limit ?? 50, 1), 200);
    const expand = params.expand !== false;

    // pega offers ativas
    const snap = await this.db.collection('market_offers')
      .where('active', '==', true)
      .get();

    const offers = snap.docs.map(d => d.data());

    let attempts = 0;
    let expanded = false;

    while (true) {
      attempts += 1;

      // mapa por empresa (supplierTenantId)
      const map = new Map<string, any>();

      for (const o of offers) {
        const supplierLoc = o.supplierLocation;
        if (!supplierLoc?.lat || !supplierLoc?.lng) continue;

        const distanceKm = haversineKm(origin, supplierLoc);
        const serviceRadiusKm = Number(o.serviceRadiusKm ?? 0) || 0;

        // dentro do raio do usuário e do raio do fornecedor
        if (distanceKm > radius) continue;
        if (serviceRadiusKm > 0 && distanceKm > serviceRadiusKm)
continue;

        const key = o.supplierTenantId;
        const cur = map.get(key);

        if (!cur) {
          map.set(key, {
            supplierTenantId: o.supplierTenantId,
            supplierName: o.supplierName,
            regionLabel: o.regionLabel ?? null,
            distanceKm,
            offersCount: 1
          });
        } else {
          // mantém menor distância e soma offers
```

```
          cur.distanceKm = Math.min(cur.distanceKm, distanceKm);
          cur.offersCount += 1;
        }
      }

      const companies = Array.from(map.values())
        .sort((a:any, b:any) => a.distanceKm - b.distanceKm)
        .slice(0, limit)
        .map((c:any) => ({
          supplierTenantId: c.supplierTenantId,
          supplierName: c.supplierName,
          distanceKm: Number(c.distanceKm.toFixed(1)),
          offersCount: c.offersCount,
          regionLabel: c.regionLabel
        }));

      if (companies.length > 0) return { items: companies,
usedRadiusKm: radius, expanded, attempts };

      if (!expand || radius >= maxRadius || attempts >= 6) {
        return { items: [], usedRadiusKm: radius, expanded, attempts
};
      }

      radius = Math.min(maxRadius, radius * 2);
      expanded = true;
    }
  }
}
```

1.4 Controller

```
// src/market/market.controller.ts (adicionar endpoint)
import { MarketCompaniesService } from './market-companies.service';
import { MarketCompaniesDto } from './dto/market-companies.dto';

// no constructor:
constructor(
  private readonly market: MarketOffersService,
  private readonly companies: MarketCompaniesService
) {}

@Get('/companies')
async companiesNear(@Query() q: MarketCompaniesDto) {
  return this.companies.listCompaniesNear({
    lat: q.lat, lng: q.lng,
    radiusKm: q.radiusKm, maxRadiusKm: q.maxRadiusKm,
    limit: q.limit, expand: q.expand
  });
}
```

1.5 MarketModule (registrar service)

```
// src/market/market.module.ts
import { MarketCompaniesService } from './market-companies.service';

@Module({
  providers: [firestoreProvider, MarketOffersService,
MarketSyncService, MarketCompaniesService],
  controllers: [MarketController],
  exports: [MarketSyncService],
})
export class MarketModule {}
```

2) Pré-reserva: `POST /api/market/offers/:id/reserve`

Objetivo: quando comprador quer garantir quantidade, cria um "hold" com expiração. Depois, o fluxo de contrato/escrow utiliza esse hold.

2.1 Coleção `market_reservations/{reservationId}`

```
{
  "reservationId": "res_...",
  "buyerTenantId": "t_buyer_01",
  "buyerUid": "u_...",
  "offerId": "off_123",
  "qty": 100,
  "status": "HELD",
  "expiresAt": 1730000600000,
  "createdAt": 1730000000000
}
```

2.2 Atualizar `market_offer_stock` para suportar `qtyReserved`

```
{
  "offerId": "off_123",
  "qtyAvailable": 980,
  "qtyReserved": 120,
  "updatedAt": 1730000000000
}
```

Disponível real = **`qtyAvailable - qtyReserved`** (nunca negativo).

2.3 DTO

```
// src/market/dto/market-reserve.dto.ts
import { IsInt, IsNumber, IsOptional, Max, Min } from 'class-
validator';
import { Type } from 'class-transformer';

export class ReserveOfferDto {
  @Type(() => Number) @IsNumber() lat: number;
  @Type(() => Number) @IsNumber() lng: number;

  @Type(() => Number) @IsInt() @Min(1) @Max(1_000_000)
  qty: number;

  @IsOptional() @Type(() => Number) @IsInt() @Min(5) @Max(240)
  holdMinutes?: number = 30; // expira sozinho
}
```

2.4 Service (transação segura)

```
// src/market/market-reserve.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';
import { MarketOffersService } from './market-offers.service';

@Injectable()
export class MarketReserveService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly offers: MarketOffersService
  ) {}
```

```
  async reserve(buyerTenantId: string, actor: any, offerId: string,
dto: any) {
    // valida proximidade e visibilidade
    await this.offers.getOfferNear(buyerTenantId, offerId, { lat:
dto.lat, lng: dto.lng });

    const reservationId = makeId('res');
    const resRef =
this.db.collection('market_reservations').doc(reservationId);
    const stockRef =
this.db.collection('market_offer_stock').doc(offerId);

    const holdMs = (dto.holdMinutes ?? 30) * 60_000;
    const now = Date.now();
    const expiresAt = now + holdMs;

    await this.db.runTransaction(async (tx) => {
      const stSnap = await tx.get(stockRef);
      const st = stSnap.exists ? stSnap.data()! : { qtyAvailable: 0,
qtyReserved: 0 };

      const qtyAvailable = Number(st.qtyAvailable || 0);
      const qtyReserved = Number(st.qtyReserved || 0);
      const free = qtyAvailable - qtyReserved;

      if (dto.qty > free) throw new
Error('INSUFFICIENT_MARKET_STOCK');

      tx.set(stockRef, {
        offerId,
        qtyReserved: qtyReserved + dto.qty,
        updatedAt: now,
        lastSyncReason: 'RESERVE_HOLD'
      }, { merge: true });

      tx.create(resRef, {
        reservationId,
        buyerTenantId,
        buyerUid: actor.uid,
        offerId,
        qty: dto.qty,
        status: 'HELD',
        expiresAt,
        createdAt: now
      });
    });

    return { reservationId, offerId, qty: dto.qty, status: 'HELD',
expiresAt };
  }

  async releaseExpired() {
    // job periódico (a cada 1-5 min)
    const now = Date.now();
    const snap = await this.db.collection('market_reservations')
      .where('status', '==', 'HELD')
      .where('expiresAt', '<=', now)
      .get();

    for (const doc of snap.docs) {
```

```
      const r = doc.data()!;
      const stockRef =
this.db.collection('market_offer_stock').doc(r.offerId);
      await this.db.runTransaction(async (tx) => {
        const stSnap = await tx.get(stockRef);
        if (!stSnap.exists) return;

        const st = stSnap.data()!;
        const qtyReserved = Number(st.qtyReserved || 0);
        const newReserved = Math.max(0, qtyReserved - Number(r.qty ||
0));

        tx.set(stockRef, { qtyReserved: newReserved, updatedAt:
Date.now(), lastSyncReason: 'RESERVE_EXPIRE' }, { merge: true });
        tx.set(doc.ref, { status: 'EXPIRED' }, { merge: true });
      });
    }

    return { expired: snap.size };
  }
}
```

2.5 Controller

```
// src/market/market.controller.ts (add)
import { MarketReserveService } from './market-reserve.service';
import { ReserveOfferDto } from './dto/market-reserve.dto';

// constructor add:
constructor(
  private readonly market: MarketOffersService,
  private readonly companies: MarketCompaniesService,
  private readonly reserveSvc: MarketReserveService
) {}

@Post('/offers/:id/reserve')
@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN')
async reserve(@Actor() actor: any, @Param('id') id: string, @Body()
dto: ReserveOfferDto) {
  return this.reserveSvc.reserve(actor.tenantId, actor, id, dto);
}
```

2.6 Module

```
// src/market/market.module.ts
import { MarketReserveService } from './market-reserve.service';

@Module({
  providers: [firestoreProvider, MarketOffersService,
MarketSyncService, MarketCompaniesService, MarketReserveService],
  controllers: [MarketController],
  exports: [MarketSyncService],
})
export class MarketModule {}
```

---

3) Segurança B2C “somente cadastrados”

Você já tem o `MarketAccessGuard`. Para ficar perfeito:

- No cadastro do tenant (`tenants/{tenantId}`):
  - `marketplace.enabled=true`

    - `marketplace.status='ACTIVE'`
    - `marketplace.tier='B2C'`
- O guard bloqueia todo marketplace.

✅ Isso garante que “somente clientes cadastrados” entram.

---

4) Regras Firestore (Marketplace fechado)

```
match /market_offers/{id} { allow read, write: if false; }
match /market_offer_stock/{id} { allow read, write: if false; }
match /market_reservations/{id} { allow read, write: if false; }
```

Tudo via API.

---

5) App (Expo) — endpoints novos

5.1 Companies near me

```
export async function listMarketCompanies(lat:number, lng:number) {
  const res = await api.get('/api/market/companies', { params: { lat,
lng, radiusKm: 50, maxRadiusKm: 800, expand: true, limit: 50 } });
  return res.data;
}
```

5.2 Reserve

```
export async function reserveOffer(offerId:string, lat:number,
lng:number, qty:number) {
  const res = await api.post(`/api/market/offers/${offerId}/reserve`,
{ lat, lng, qty, holdMinutes: 30, clientRequestId: cryptoRandomId()
});
  return res.data;
}
```

Se quiser idempotência no reserve, implemente `clientRequestId` no backend também (recomendado).

---

6) Job para expirar reservas

Você pode:

- rodar no Nest com `@nestjs/schedule` (a cada 1-2 min)
- ou Cloud Scheduler + endpoint admin
- ou Cloud Functions

Exemplo Nest (se usar schedule):

```
@Cron('*/2 * * * *')
async handleExpired() {
  await this.reserveSvc.releaseExpired();
}
```

Quer que eu inclua agora a “expiração automática” no NestJS (ScheduleModule) + idempotência em `reserve`?

Eu recomendo muito para ficar realmente “perfeito”.

Beleza — abaixo vai o pacote completo para ficar “perfeito”:

1. **ScheduleModule + cron** para expirar reservas automaticamente
2. **Idempotência real no `reserve`** (clientRequestId)
3. Ajustes para evitar corrida/duplicidade
4. Resposta consistente para o app

---

1) Habilitar agendamento no NestJS

1.1 Instalar

```
npm i @nestjs/schedule
```

1.2 AppModule

```
// src/app.module.ts
import { Module } from '@nestjs/common';
import { ScheduleModule } from '@nestjs/schedule';
// ... seus módulos

@Module({
  imports: [
    ScheduleModule.forRoot(), // ✅
    // ... outros módulos
  ],
})
export class AppModule {}
```

---

2) Idempotência padrão (reutilizável)

Crie um helper simples para gravar/ler idempotência.

2.1 Utilitário

```
// src/common/idempotency/idempotency.util.ts
export function idemKey(tenantId: string, uid: string,
clientRequestId: string, scope: string) {
  return `${tenantId}:${uid}:${scope}:${clientRequestId}`;
}
```

2.2 Coleção

Use `idempotency/{key}`:

```
{
  "tenantId": "t_buyer",
  "uid": "u_...",
  "scope": "market.reserve",
  "clientRequestId": "uuid",
  "createdAt": 1730000000000,
  "result": { ... }
}
```

---

3) ReserveOfferDto com `clientRequestId` (MVP perfeito)

```
// src/market/dto/market-reserve.dto.ts
import { IsInt, IsNotEmpty, IsNumber, IsOptional, IsString, Max, Min }
from 'class-validator';
import { Type } from 'class-transformer';

export class ReserveOfferDto {
  @Type(() => Number) @IsNumber() lat: number;
  @Type(() => Number) @IsNumber() lng: number;

  @Type(() => Number) @IsInt() @Min(1) @Max(1_000_000)
  qty: number;

  @IsOptional() @Type(() => Number) @IsInt() @Min(5) @Max(240)
  holdMinutes?: number = 30;

  @IsString() @IsNotEmpty()
  clientRequestId: string; // ✅ idempotência
}
```

---

4) MarketReserveService com idempotência + expiração automática

4.1 Reserve idempotente (sem duplicar hold)

```
// src/market/market-reserve.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';
import { MarketOffersService } from './market-offers.service';
import { idemKey } from '../common/idempotency/idempotency.util';

@Injectable()
export class MarketReserveService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly offers: MarketOffersService
  ) {}

  async reserve(buyerTenantId: string, actor: any, offerId: string,
dto: any) {
    // valida range e visibilidade
    await this.offers.getOfferNear(buyerTenantId, offerId, { lat:
dto.lat, lng: dto.lng });

    const now = Date.now();
    const holdMs = (dto.holdMinutes ?? 30) * 60_000;
    const expiresAt = now + holdMs;
```

```
    const key = idemKey(buyerTenantId, actor.uid, dto.clientRequestId,
'market.reserve');
    const idemRef = this.db.collection('idempotency').doc(key);

    const reservationId = makeId('res');
    const resRef =
this.db.collection('market_reservations').doc(reservationId);
    const stockRef =
this.db.collection('market_offer_stock').doc(offerId);

    const result = await this.db.runTransaction(async (tx) => {
      // 1) se já existe idempotência, devolve o resultado gravado
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      // 2) carrega stock e verifica disponibilidade real
      const stSnap = await tx.get(stockRef);
      const st = stSnap.exists ? stSnap.data()! : { qtyAvailable: 0,
qtyReserved: 0 };

      const qtyAvailable = Number(st.qtyAvailable || 0);
      const qtyReserved = Number(st.qtyReserved || 0);
      const free = qtyAvailable - qtyReserved;

      if (dto.qty > free) throw new
Error('INSUFFICIENT_MARKET_STOCK');

      // 3) incrementa reserva
      tx.set(stockRef, {
        offerId,
        qtyReserved: qtyReserved + dto.qty,
        updatedAt: now,
        lastSyncReason: 'RESERVE_HOLD'
      }, { merge: true });

      // 4) cria reserva
      const reservation = {
        reservationId,
        buyerTenantId,
        buyerUid: actor.uid,
        offerId,
        qty: dto.qty,
        status: 'HELD',
        expiresAt,
        createdAt: now
      };
      tx.create(resRef, reservation);

      // 5) grava idempotência com resultado
      const out = { reservationId, offerId, qty: dto.qty, status:
'HELD', expiresAt };
      tx.create(idemRef, {
        tenantId: buyerTenantId,
        uid: actor.uid,
        scope: 'market.reserve',
        clientRequestId: dto.clientRequestId,
        createdAt: now,
        result: out
      });
```

```
      return out;
    });

    return result;
  }

  async releaseExpired(limit = 200) {
    const now = Date.now();

    // pega um lote (para não estourar tempo em grandes volumes)
    const snap = await this.db.collection('market_reservations')
      .where('status', '==', 'HELD')
      .where('expiresAt', '<=', now)
      .limit(limit)
      .get();

    let expired = 0;

    for (const doc of snap.docs) {
      const r = doc.data()!;
      const stockRef =
this.db.collection('market_offer_stock').doc(r.offerId);

      await this.db.runTransaction(async (tx) => {
        const rSnap = await tx.get(doc.ref);
        if (!rSnap.exists) return;
        const cur = rSnap.data()!;
        if (cur.status !== 'HELD') return;

        const stSnap = await tx.get(stockRef);
        if (!stSnap.exists) {
          tx.set(doc.ref, { status: 'EXPIRED', updatedAt: Date.now()
}, { merge: true });
          return;
        }

        const st = stSnap.data()!;
        const qtyReserved = Number(st.qtyReserved || 0);
        const newReserved = Math.max(0, qtyReserved - Number(cur.qty
|| 0));

        tx.set(stockRef, { qtyReserved: newReserved, updatedAt:
Date.now(), lastSyncReason: 'RESERVE_EXPIRE' }, { merge: true });
        tx.set(doc.ref, { status: 'EXPIRED', updatedAt: Date.now() },
{ merge: true });
      });

      expired += 1;
    }

    return { expired };
  }
}
```

---

5) Cron automático (a cada 2 minutos)

```
// src/market/market-reserve.cron.ts
import { Injectable, Logger } from '@nestjs/common';
import { Cron } from '@nestjs/schedule';
import { MarketReserveService } from './market-reserve.service';
```

```
@Injectable()
export class MarketReserveCron {
  private readonly logger = new Logger(MarketReserveCron.name);

  constructor(private readonly reserve: MarketReserveService) {}

  @Cron('*/2 * * * *') // ✅ a cada 2 min
  async expireHolds() {
    try {
      const r = await this.reserve.releaseExpired(200);
      if (r.expired > 0) this.logger.log(`Expired holds:
${r.expired}`);
    } catch (e: any) {
      this.logger.error(`Expire holds failed: ${e?.message || e}`);
    }
  }
}
```

Registrar no MarketModule

```
// src/market/market.module.ts
import { MarketReserveCron } from './market-reserve.cron';

@Module({
  providers: [
    firestoreProvider,
    MarketOffersService,
    MarketSyncService,
    MarketCompaniesService,
    MarketReserveService,
    MarketReserveCron, // ✅
  ],
  controllers: [MarketController],
  exports: [MarketSyncService],
})
export class MarketModule {}
```

---

6) Controller: reserve com clientRequestId obrigatório

```
// src/market/market.controller.ts (reserve endpoint)
@Post('/offers/:id/reserve')
@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN')
async reserve(@Actor() actor: any, @Param('id') id: string, @Body()
dto: ReserveOfferDto) {
  return this.reserveSvc.reserve(actor.tenantId, actor, id, dto);
}
```

---

7) App (Expo): gerar `clientRequestId` e repetir sem duplicar

Use `uuid` e passe no payload. Se a rede cair, você reenvia com o mesmo `clientRequestId` e o backend devolve o mesmo resultado.

```
import { v4 as uuid } from 'uuid';

export async function reserveOffer(offerId: string, lat: number, lng:
number, qty: number) {
  const clientRequestId = uuid();
```

```
  const res = await api.post(`/api/market/offers/${offerId}/reserve`,
{
    lat, lng, qty,
    holdMinutes: 30,
    clientRequestId
  });
  return res.data;
}
```

---

8) “Perfeito” mesmo: ajuste de resposta em erro

Se não tiver estoque:

- backend lança `INSUFFICIENT_MARKET_STOCK`
- seu `DomainExceptionFilter` mapeia para `409 CONFLICT`
- app mostra: “Estoque insuficiente para reserva”.

Checklist final do marketplace (com o que você pediu)

✅ `marketEnabled` por location (fornecedor controla o que publica)
✅ Somatório total por oferta **apenas dos locais habilitados**
✅ Acesso B2C apenas para clientes cadastrados e ativos (MarketAccessGuard)
✅ Reserva com hold + expiração automática (cron)
✅ Idempotência (sem duplicar reserva)

Se você quiser, eu também posso incluir:

- `cancel reservation` (libera antes de expirar)
- `confirm reservation` (transforma em pedido/escrow)

Perfeito — abaixo vão os dois endpoints que faltam para fechar o ciclo da reserva:

1. **Cancelar reserva** (libera quantidade imediatamente)
2. **Confirmar reserva** (transforma hold em “reservado-confirmado” e gera um `market_order` inicial — base para escrow/contrato)

Tudo com:

- idempotência (clientRequestId)
- transação segura (não libera duas vezes)
- validação de dono (somente quem criou pode cancelar/confirmar, ou MANAGER/ADMIN do mesmo tenant)

---

1) DTOs

```
// src/market/dto/market-reservation-actions.dto.ts
```

```
import { IsNotEmpty, IsOptional, IsString } from 'class-validator';

export class CancelReservationDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsOptional() @IsString()
  reason?: string;
}

export class ConfirmReservationDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsOptional() @IsString()
  note?: string;
}
```

---

2) Modelo de `market_orders` (MVP)

Coleção: `market_orders/{orderId}`

```
{
  "orderId": "ord_...",
  "buyerTenantId": "t_buyer",
  "buyerUid": "u_...",
  "offerId": "off_123",
  "reservationId": "res_...",
  "qty": 100,
  "status": "PENDING_ESCROW",
  "createdAt": 1730000000000
}
```

Depois você liga isso ao módulo de contrato/escrow do Ciclo+.

---

3) Service: cancelar e confirmar (com idempotência)

```
// src/market/market-reservation-actions.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';
import { idemKey } from '../common/idempotency/idempotency.util';

@Injectable()
export class MarketReservationActionsService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  private canManage(actor: any, r: any) {
    // dono da reserva ou manager/admin do mesmo tenant
    const isOwner = r.buyerUid === actor.uid && r.buyerTenantId ===
actor.tenantId;
    const isAdmin = ['ADMIN','MANAGER'].includes(actor.role) &&
r.buyerTenantId === actor.tenantId;
    return isOwner || isAdmin;
```

```
  }

  async cancelReservation(actor: any, reservationId: string, dto: {
clientRequestId: string; reason?: string }) {
    const now = Date.now();
    const key = idemKey(actor.tenantId, actor.uid,
dto.clientRequestId, 'market.reserve.cancel');
    const idemRef = this.db.collection('idempotency').doc(key);

    const resRef =
this.db.collection('market_reservations').doc(reservationId);

    return this.db.runTransaction(async (tx) => {
      // idempotência
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const rSnap = await tx.get(resRef);
      if (!rSnap.exists) throw new Error('RESERVATION_NOT_FOUND');
      const r = rSnap.data()!;

      if (!this.canManage(actor, r)) throw new
Error('RESERVATION_FORBIDDEN');

      // se já expirou/cancelou/confirmou, devolve estado
(idempotente)
      if (r.status !== 'HELD') {
        const out = { reservationId, status: r.status, alreadyFinal:
true };
        tx.create(idemRef, { tenantId: actor.tenantId, uid: actor.uid,
scope: 'market.reserve.cancel', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });
        return out;
      }

      // se já passou da expiração, marca expirado (sem mexer
qtyReserved aqui; cron também faz, mas aqui resolve)
      if (Number(r.expiresAt || 0) <= now) {
        tx.set(resRef, { status: 'EXPIRED', updatedAt: now }, { merge:
true });
        const out = { reservationId, status: 'EXPIRED' };
        tx.create(idemRef, { tenantId: actor.tenantId, uid: actor.uid,
scope: 'market.reserve.cancel', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });
        return out;
      }

      // libera estoque reservado
      const stockRef =
this.db.collection('market_offer_stock').doc(r.offerId);
      const stSnap = await tx.get(stockRef);
      if (stSnap.exists) {
        const st = stSnap.data()!;
        const qtyReserved = Number(st.qtyReserved || 0);
        const newReserved = Math.max(0, qtyReserved - Number(r.qty ||
0));
        tx.set(stockRef, { qtyReserved: newReserved, updatedAt: now,
lastSyncReason: 'RESERVE_CANCEL' }, { merge: true });
      }

      tx.set(resRef, {
```

```
        status: 'CANCELLED',
        cancelReason: dto.reason ?? null,
        cancelledBy: { uid: actor.uid, role: actor.role },
        cancelledAt: now,
        updatedAt: now,
      }, { merge: true });

      const out = { reservationId, status: 'CANCELLED' };
      tx.create(idemRef, { tenantId: actor.tenantId, uid: actor.uid,
scope: 'market.reserve.cancel', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });
      return out;
    });
  }

  async confirmReservation(actor: any, reservationId: string, dto: {
clientRequestId: string; note?: string }) {
    const now = Date.now();
    const key = idemKey(actor.tenantId, actor.uid,
dto.clientRequestId, 'market.reserve.confirm');
    const idemRef = this.db.collection('idempotency').doc(key);

    const resRef =
this.db.collection('market_reservations').doc(reservationId);

    return this.db.runTransaction(async (tx) => {
      // idempotência
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const rSnap = await tx.get(resRef);
      if (!rSnap.exists) throw new Error('RESERVATION_NOT_FOUND');
      const r = rSnap.data()!;

      if (!this.canManage(actor, r)) throw new
Error('RESERVATION_FORBIDDEN');

      // se já confirmou, retorna orderId
      if (r.status === 'CONFIRMED') {
        const out = { reservationId, status: 'CONFIRMED', orderId:
r.orderId ?? null };
        tx.create(idemRef, { tenantId: actor.tenantId, uid: actor.uid,
scope: 'market.reserve.confirm', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });
        return out;
      }

      // só pode confirmar se HELD e não expirou
      if (r.status !== 'HELD') throw new
Error('RESERVATION_NOT_HELD');
      if (Number(r.expiresAt || 0) <= now) throw new
Error('RESERVATION_EXPIRED');

      // cria order
      const orderId = makeId('ord');
      const orderRef =
this.db.collection('market_orders').doc(orderId);

      tx.create(orderRef, {
        orderId,
        buyerTenantId: r.buyerTenantId,
```

```
        buyerUid: r.buyerUid,
        offerId: r.offerId,
        reservationId: r.reservationId,
        qty: r.qty,
        status: 'PENDING_ESCROW',
        note: dto.note ?? null,
        createdAt: now,
      });

      // marca reserva confirmada
      tx.set(resRef, {
        status: 'CONFIRMED',
        confirmedAt: now,
        confirmedBy: { uid: actor.uid, role: actor.role },
        orderId,
        updatedAt: now,
      }, { merge: true });

      // IMPORTANTE: não decrementa qtyReserved aqui — ela continua
reservada até finalizar escrow/contrato.
      // Quando o pedido virar contrato/escrow, você "consome" o
reservado em uma etapa posterior.

      const out = { reservationId, status: 'CONFIRMED', orderId };
      tx.create(idemRef, { tenantId: actor.tenantId, uid: actor.uid,
scope: 'market.reserve.confirm', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });
      return out;
    });
  }
}
```

---

4) Controller: rotas de cancel/confirm

```
// src/market/market.controller.ts (adicionar)
import { MarketReservationActionsService } from './market-reservation-
actions.service';
import { CancelReservationDto, ConfirmReservationDto } from
'./dto/market-reservation-actions.dto';

// constructor add:
constructor(
  private readonly market: MarketOffersService,
  private readonly companies: MarketCompaniesService,
  private readonly reserveSvc: MarketReserveService,
  private readonly reservationActions: MarketReservationActionsService
) {}

@Post('/reservations/:id/cancel')
@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN')
async cancel(@Actor() actor: any, @Param('id') id: string, @Body()
dto: CancelReservationDto) {
  return this.reservationActions.cancelReservation(actor, id, dto);
}

@Post('/reservations/:id/confirm')
@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN')
async confirm(@Actor() actor: any, @Param('id') id: string, @Body()
dto: ConfirmReservationDto) {
  return this.reservationActions.confirmReservation(actor, id, dto);
```

```
}
```

---

5) MarketModule: registrar o serviço

```
// src/market/market.module.ts
import { MarketReservationActionsService } from './market-reservation-
actions.service';

@Module({
  providers: [
    firestoreProvider,
    MarketOffersService,
    MarketSyncService,
    MarketCompaniesService,
    MarketReserveService,
    MarketReserveCron,
    MarketReservationActionsService, // ✅
  ],
  controllers: [MarketController],
  exports: [MarketSyncService],
})
export class MarketModule {}
```

---

6) Firestore Rules (ordens e reservas só via API)

```
match /market_reservations/{id} { allow read, write: if false; }
match /market_orders/{id} { allow read, write: if false; }
```

---

7) App (Expo): endpoints cancel/confirm

```
import { v4 as uuid } from 'uuid';
import { api } from './client';

export async function cancelReservation(reservationId: string,
reason?: string) {
  const res = await
api.post(`/api/market/reservations/${reservationId}/cancel`, {
    clientRequestId: uuid(),
    reason: reason ?? null
  });
  return res.data;
}

export async function confirmReservation(reservationId: string, note?:
string) {
  const res = await
api.post(`/api/market/reservations/${reservationId}/confirm`, {
    clientRequestId: uuid(),
    note: note ?? null
  });
  return res.data;
}
```

Como isso fecha o Ciclo+ no agro (B2C)

- **Reserve** segura quantidade (hold) ✅
- **Cron** expira automaticamente ✅
- **Cancel** libera instantâneo ✅
- **Confirm** gera `order` e segura reservado para escrow/contrato ✅

Se você quiser, eu posso agora fechar o próximo passo (fase escrow):

- `POST /api/market/orders/:id/escrow/start`
- vincular Asaas
- split e liberação por evidência/marcos (Ciclo+ core)

Sim — dá para fazer, e a melhor forma (segura e "jurídica") é mudar o fluxo para:

✅ **NÃO reservar estoque no "reserve"**
✅ Só "travar" (reservar de verdade) **após pagamento do escrow** (webhook Asaas confirmado)
✅ **Cancelamento com multa** (regras claras + cobrança automática via Asaas)

Abaixo vai o desenho completo + endpoints + estados + campos.

---

1) Novo fluxo de estados

1.1 Reservation (intenção, sem travar estoque)

- `reservation.status`:
  - `INTENT_CREATED` (criada)
  - `ESCROW_PENDING` (boleto/pix/link criado)
  - `ESCROW_PAID` (pagamento confirmado)
  - `HELD` (estoque travado **após pagamento**)
  - `CONFIRMED` (gerou order/contrato)
  - `CANCELLED`
  - `EXPIRED`

Mudança principal:
➡️ Antes: `reserve` já incrementava `qtyReserved`.
➡️ Agora: `reserve` só cria intenção e inicia escrow. Quem trava estoque é o webhook `ESCROW_PAID`.

---

2) Modelo de dados atualizado

2.1 `market_offer_stock/{offerId}`

```
{
  "offerId": "off_123",
  "qtyAvailable": 980,
```

```
  "qtyReserved": 0,
  "updatedAt": 1730000000000
}
```

2.2 `market_reservations/{reservationId}`

```
{
  "reservationId": "res_...",
  "buyerTenantId": "t_buyer",
  "buyerUid": "u_...",
  "offerId": "off_123",
  "qty": 100,

  "status": "ESCROW_PENDING",

  "escrow": {
    "provider": "ASAAS",
    "chargeId": "pay_...",
    "invoiceUrl": "...",
    "status": "PENDING",
    "amount": 12050.00,
    "currency": "BRL",
    "createdAt": 1730000000000,
    "paidAt": null
  },

  "penaltyPolicy": {
    "type": "PERCENT",
    "value": 2.5,
    "minValue": 15.00,
    "afterPaidPercent": 5.0,
    "afterPaidMinValue": 30.00
  },

  "expiresAt": 1730000600000,
  "createdAt": 1730000000000,
  "updatedAt": 1730000000000
}
```

2.3 `market_orders/{orderId}` (após confirm)

```
{
  "orderId": "ord_...",
  "reservationId": "res_...",
  "offerId": "off_123",
  "buyerTenantId": "t_buyer",
  "qty": 100,
  "status": "PENDING_CONTRACT_OR_FULFILLMENT",
  "createdAt": 1730000000000
}
```

---

3) Endpoints novos/alterados

3.1 Criar intenção + gerar escrow

POST `/api/market/offers/:id/reserve`

Entrada

```
{
  "lat": -15.6,
```

```
  "lng": -56.1,
  "qty": 100,
  "holdMinutes": 30,
  "clientRequestId": "uuid"
}
```

Saída

```
{
  "reservationId": "res_...",
  "status": "ESCROW_PENDING",
  "expiresAt": 1730000600000,
  "escrow": { "invoiceUrl": "...", "chargeId": "pay_..." }
}
```

✅ Aqui NÃO trava estoque.
Só cria reserva-intenção e cria cobrança escrow via Asaas.

3.2 Webhook Asaas (confirma pagamento)

POST `/api/webhooks/asaas`

Quando evento for "payment confirmed / received":

- localiza reservation pelo `chargeId`
- valida não expirou e status correto
- **trava estoque**: incrementa `qtyReserved` se houver `free >= qty`
- muda reservation: `ESCROW_PAID` → `HELD`

Se não houver estoque no momento do pagamento:

- você tem duas opções (recomendo a primeira):
  1. **Reembolsar automaticamente** (refund) e marcar `FAILED_OUT_OF_STOCK`
  2. Colocar em "WAITLIST" com prazo curto e autorização explícita do comprador

---

3.3 Cancelamento com multa

POST `/api/market/reservations/:id/cancel`

Regras:

- Se `status = ESCROW_PENDING`: cancelamento sem multa (ou multa fixa pequena, opcional)
- Se `status = ESCROW_PAID` ou `HELD` ou `CONFIRMED`: **multa obrigatória**
  - calcula multa

    - cria cobrança de multa (ou desconta do valor do escrow e devolve o restante)
    - libera estoque travado se estava `HELD`
    - muda status para `CANCELLED`

3.4 Confirmar (gera order/contrato)

POST `/api/market/reservations/:id/confirm`

Pré-condição:

- `status` tem que ser `HELD` (ou seja: escrow pago + estoque travado)

Ação:

- cria `market_order`
- muda reservation → `CONFIRMED`

---

4) Multa no cancelamento — política simples e forte

Política recomendada (bem prática)

- Antes de pagar (ESCROW_PENDING): multa = **0**
- Depois de pagar:
    - multa = **max(afterPaidMinValue, amount * afterPaidPercent/100)**
    - exemplo: 5% com mínimo 30 reais

Isso resolve o "calote" e pune desistência depois de bloquear o fluxo.

---

5) Implementação NestJS — pontos essenciais

5.1 Reserve agora só cria escrow (não mexe no estoque)

No `MarketReserveService.reserve(...)`:

- remove o trecho que altera `market_offer_stock.qtyReserved`
- cria reservation com `status: ESCROW_PENDING`
- chama `AsaasService.createEscrowCharge(...)`
- grava `escrow.chargeId` e `invoiceUrl`

Pseudocódigo do Reserve

```
// dentro da transaction
tx.create(resRef, {
  reservationId,
```

```
  buyerTenantId,
  buyerUid: actor.uid,
  offerId,
  qty: dto.qty,
  status: 'ESCROW_PENDING',
  escrow: { provider:'ASAAS', chargeId, invoiceUrl, status:'PENDING',
amount, createdAt: now },
  penaltyPolicy: {...},
  expiresAt,
  createdAt: now,
  updatedAt: now
});
```

✅ Idempotência continua igual.

5.2 Webhook: travar estoque após pagamento

Pseudocódigo do handler

```
// on Asaas payment confirmed:
await db.runTransaction(async (tx) => {
  const res = await findReservationByChargeId(tx, chargeId);
  if (!res) return;

  if (res.status !== 'ESCROW_PENDING') return; // idempotente
  if (Date.now() > res.expiresAt) {
     // reembolsar e expirar
  }

  const stockRef =
db.collection('market_offer_stock').doc(res.offerId);
  const st = (await tx.get(stockRef)).data() || { qtyAvailable:0,
qtyReserved:0 };
  const free = st.qtyAvailable - st.qtyReserved;

  if (free < res.qty) {
     // refund + marcar FAIL_OUT_OF_STOCK
  } else {
     tx.set(stockRef, { qtyReserved: st.qtyReserved + res.qty,
updatedAt: now, lastSyncReason:'ESCROW_PAID_HOLD' }, {merge:true});
     tx.set(res.ref, { status:'HELD', 'escrow.status':'PAID',
'escrow.paidAt': now, updatedAt: now }, {merge:true});
  }
});
```

5.3 Cancelamento com multa (desconto/refund)

Pseudocódigo

```
if (res.status === 'ESCROW_PENDING') {
   // cancel without penalty, cancel charge on Asaas
}
else {
   penalty = max(minValue, amount*percent)
   // opção A (melhor): reembolsar escrow - penalty (refund parcial)
```

```
    // opção B: cobrar penalty separado e reembolsar total escrow
    // libera estoque se HELD
}
```

Opção A (mais limpa): refund parcial do escrow e fica tudo "dentro" do pagamento. Se Asaas não suportar parcial em certos métodos, usa Opção B (cobrança separada).

---

6) Endpoints de Escrow (Asaas) — o mínimo que precisa

`AsaasService`

- `createCharge(...)` (PIX/boleto/cartão)
- `cancelCharge(chargeId)` (se não pago)
- `refundCharge(chargeId, amount)` (total ou parcial)
- valida assinatura do webhook (token)

Você já usa Asaas em billing/webhooks, então reaproveita o mesmo módulo.

---

7) Segurança: "só reserva após escrow pago"

Com esse desenho:

- `reserve` **não mexe no estoque**
- só o webhook pago (servidor) mexe em `qtyReserved`
- Firestore Rules continuam bloqueando write pelo cliente
  ✅ garantia total.

---

8) O que eu faço na próxima etapa (seu "Sim" já autorizou)

Para fechar 100%, eu te mando o código completo de:

1. `AsaasEscrowService` (interfaces + stubs + onde plugar)
2. `WebhookController` com roteamento `payment_confirmed`
3. `reserve` atualizado
4. `cancel` com multa (refund parcial ou charge de multa)
5. `confirm` exigindo `HELD`
6. permissões:
   - `market.reserve`
   - `market.cancel`
   - `market.confirm`
   - `webhook.asaas` (interno)

Só uma escolha técnica (vou assumir a melhor se você não responder)

Multa será aplicada como:

- **A) desconto do valor do escrow (refund parcial)** ✅ recomendado
- **B) cobrança separada de multa**

Se você não falar nada, vou seguir com **A** (e cair para B se método não permitir parcial).

Fechado: **Opção A = multa por desconto direto do escrow (refund parcial)**, com **fallback automático** para cobrança separada se o método não permitir refund parcial.

Abaixo vai o pacote "pronto" (NestJS) com:

- `POST /api/market/offers/:id/reserve` → cria **INTENÇÃO + cobrança escrow** (NÃO trava estoque)
- `POST /api/webhooks/asaas` → quando **pago**, trava estoque (`qtyReserved += qty`) e muda reservation → `HELD`
- `POST /api/market/reservations/:id/cancel` → se pago, aplica **multa** e faz **refund parcial** (ou fallback)
- `POST /api/market/reservations/:id/confirm` → só se `HELD`
- cron expira `ESCROW_PENDING` (cancela cobrança) e expira `HELD` (libera estoque + aplica política se quiser)

Importante: código abaixo assume que você já tem módulo Asaas (billing/webhooks). Se não tiver, eu deixei stubs claros para plugar.

---

0) Novas permissões sugeridas (use seu RBAC)

- `market.reserve`
- `market.cancel`
- `market.confirm`
- `webhook.asaas` (interno)

---

1) DTO Reserve (intenção + escrow)

```
// src/market/dto/market-reserve.dto.ts
import { IsInt, IsNotEmpty, IsNumber, IsOptional, IsString, Max, Min }
from 'class-validator';
import { Type } from 'class-transformer';

export class ReserveOfferDto {
  @Type(() => Number) @IsNumber() lat: number;
  @Type(() => Number) @IsNumber() lng: number;

  @Type(() => Number) @IsInt() @Min(1) @Max(1_000_000)
  qty: number;

  @IsOptional() @Type(() => Number) @IsInt() @Min(5) @Max(240)
```

```
  holdMinutes?: number = 30;

  @IsString() @IsNotEmpty()
  clientRequestId: string;

  // opcional: preferencia método
  @IsOptional() @IsString()
  payMethod?: 'PIX' | 'BOLETO' | 'CREDIT_CARD';
}
```

---

2) AsaasEscrowService (stub com contrato mínimo)

```
// src/integrations/asaas/asaas-escrow.service.ts
import { Injectable } from '@nestjs/common';

export type AsaasCharge = {
  chargeId: string;
  invoiceUrl?: string;
  pixQrCode?: string;
  amount: number;
  status: 'PENDING'|'RECEIVED'|'CONFIRMED'|'CANCELLED'|'REFUNDED';
};

@Injectable()
export class AsaasEscrowService {
  // ✅ criar cobrança (escrow)
  async createCharge(input: {
    buyerTenantId: string;
    reservationId: string;
    description: string;
    amount: number;
    method: 'PIX'|'BOLETO'|'CREDIT_CARD';
    // dados do cliente (CPF/CNPJ, nome, etc.) viriam do seu
tenant/profile
  }): Promise<AsaasCharge> {
    // TODO: implementar com API Asaas (create payment)
    // retornar chargeId + invoiceUrl/pix payload
    return {
      chargeId: `pay_${input.reservationId}`,
      invoiceUrl: 'https://asaas.example/invoice',
      amount: input.amount,
      status: 'PENDING'
    };
  }

  // ✅ cancelar cobrança (se ainda não paga)
  async cancelCharge(chargeId: string): Promise<void> {
    // TODO: Asaas cancel
  }

  // ✅ refund TOTAL ou PARCIAL (Opção A)
  async refundCharge(chargeId: string, amount: number): Promise<void>
{
    // TODO: Asaas refund/chargeback (parcial se suportado)
    // Se Asaas não suportar parcial para o método, lance erro com
code/type para fallback
  }

  // ✅ fallback: cobrar multa separada
```

```
  async chargePenalty(input: {
    buyerTenantId: string;
    reservationId: string;
    amount: number;
    description: string;
  }): Promise<AsaasCharge> {
    // TODO: create charge de multa
    return { chargeId: `pen_${input.reservationId}`, amount:
input.amount, status: 'PENDING' };
  }

  // ✅ webhook verify (assinatura / token)
  verifyWebhook(headers: any, body: any): boolean {
    // TODO: validar assinatura/token do Asaas
    return true;
  }
}
```

---

3) MarketReserveService (cria INTENÇÃO + escrow; sem travar estoque)

```
// src/market/market-reserve.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';
import { idemKey } from '../common/idempotency/idempotency.util';
import { MarketOffersService } from './market-offers.service';
import { AsaasEscrowService } from '../integrations/asaas/asaas-
escrow.service';

function calcAmount(price: number, qty: number) {
  // MVP: price * qty
  // Pode incluir frete/impostos depois
  return Math.round(price * qty * 100) / 100;
}

function penaltyPolicyDefault() {
  return {
    type: 'PERCENT',
    value: 2.5,
    minValue: 15.0,
    afterPaidPercent: 5.0,
    afterPaidMinValue: 30.0
  };
}

@Injectable()
export class MarketReserveService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly offers: MarketOffersService,
    private readonly asaas: AsaasEscrowService
  ) {}

  async reserve(buyerTenantId: string, actor: any, offerId: string,
dto: any) {
    // valida range + visibilidade
```

```
    const offer = await this.offers.getOfferNear(buyerTenantId,
offerId, { lat: dto.lat, lng: dto.lng });

    const now = Date.now();
    const expiresAt = now + (dto.holdMinutes ?? 30) * 60_000;

    const scope = 'market.reserve.intent';
    const key = idemKey(buyerTenantId, actor.uid, dto.clientRequestId,
scope);
    const idemRef = this.db.collection('idempotency').doc(key);

    const reservationId = makeId('res');
    const resRef =
this.db.collection('market_reservations').doc(reservationId);

    // calcula valor do escrow
    const amount = calcAmount(Number(offer.price || 0), Number(dto.qty
|| 0));
    const method = (dto.payMethod ?? 'PIX') as
'PIX'|'BOLETO'|'CREDIT_CARD';

    // 1) idempotência + criação doc
    const created = await this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const reservation = {
        reservationId,
        buyerTenantId,
        buyerUid: actor.uid,
        offerId,
        qty: dto.qty,
        status: 'ESCROW_PENDING',
        escrow: {
          provider: 'ASAAS',
          chargeId: null,
          invoiceUrl: null,
          status: 'PENDING',
          amount,
          currency: 'BRL',
          createdAt: now,
          paidAt: null
        },
        penaltyPolicy: penaltyPolicyDefault(),
        expiresAt,
        createdAt: now,
        updatedAt: now
      };

      tx.create(resRef, reservation);

      const out = { reservationId, status: 'ESCROW_PENDING',
expiresAt, amount };
      tx.create(idemRef, {
        tenantId: buyerTenantId,
        uid: actor.uid,
        scope,
        clientRequestId: dto.clientRequestId,
        createdAt: now,
        result: out
      });
```

```
      return out;
    });

    // 2) cria cobrança no Asaas (fora da tx)
    // se der erro aqui, você pode marcar reservation como
ESCROW_FAILED
    const charge = await this.asaas.createCharge({
      buyerTenantId,
      reservationId,
      description: `Reserva marketplace ${offer.title} (${dto.qty}
${offer.uom})`,
      amount,
      method
    });

    await resRef.set({
      status: 'ESCROW_PENDING',
      escrow: {
        provider: 'ASAAS',
        chargeId: charge.chargeId,
        invoiceUrl: charge.invoiceUrl ?? null,
        pixQrCode: charge.pixQrCode ?? null,
        status: 'PENDING',
        amount: charge.amount,
        currency: 'BRL',
        createdAt: now,
        paidAt: null
      },
      updatedAt: Date.now()
    }, { merge: true });

    return {
      ...created,
      escrow: { chargeId: charge.chargeId, invoiceUrl:
charge.invoiceUrl ?? null, pixQrCode: charge.pixQrCode ?? null }
    };
  }
}
```

---

4) Webhook Asaas: após pago → trava estoque e muda reservation para HELD

```
// src/integrations/asaas/asaas-webhook.controller.ts
import { Body, Controller, Headers, Post } from '@nestjs/common';
import { Inject } from '@nestjs/common';
import { FIRESTORE } from '../../common/firestore.provider';
import { AsaasEscrowService } from './asaas-escrow.service';

@Controller('/api/webhooks/asaas')
export class AsaasWebhookController {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly asaas: AsaasEscrowService
  ) {}

  @Post()
  async handle(@Headers() headers: any, @Body() body: any) {
```

```
    if (!this.asaas.verifyWebhook(headers, body)) throw new
Error('WEBHOOK_INVALID');

    // Ajuste conforme payload do Asaas:
    const event = body?.event; // ex: PAYMENT_CONFIRMED /
PAYMENT_RECEIVED
    const chargeId = body?.payment?.id || body?.payment?.payment?.id;

    if (!chargeId) return { ok: true };

    const isPaid =
['PAYMENT_CONFIRMED','PAYMENT_RECEIVED','RECEIVED','CONFIRMED'].includ
es(event);
    if (!isPaid) return { ok: true };

    // acha reservation por escrow.chargeId
    const snap = await this.db.collection('market_reservations')
      .where('escrow.chargeId', '==', chargeId)
      .limit(1)
      .get();

    if (snap.empty) return { ok: true };

    const doc = snap.docs[0];
    const now = Date.now();

    await this.db.runTransaction(async (tx) => {
      const rSnap = await tx.get(doc.ref);
      if (!rSnap.exists) return;
      const r = rSnap.data()!;

      // idempotência do webhook
      if (r.status === 'HELD' || r.status === 'CONFIRMED') return;

      // expirada? reembolsar e marcar
      if (Number(r.expiresAt || 0) <= now) {
        tx.set(doc.ref, { status: 'EXPIRED', updatedAt: now,
'escrow.status': 'PAID_LATE' }, { merge: true });
        return;
      }

      // trava estoque AGORA (após pagamento)
      const stockRef =
this.db.collection('market_offer_stock').doc(r.offerId);
      const stSnap = await tx.get(stockRef);
      const st = stSnap.exists ? stSnap.data()! : { qtyAvailable: 0,
qtyReserved: 0 };

      const qtyAvailable = Number(st.qtyAvailable || 0);
      const qtyReserved = Number(st.qtyReserved || 0);
      const free = qtyAvailable - qtyReserved;

      if (Number(r.qty || 0) > free) {
        // sem estoque: marca falha (você pode acionar refund fora da
tx)
        tx.set(doc.ref, { status: 'FAILED_OUT_OF_STOCK', updatedAt:
now, 'escrow.status': 'PAID_NO_STOCK' }, { merge: true });
        return;
      }

      tx.set(stockRef, {
```

```
        offerId: r.offerId,
        qtyReserved: qtyReserved + Number(r.qty || 0),
        updatedAt: now,
        lastSyncReason: 'ESCROW_PAID_HOLD'
      }, { merge: true });

      tx.set(doc.ref, {
        status: 'HELD',
        updatedAt: now,
        escrow: { ...r.escrow, status: 'PAID', paidAt: now }
      }, { merge: true });
    });

    // Se status virou FAILED_OUT_OF_STOCK, aqui fora você faz refund
total (opcional).
    return { ok: true };
  }
}
```

---

5) Cancelamento com multa (Opção A: refund parcial)

Regra:

- `ESCROW_PENDING` → cancela cobrança, sem multa
- `HELD/CONFIRMED/ESCROW_PAID` → multa:
  - `penalty = max(afterPaidMinValue, amount * afterPaidPercent/100)`
  - Refund = `amount - penalty` (se refund parcial possível)
  - Se não for possível refund parcial → refund total + cobrar multa separada (fallback)

```
// src/market/market-cancel.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { AsaasEscrowService } from '../integrations/asaas/asaas-
escrow.service';

function computePenalty(r: any) {
  const pp = r.penaltyPolicy || {};
  const amount = Number(r.escrow?.amount || 0);
  const pct = Number(pp.afterPaidPercent ?? 5.0);
  const minV = Number(pp.afterPaidMinValue ?? 30.0);
  const calc = (amount * pct) / 100;
  return Math.round(Math.max(minV, calc) * 100) / 100;
}

@Injectable()
export class MarketCancelService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly asaas: AsaasEscrowService
  ) {}

  async cancel(actor: any, reservationId: string, dto: {
clientRequestId: string; reason?: string }) {
```

```
    const now = Date.now();
    const key = idemKey(actor.tenantId, actor.uid,
dto.clientRequestId, 'market.cancel');
    const idemRef = this.db.collection('idempotency').doc(key);

    const resRef =
this.db.collection('market_reservations').doc(reservationId);

    // 1) transação: marcar cancelamento e (se HELD) liberar
qtyReserved
    const { escrowChargeId, escrowAmount, doPenalty, penaltyAmount,
offerId, qty } =
      await this.db.runTransaction(async (tx) => {
        const idemSnap = await tx.get(idemRef);
        if (idemSnap.exists) return idemSnap.data()!.result;

        const rSnap = await tx.get(resRef);
        if (!rSnap.exists) throw new Error('RESERVATION_NOT_FOUND');
        const r = rSnap.data()!;

        // somente dono ou admin/manager do tenant
        const isOwner = r.buyerUid === actor.uid && r.buyerTenantId
=== actor.tenantId;
        const isAdmin = ['ADMIN','MANAGER'].includes(actor.role) &&
r.buyerTenantId === actor.tenantId;
        if (!isOwner && !isAdmin) throw new
Error('RESERVATION_FORBIDDEN');

        if (['CANCELLED','EXPIRED'].includes(r.status)) {
          const out = { alreadyFinal: true, status: r.status };
          tx.create(idemRef, { tenantId: actor.tenantId, uid:
actor.uid, scope: 'market.cancel', clientRequestId:
dto.clientRequestId, createdAt: now, result: out });
          return out;
        }

        const escrowChargeId = r.escrow?.chargeId;
        const escrowAmount = Number(r.escrow?.amount || 0);

        const paidLike =
['HELD','CONFIRMED','ESCROW_PAID'].includes(r.status) ||
r.escrow?.status === 'PAID';
        const doPenalty = paidLike;
        const penaltyAmount = doPenalty ? computePenalty(r) : 0;

        // libera estoque se estava HELD/CONFIRMED (travado)
        if (['HELD','CONFIRMED'].includes(r.status)) {
          const stockRef =
this.db.collection('market_offer_stock').doc(r.offerId);
          const stSnap = await tx.get(stockRef);
          if (stSnap.exists) {
            const st = stSnap.data()!;
            const qtyReserved = Number(st.qtyReserved || 0);
            tx.set(stockRef, { qtyReserved: Math.max(0, qtyReserved -
Number(r.qty || 0)), updatedAt: now, lastSyncReason: 'CANCEL_RELEASE'
}, { merge: true });
          }
        }

        tx.set(resRef, {
          status: 'CANCELLED',
```

```
          cancelReason: dto.reason ?? null,
          cancelledAt: now,
          cancelledBy: { uid: actor.uid, role: actor.role },
          updatedAt: now
        }, { merge: true });

        const out = { escrowChargeId, escrowAmount, doPenalty,
penaltyAmount, offerId: r.offerId, qty: r.qty, status: 'CANCELLED' };
        tx.create(idemRef, { tenantId: actor.tenantId, uid: actor.uid,
scope: 'market.cancel', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });
        return out;
      });

    // 2) operações Asaas fora da transação
    if (!escrowChargeId) return { ok: true, status: 'CANCELLED' };

    if (!doPenalty) {
      // não pago: cancelar charge
      await this.asaas.cancelCharge(escrowChargeId);
      return { ok: true, status: 'CANCELLED', penaltyApplied: false };
    }

    // pago: aplicar multa via refund parcial (A)
    const refundAmount = Math.max(0, Math.round((escrowAmount -
penaltyAmount) * 100) / 100);

    try {
      // tenta refund parcial
      if (refundAmount > 0) await
this.asaas.refundCharge(escrowChargeId, refundAmount);
      // se refundAmount==0: fica tudo como multa (não reembolsa nada)
      return { ok: true, status: 'CANCELLED', penaltyApplied: true,
penaltyAmount, refundAmount, mode: 'PARTIAL_REFUND' };
    } catch (e) {
      // fallback: refund total + cobrar multa separada
      try {
        await this.asaas.refundCharge(escrowChargeId, escrowAmount);
      } catch {}
      await this.asaas.chargePenalty({
        buyerTenantId: actor.tenantId,
        reservationId,
        amount: penaltyAmount,
        description: `Multa cancelamento reserva ${reservationId}`
      });

      return { ok: true, status: 'CANCELLED', penaltyApplied: true,
penaltyAmount, refundAmount: escrowAmount, mode:
'REFUND_TOTAL_AND_PENALTY_CHARGE' };
    }
  }
}
```

---

6) Confirmar reserva (só se HELD)

```
// src/market/market-confirm.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';
import { idemKey } from '../common/idempotency/idempotency.util';
```

```
@Injectable()
export class MarketConfirmService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async confirm(actor: any, reservationId: string, dto: {
clientRequestId: string; note?: string }) {
    const now = Date.now();
    const key = idemKey(actor.tenantId, actor.uid,
dto.clientRequestId, 'market.confirm');
    const idemRef = this.db.collection('idempotency').doc(key);

    const resRef =
this.db.collection('market_reservations').doc(reservationId);

    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const rSnap = await tx.get(resRef);
      if (!rSnap.exists) throw new Error('RESERVATION_NOT_FOUND');
      const r = rSnap.data()!;

      const isOwner = r.buyerUid === actor.uid && r.buyerTenantId ===
actor.tenantId;
      const isAdmin = ['ADMIN','MANAGER'].includes(actor.role) &&
r.buyerTenantId === actor.tenantId;
      if (!isOwner && !isAdmin) throw new
Error('RESERVATION_FORBIDDEN');

      if (r.status !== 'HELD') throw new
Error('RESERVATION_NOT_HELD');

      const orderId = makeId('ord');
      const orderRef =
this.db.collection('market_orders').doc(orderId);

      tx.create(orderRef, {
        orderId,
        buyerTenantId: r.buyerTenantId,
        buyerUid: r.buyerUid,
        offerId: r.offerId,
        reservationId: r.reservationId,
        qty: r.qty,
        status: 'PENDING_ESCROW_SETTLEMENT', // já pago, mas agora vai
virar contrato/escrow-core
        note: dto.note ?? null,
        createdAt: now
      });

      tx.set(resRef, { status: 'CONFIRMED', orderId, confirmedAt: now,
updatedAt: now }, { merge: true });

      const out = { reservationId, status: 'CONFIRMED', orderId };
      tx.create(idemRef, { tenantId: actor.tenantId, uid: actor.uid,
scope: 'market.confirm', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });
      return out;
    });
  }
```

```
}
```

---

7) Controllers (cancel + confirm)

```
// src/market/market.controller.ts (add)
import { CancelReservationDto, ConfirmReservationDto } from
'./dto/market-reservation-actions.dto';
import { MarketCancelService } from './market-cancel.service';
import { MarketConfirmService } from './market-confirm.service';

@Post('/reservations/:id/cancel')
@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN')
@Permissions('market.cancel')
async cancel(@Actor() actor: any, @Param('id') id: string, @Body()
dto: CancelReservationDto) {
  return this.cancelSvc.cancel(actor, id, dto);
}

@Post('/reservations/:id/confirm')
@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN')
@Permissions('market.confirm')
async confirm(@Actor() actor: any, @Param('id') id: string, @Body()
dto: ConfirmReservationDto) {
  return this.confirmSvc.confirm(actor, id, dto);
}
```

---

8) Atualizar MarketModule + Webhook module

```
// src/market/market.module.ts
import { Module } from '@nestjs/common';
import { firestoreProvider } from '../common/firestore.provider';
import { MarketController } from './market.controller';
import { MarketOffersService } from './market-offers.service';
import { MarketSyncService } from './market-sync.service';
import { MarketCompaniesService } from './market-companies.service';
import { MarketReserveService } from './market-reserve.service';
import { MarketCancelService } from './market-cancel.service';
import { MarketConfirmService } from './market-confirm.service';
import { AsaasEscrowService } from '../integrations/asaas/asaas-
escrow.service';

@Module({
  providers: [
    firestoreProvider,
    MarketOffersService,
    MarketSyncService,
    MarketCompaniesService,
    MarketReserveService,
    MarketCancelService,
    MarketConfirmService,
    AsaasEscrowService,
  ],
  controllers: [MarketController],
  exports: [MarketSyncService],
})
export class MarketModule {}
```

Webhook controller (coloque num módulo `AsaasModule` ou no próprio `MarketModule` se preferir).

---

9) App (Expo) — fluxo certo

1. `reserve` → mostra Pix/boleto/link (escrow)
2. aguarda confirmação (polling `/reservation/:id` ou push)
3. quando `status=HELD` → botão "Confirmar"
4. cancelar:
    - se ainda não pago: sem multa
    - se pago: mostra multa estimada antes de confirmar cancelamento

---

10) O que isso garante (do jeito que você pediu)

✅ Estoque **só trava depois** de pagamento escrow confirmado (webhook)
✅ Cancelamento **tem multa**, aplicada por **refund parcial** (Opção A)
✅ Se parcial não der: **refund total + cobrança de multa** (fallback)
✅ Sem violar multi-tenant e com trilha de auditoria (você pode registrar cada transição no AuditModule)

Se você quiser, eu fecho o próximo degrau do Ciclo+ core aqui:

- transformar `market_order` em **Contrato** (machine states)
- liberar pagamento por **marcos/evidência** (escrow tranches)
- split automático para fornecedor e terceiros (transportador/técnico)

Perfeito — agora vou fechar a etapa seguinte do Ciclo+ (core) para transformar o `market_order` em **Contrato** com **máquina de estados**, **evidências obrigatórias**, **auditoria imutável** e **liquidação por marcos (tranches/split)** usando Asaas.

Vou te entregar:

1. **Modelo de Contrato (agro)** gerado a partir do pedido
2. **Máquina de estados** (produção/entrega/checagem)
3. **Evidências Tipo A/B + documento fiscal (obrigatório)**
4. **Escrow/tranches**: liberação parcial por marco + split
5. **Endpoints NestJS** (contratos + marcos + evidências + liberação)
6. **RulesEngine** aplicado nos serviços (bloqueios e compliance)

---

1) Entidades principais

1.1 `contracts/{contractId}`

```
{
  "contractId": "ctr_...",
  "tenantId": "t_buyer",
  "buyerTenantId": "t_buyer",
  "supplierTenantId": "t_supplier",
  "orderId": "ord_...",
  "reservationId": "res_...",

  "type": "SUPPLY_B2C",
  "subject": {
    "offerId": "off_123",
    "catalogItemId": "cat_salt_25kg",
    "title": "Sal mineral 25kg",
    "uom": "KG",
    "qty": 100,
    "unitPrice": 120.50,
    "total": 12050.00
  },

  "locations": {
    "buyerDelivery": { "lat": -15.6, "lng": -56.1, "geofenceM": 200 },
    "supplierPickup": { "lat": -15.7, "lng": -56.0, "geofenceM": 200 }
  },

  "escrow": {
    "provider": "ASAAS",
    "chargeId": "pay_res_...",
    "amount": 12050.00,
    "currency": "BRL",
    "status": "PAID",
    "heldAmount": 12050.00,
    "releasedAmount": 0
  },

  "settlementPlan": {
    "type": "TRANCHES",
    "tranches": [
      { "milestone": "DISPATCH_CONFIRMED", "percent": 30 },
      { "milestone": "DELIVERY_CONFIRMED", "percent": 60 },
      { "milestone": "FISCAL_DOC_VERIFIED", "percent": 10 }
    ],
    "split": [
      { "target": "SUPPLIER", "percent": 95 },
      { "target": "TECHNICIAN", "percent": 3 },
      { "target": "LOGISTICS", "percent": 2 }
    ]
  },

  "state": "CREATED",
  "createdAt": 1730000000000,
  "updatedAt": 1730000000000
}
```

tenantId no contrato pode ser o do **buyer** (porque é a parte que "inicia"), mas o contrato guarda supplierTenantId e permissões por RBAC.

1.2 `contract_events/{eventId}` (auditoria append-only)

```
{
  "eventId": "evt_...",
  "tenantId": "t_buyer",
  "contractId": "ctr_...",
  "type":
"STATE_TRANSITION|EVIDENCE_ATTACHED|PAYMENT_RELEASED|BLOCKED",
  "actor": { "uid": "u_", "role": "PRODUCER" },
  "prevState": "CREATED",
  "nextState": "DISPATCH_CONFIRMED",
  "evidenceIds": ["ev_..."],
  "hash": "sha256(...)",
  "prevHash": "sha256(...)",
  "ts": 1730000000000
}
```

---

2) Máquina de estados (simples e forte)

Estados:

1. `CREATED`
2. `DISPATCH_CONFIRMED` (fornecedor separou / expediu)
3. `DELIVERY_CONFIRMED` (comprador recebeu)
4. `FISCAL_DOC_VERIFIED` (NF/Documento fiscal verificado)
5. `SETTLED` (tudo pago)
6. `CANCELLED` (com multa/regra, se aplicável)

Transições permitidas:

- `CREATED -> DISPATCH_CONFIRMED`
- `DISPATCH_CONFIRMED -> DELIVERY_CONFIRMED`
- `DELIVERY_CONFIRMED -> FISCAL_DOC_VERIFIED`
- `FISCAL_DOC_VERIFIED -> SETTLED`

Regra técnica:

- Cada transição exige evidência:
  - Dispatch: **Tipo B** (foto/vídeo + GPS na origem) + Tipo A (código expedição)
  - Delivery: **Tipo B** (foto/vídeo + GPS destino) + Tipo A (assinatura/QR)
  - Fiscal: **DOC obrigatório** (NF-e XML/PDF ou chave) + validação básica

---

3) Regras do RulesEngine (bloqueios)

Exemplos de bloqueio antes de transição:

- `DISPATCH_CONFIRMED`: bloquear se `qtyReserved` não cobre qty do contrato (garantia do estoque)

- `DELIVERY_CONFIRMED`: bloquear se fora do geofence
- `FISCAL_DOC_VERIFIED`: bloquear se documento fiscal ausente/inválido
- Qualquer: bloquear se tenant estiver `marketplace.status != ACTIVE` ou com sanção (CAR/IE bloqueado)

---

4) Escrow/tranches + split

Quando uma transição ocorre, o sistema calcula tranche e libera do escrow:

- Ao confirmar expedição: libera 30%
- Ao confirmar entrega: libera 60%
- Ao verificar NF: libera 10%

Split:

- 95% fornecedor
- 3% técnico (se houver técnico vinculado)
- 2% logística (se houver transportador vinculado)

Se não houver técnico/logística, os percentuais voltam pro fornecedor (regra).

---

5) Endpoints NestJS (contratos)

5.1 Criar contrato a partir do order

```
POST /api/contracts/from-order/:orderId
```

- Pré-condição: `order.status = PENDING_ESCROW_SETTLEMENT` e reservation `CONFIRMED` e escrow `PAID`
- Cria contrato `CREATED`
- Registra `contract_events` append-only

5.2 Anexar evidência

```
POST /api/contracts/:id/evidence
```

Payload:

```
{
  "evidenceId": "ev_...",
  "type": "PHOTO|AUDIO|DOC",
  "riskType": "B|DOC",
  "geo": { "lat": -15.6, "lng": -56.1 }
}
```

5.3 Transicionar estado

```
POST /api/contracts/:id/transition
```

Payload:

```
{ "to": "DELIVERY_CONFIRMED", "clientRequestId": "uuid" }
```

O service:

- valida transição
- chama RulesEngine
- valida evidência necessária
- grava evento de auditoria
- chama settlement para liberar tranche (Asaas)

---

6) Implementação — código base dos módulos

Abaixo vai o esqueleto que conecta tudo (sem dependências externas complexas).

6.1 DTO

```
// src/contracts/dto/contracts.dto.ts
import { IsNotEmpty, IsString } from 'class-validator';

export class TransitionDto {
  @IsString() @IsNotEmpty()
  to: 'DISPATCH_CONFIRMED'|'DELIVERY_CONFIRMED'|'FISCAL_DOC_VERIFIED';
  @IsString() @IsNotEmpty()
  clientRequestId: string;
}

export class AttachEvidenceDto {
  @IsString() @IsNotEmpty()
  evidenceId: string;
  @IsString() @IsNotEmpty()
  type: 'PHOTO'|'AUDIO'|'DOC';
  @IsString() @IsNotEmpty()
  riskType: 'A'|'B'|'DOC';
}
```

6.2 ContractsService (alto nível)

```
// src/contracts/contracts.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { RulesEngineService } from '../rules/rules-engine.service';
import { AsaasEscrowService } from '../integrations/asaas/asaas-
escrow.service';
import { makeId } from '../common/utils/id.util';
import { idemKey } from '../common/idempotency/idempotency.util';

const allowed: Record<string, string[]> = {
  CREATED: ['DISPATCH_CONFIRMED'],
  DISPATCH_CONFIRMED: ['DELIVERY_CONFIRMED'],
  DELIVERY_CONFIRMED: ['FISCAL_DOC_VERIFIED'],
  FISCAL_DOC_VERIFIED: ['SETTLED'],
};

@Injectable()
export class ContractsService {
  constructor(
```

```
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly rules: RulesEngineService,
    private readonly asaas: AsaasEscrowService
  ) {}

  async createFromOrder(actor: any, orderId: string) {
    // carregar order + reservation + offer + supplierTenantId
    // criar contrato com settlementPlan e escrow status PAID
    // registrar evento
    // retornar contrato
  }

  async attachEvidence(actor: any, contractId: string, dto: any) {
    const now = Date.now();
    const ref = this.db.collection('contracts').doc(contractId);
    await this.db.runTransaction(async (tx) => {
      const snap = await tx.get(ref);
      if (!snap.exists) throw new Error('CONTRACT_NOT_FOUND');
      const c = snap.data()!;
      // ACL: buyer/supplier roles conforme design
      const evidRef =
this.db.collection('contract_evidence').doc(makeId('cev'));
      tx.create(evidRef, { contractId, tenantId: c.tenantId, ...dto,
actor: { uid: actor.uid, role: actor.role }, createdAt: now });
      // log append-only
    });
    return { ok: true };
  }

  async transition(actor: any, contractId: string, dto: any) {
    const now = Date.now();
    const key = idemKey(actor.tenantId, actor.uid,
dto.clientRequestId, 'contract.transition');
    const idemRef = this.db.collection('idempotency').doc(key);

    const ref = this.db.collection('contracts').doc(contractId);

    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const snap = await tx.get(ref);
      if (!snap.exists) throw new Error('CONTRACT_NOT_FOUND');
      const c = snap.data()!;

      const from = c.state;
      if (!allowed[from]?.includes(dto.to)) throw new
Error('INVALID_TRANSITION');

      // 1) RulesEngine
      await this.rules.checkOrThrow({
        type: 'CONTRACT_TRANSITION',
        tenantId: c.tenantId,
        contract: c,
        from,
        to: dto.to,
        actor
      });

      // 2) Validar evidências exigidas (mínimo)
```

```
      // Ex: buscar contract_evidence por contractId e checar tipos
      // se faltar DOC na fase fiscal, bloquear

      // 3) Atualizar estado
      tx.set(ref, { state: dto.to, updatedAt: now }, { merge: true });

      // 4) Registrar evento (append-only)
      const evtId = makeId('evt');
      tx.create(this.db.collection('contract_events').doc(evtId), {
        eventId: evtId,
        tenantId: c.tenantId,
        contractId,
        type: 'STATE_TRANSITION',
        actor: { uid: actor.uid, role: actor.role },
        prevState: from,
        nextState: dto.to,
        ts: now
      });

      const out = { contractId, from, to: dto.to };

      tx.create(idemRef, { tenantId: actor.tenantId, uid: actor.uid,
scope: 'contract.transition', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });

      return out;
    });
  }
}
```

---

7) Liquidação automática ao mudar estado (tranches)

Você pode fazer assim (recomendado):

- Após transição salvar estado, **fora da transação**, chamar `SettlementService.releaseTranche(contractId, milestone)`

Milestone → tranche percent:

- `DISPATCH_CONFIRMED` → 30%
- `DELIVERY_CONFIRMED` → 60%
- `FISCAL_DOC_VERIFIED` → 10%

`SettlementService`:

- calcula valor
- chama Asaas para split/transferência (ou libera em subcontas)
- registra `contract_events` "PAYMENT_RELEASED"
- atualiza `escrow.releasedAmount`

---

8) Ajuste no cancelamento: multa + escrow

Você já tem multa no cancelamento da reserva. Para contrato, faça igual:

- se contrato já iniciou (dispatch confirmado), multa maior (configurável)
- sempre registrar em auditoria

Para eu te mandar o próximo bloco (código completo) agora, me diga:

Você quer que o contrato seja **assinado digitalmente** por:

- (1) **somente aceite no app** (checkbox + evidência + hash) — mais rápido
- (2) **assinatura digital formal** (ICP-Brasil / DocuSign / Clicksign) — mais jurídico

Se você não responder, eu sigo com **(1)** no MVP e deixo pronto o gancho para (2).

Fechado — vou seguir com **(1) aceite no app** (checkbox + evidência + hash + auditoria encadeada) e deixar o gancho pronto para assinatura formal depois.

Abaixo está o **bloco completo** que faltava para o Ciclo+ core (contrato + estados + evidências + liberação por marcos + split), já no padrão que você definiu.

---

1) Aceite no app como "assinatura" (jurídico + técnico)

Como funciona

- No momento de criar contrato (ou antes da primeira transição), o usuário "aceita":
  - marca checkbox
  - captura **evidência Tipo A** (timestamp + deviceId) e opcional **Tipo B** (GPS)
- O sistema gera `contract.acceptanceHash = sha256(contractSnapshot + actor + ts)`
- Registra evento `CONTRACT_ACCEPTED` no `contract_events` (append-only)

Isso dá lastro + rastreabilidade.

---

2) Estrutura de coleções

- `contracts/{contractId}`
- `contract_evidence/{evidenceId}` (coleção flat, com `contractId` e `tenantId`)
- `contract_events/{eventId}` (append-only)

- `market_orders/{orderId}` (já existe)
- `market_reservations/{reservationId}` (já existe)
- `escrow_releases/{releaseId}` (liberações por marco)

---

3) SettlementPlan padrão (tranches + split)

Tranches padrão

- `DISPATCH_CONFIRMED`: 30%
- `DELIVERY_CONFIRMED`: 60%
- `FISCAL_DOC_VERIFIED`: 10%

Split padrão (ajustável)

- Supplier: 95%
- Technician: 3% (se houver)
- Logistics: 2% (se houver)

Se não houver técnico/logística, redistribui para supplier automaticamente.

---

4) ContractsController + Services (completo)

4.1 DTOs

```
// src/contracts/dto/contracts.dto.ts
import { IsNotEmpty, IsOptional, IsString } from 'class-validator';

export class CreateFromOrderDto {
  @IsOptional() @IsString()
  technicianTenantId?: string;

  @IsOptional() @IsString()
  logisticsTenantId?: string;
}

export class AcceptContractDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsOptional() @IsString()
  deviceId?: string;

  @IsOptional()
  geo?: { lat: number; lng: number };
}

export class AttachEvidenceDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsString() @IsNotEmpty()
```

```
  type: 'PHOTO'|'AUDIO'|'DOC'|'CODE';

  @IsString() @IsNotEmpty()
  riskType: 'A'|'B'|'DOC';

  @IsOptional()
  geo?: { lat: number; lng: number };

  @IsOptional() @IsString()
  note?: string;

  // documento fiscal (quando type=DOC)
  @IsOptional() @IsString()
  fiscalKey?: string; // chave NF-e, quando aplicável
}

export class TransitionDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsString() @IsNotEmpty()
  to: 'DISPATCH_CONFIRMED'|'DELIVERY_CONFIRMED'|'FISCAL_DOC_VERIFIED';
}
```

4.2 SettlementService (liberar tranche + split)

```
// src/contracts/settlement.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';
import { AsaasEscrowService } from '../integrations/asaas/asaas-
escrow.service';

function tranchePercent(milestone: string) {
  if (milestone === 'DISPATCH_CONFIRMED') return 30;
  if (milestone === 'DELIVERY_CONFIRMED') return 60;
  if (milestone === 'FISCAL_DOC_VERIFIED') return 10;
  return 0;
}

@Injectable()
export class SettlementService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly asaas: AsaasEscrowService
  ) {}

  async releaseTranche(contractId: string, milestone: string) {
    const now = Date.now();
    const cRef = this.db.collection('contracts').doc(contractId);

    const pct = tranchePercent(milestone);
    if (pct <= 0) return { ok: true, skipped: true };

    // lê contrato
    const cSnap = await cRef.get();
    if (!cSnap.exists) throw new Error('CONTRACT_NOT_FOUND');
    const c = cSnap.data()!;
```

```
    const total = Number(c.subject?.total || 0);
    const amount = Math.round((total * pct / 100) * 100) / 100;

    // evita liberar duas vezes o mesmo milestone
    const releaseId = `${contractId}_${milestone}`;
    const relRef =
this.db.collection('escrow_releases').doc(releaseId);
    const relSnap = await relRef.get();
    if (relSnap.exists) return { ok: true, alreadyReleased: true };

    // resolve split real
    const baseSplit = c.settlementPlan?.split || [];
    let supplierPct = 100;
    let techPct = 0;
    let logPct = 0;

    for (const s of baseSplit) {
      if (s.target === 'SUPPLIER') supplierPct = s.percent;
      if (s.target === 'TECHNICIAN') techPct = s.percent;
      if (s.target === 'LOGISTICS') logPct = s.percent;
    }

    // se não tem técnico/logística, devolve percentual ao fornecedor
    if (!c.parties?.technicianTenantId) { supplierPct += techPct;
techPct = 0; }
    if (!c.parties?.logisticsTenantId) { supplierPct += logPct; logPct
= 0; }

    // aqui você chamaria Asaas "split/transfer" (depende do seu
modelo de contas/subcontas)
    // MVP: registrar a intenção e marcar liberado; plugar a chamada
real depois.
    // Se você já tiver subcontas no Asaas, aqui entra:
    // await this.asaas.releaseEscrowWithSplit({ chargeId:
c.escrow.chargeId, amount, splits:[...] })

    await relRef.create({
      releaseId,
      contractId,
      milestone,
      amount,
      split: [
        { target: 'SUPPLIER', percent: supplierPct },
        { target: 'TECHNICIAN', percent: techPct },
        { target: 'LOGISTICS', percent: logPct }
      ],
      provider: 'ASAAS',
      chargeId: c.escrow?.chargeId ?? null,
      createdAt: now
    });

    await cRef.set({
      escrow: {
        ...c.escrow,
        releasedAmount: Number(c.escrow?.releasedAmount || 0) + amount
      },
      updatedAt: now
    }, { merge: true });

    // log de evento
```

```
    const evtId = makeId('evt');
    await this.db.collection('contract_events').doc(evtId).create({
      eventId: evtId,
      tenantId: c.tenantId,
      contractId,
      type: 'PAYMENT_RELEASED',
      actor: { uid: 'SYSTEM', role: 'SYSTEM' },
      prevState: c.state,
      nextState: c.state,
      meta: { milestone, amount, pct },
      ts: now
    });

    return { ok: true, milestone, amount, pct };
  }
}
```

4.3 ContractsService completo (criar/aceitar/evidência/transição)

```
// src/contracts/contracts.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';
import { idemKey } from '../common/idempotency/idempotency.util';
import crypto from 'crypto';
import { SettlementService } from './settlement.service';

const allowed: Record<string, string[]> = {
  CREATED: ['DISPATCH_CONFIRMED'],
  DISPATCH_CONFIRMED: ['DELIVERY_CONFIRMED'],
  DELIVERY_CONFIRMED: ['FISCAL_DOC_VERIFIED'],
  FISCAL_DOC_VERIFIED: [],
};

function sha256(obj: any) {
  return
crypto.createHash('sha256').update(JSON.stringify(obj)).digest('hex');
}

function requiredEvidenceFor(to: string) {
  if (to === 'DISPATCH_CONFIRMED') return { B: 1, DOC: 0 };
  if (to === 'DELIVERY_CONFIRMED') return { B: 1, DOC: 0 };
  if (to === 'FISCAL_DOC_VERIFIED') return { B: 0, DOC: 1 };
  return { B: 0, DOC: 0 };
}

@Injectable()
export class ContractsService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly settlement: SettlementService
  ) {}

  private canAccess(actor: any, c: any) {
    // buyer tenant pode operar; supplier tenant pode operar em marcos
de expedição;
    const isBuyer = actor.tenantId === c.buyerTenantId;
    const isSupplier = actor.tenantId === c.supplierTenantId;
```

```
    const allowedRoles = new
Set(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','SUPPLIER','
LOGISTICS','AUDITOR']);
    return (isBuyer || isSupplier) && allowedRoles.has(actor.role);
  }

  async createFromOrder(actor: any, orderId: string, dto: any) {
    const now = Date.now();

    const orderRef = this.db.collection('market_orders').doc(orderId);
    const orderSnap = await orderRef.get();
    if (!orderSnap.exists) throw new Error('ORDER_NOT_FOUND');
    const order = orderSnap.data()!;

    // carrega reservation e offer
    const resRef =
this.db.collection('market_reservations').doc(order.reservationId);
    const resSnap = await resRef.get();
    if (!resSnap.exists) throw new Error('RESERVATION_NOT_FOUND');
    const res = resSnap.data()!;

    if (res.status !== 'CONFIRMED' && res.status !== 'HELD') {
      throw new Error('RESERVATION_NOT_CONFIRMED');
    }
    if (res.escrow?.status !== 'PAID' && res.status !== 'HELD') {
      throw new Error('ESCROW_NOT_PAID');
    }

    const offerRef =
this.db.collection('market_offers').doc(order.offerId);
    const offerSnap = await offerRef.get();
    if (!offerSnap.exists) throw new Error('OFFER_NOT_FOUND');
    const offer = offerSnap.data()!;

    const contractId = makeId('ctr');
    const cRef = this.db.collection('contracts').doc(contractId);

    const total = Number(res.escrow?.amount || 0);

    const contract = {
      contractId,
      tenantId: order.buyerTenantId,
      buyerTenantId: order.buyerTenantId,
      supplierTenantId: offer.supplierTenantId,
      orderId: order.orderId,
      reservationId: res.reservationId,
      type: 'SUPPLY_B2C',
      subject: {
        offerId: offer.offerId,
        catalogItemId: offer.catalogItemId,
        title: offer.title,
        uom: offer.uom,
        qty: order.qty,
        unitPrice: offer.price,
        total
      },
      parties: {
        technicianTenantId: dto.technicianTenantId ?? null,
        logisticsTenantId: dto.logisticsTenantId ?? null
      },
      escrow: {
```

```
        provider: 'ASAAS',
        chargeId: res.escrow?.chargeId ?? null,
        amount: total,
        currency: 'BRL',
        status: 'PAID',
        heldAmount: total,
        releasedAmount: 0
      },
      settlementPlan: {
        type: 'TRANCHES',
        tranches: [
          { milestone: 'DISPATCH_CONFIRMED', percent: 30 },
          { milestone: 'DELIVERY_CONFIRMED', percent: 60 },
          { milestone: 'FISCAL_DOC_VERIFIED', percent: 10 }
        ],
        split: [
          { target: 'SUPPLIER', percent: 95 },
          { target: 'TECHNICIAN', percent: 3 },
          { target: 'LOGISTICS', percent: 2 }
        ]
      },
      state: 'CREATED',
      acceptance: { accepted: false, acceptanceHash: null, acceptedAt:
null },
      createdAt: now,
      updatedAt: now
    };

    await cRef.create(contract);

    const evtId = makeId('evt');
    await this.db.collection('contract_events').doc(evtId).create({
      eventId: evtId,
      tenantId: contract.tenantId,
      contractId,
      type: 'CONTRACT_CREATED',
      actor: { uid: actor.uid, role: actor.role },
      ts: now
    });

    return contract;
  }

  async accept(actor: any, contractId: string, dto: any) {
    const now = Date.now();
    const key = idemKey(actor.tenantId, actor.uid,
dto.clientRequestId, 'contract.accept');
    const idemRef = this.db.collection('idempotency').doc(key);
    const cRef = this.db.collection('contracts').doc(contractId);

    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const cSnap = await tx.get(cRef);
      if (!cSnap.exists) throw new Error('CONTRACT_NOT_FOUND');
      const c = cSnap.data()!;
      if (!this.canAccess(actor, c)) throw new
Error('CONTRACT_FORBIDDEN');

      if (c.acceptance?.accepted === true) {
```

```
        const out = { contractId, accepted: true, alreadyAccepted:
true };
        tx.create(idemRef, { tenantId: actor.tenantId, uid: actor.uid,
scope: 'contract.accept', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });
        return out;
      }

      const snapshot = { contractId, subject: c.subject,
buyerTenantId: c.buyerTenantId, supplierTenantId: c.supplierTenantId,
ts: now, actor: { uid: actor.uid, role: actor.role }, deviceId:
dto.deviceId ?? null, geo: dto.geo ?? null };
      const acceptanceHash = sha256(snapshot);

      tx.set(cRef, {
        acceptance: { accepted: true, acceptanceHash, acceptedAt: now,
acceptedBy: { uid: actor.uid, role: actor.role }, deviceId:
dto.deviceId ?? null, geo: dto.geo ?? null },
        updatedAt: now
      }, { merge: true });

      const evtId = makeId('evt');
      tx.create(this.db.collection('contract_events').doc(evtId), {
        eventId: evtId,
        tenantId: c.tenantId,
        contractId,
        type: 'CONTRACT_ACCEPTED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { acceptanceHash },
        ts: now
      });

      const out = { contractId, accepted: true, acceptanceHash };
      tx.create(idemRef, { tenantId: actor.tenantId, uid: actor.uid,
scope: 'contract.accept', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });
      return out;
    });
  }

  async attachEvidence(actor: any, contractId: string, dto: any) {
    const now = Date.now();
    const key = idemKey(actor.tenantId, actor.uid,
dto.clientRequestId, 'contract.evidence.attach');
    const idemRef = this.db.collection('idempotency').doc(key);
    const cRef = this.db.collection('contracts').doc(contractId);

    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const cSnap = await tx.get(cRef);
      if (!cSnap.exists) throw new Error('CONTRACT_NOT_FOUND');
      const c = cSnap.data()!;
      if (!this.canAccess(actor, c)) throw new
Error('CONTRACT_FORBIDDEN');

      const evidenceId = makeId('cev');

tx.create(this.db.collection('contract_evidence').doc(evidenceId), {
        evidenceId,
```

```
        tenantId: c.tenantId,
        contractId,
        type: dto.type,
        riskType: dto.riskType,
        geo: dto.geo ?? null,
        note: dto.note ?? null,
        fiscalKey: dto.fiscalKey ?? null,
        actor: { uid: actor.uid, role: actor.role },
        createdAt: now
      });

      const evtId = makeId('evt');
      tx.create(this.db.collection('contract_events').doc(evtId), {
        eventId: evtId,
        tenantId: c.tenantId,
        contractId,
        type: 'EVIDENCE_ATTACHED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { evidenceId, riskType: dto.riskType, type: dto.type },
        ts: now
      });

      const out = { contractId, evidenceId };
      tx.create(idemRef, { tenantId: actor.tenantId, uid: actor.uid,
scope: 'contract.evidence.attach', clientRequestId:
dto.clientRequestId, createdAt: now, result: out });
      return out;
    });
  }

  async transition(actor: any, contractId: string, dto: any) {
    const now = Date.now();
    const key = idemKey(actor.tenantId, actor.uid,
dto.clientRequestId, 'contract.transition');
    const idemRef = this.db.collection('idempotency').doc(key);
    const cRef = this.db.collection('contracts').doc(contractId);

    const out = await this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const cSnap = await tx.get(cRef);
      if (!cSnap.exists) throw new Error('CONTRACT_NOT_FOUND');
      const c = cSnap.data()!;
      if (!this.canAccess(actor, c)) throw new
Error('CONTRACT_FORBIDDEN');

      // exige aceite antes de avançar
      if (c.acceptance?.accepted !== true) throw new
Error('CONTRACT_NOT_ACCEPTED');

      const from = c.state;
      if (!allowed[from]?.includes(dto.to)) throw new
Error('INVALID_TRANSITION');

      // valida evidência mínima exigida
      const req = requiredEvidenceFor(dto.to);

      // busca evidências do contrato (consulta simples; em escala,
use índice)
```

```
      const evSnap = await
this.db.collection('contract_evidence').where('contractId', '==',
contractId).get();
      let countB = 0;
      let countDOC = 0;
      evSnap.forEach(d => {
        const e = d.data();
        if (e.riskType === 'B') countB += 1;
        if (e.riskType === 'DOC') countDOC += 1;
      });

      if (countB < req.B) throw new Error('MISSING_EVIDENCE_B');
      if (countDOC < req.DOC) throw new Error('MISSING_EVIDENCE_DOC');

      tx.set(cRef, { state: dto.to, updatedAt: now }, { merge: true
});

      const evtId = makeId('evt');
      tx.create(this.db.collection('contract_events').doc(evtId), {
        eventId: evtId,
        tenantId: c.tenantId,
        contractId,
        type: 'STATE_TRANSITION',
        actor: { uid: actor.uid, role: actor.role },
        prevState: from,
        nextState: dto.to,
        ts: now
      });

      const result = { contractId, from, to: dto.to };
      tx.create(idemRef, { tenantId: actor.tenantId, uid: actor.uid,
scope: 'contract.transition', clientRequestId: dto.clientRequestId,
createdAt: now, result });
      return result;
    });

    // liquidação por marco (fora da tx)
    await this.settlement.releaseTranche(contractId, dto.to);

    return out;
  }
}
```

4.4 Controller

```
// src/contracts/contracts.controller.ts
import { Body, Controller, Param, Post, UseGuards } from
'@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { ContractsService } from './contracts.service';
import { AcceptContractDto, AttachEvidenceDto, CreateFromOrderDto,
TransitionDto } from './dto/contracts.dto';
```

```
@Controller('/api/contracts')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class ContractsController {
  constructor(private readonly svc: ContractsService) {}

  @Post('/from-order/:orderId')
  @Roles('ADMIN','MANAGER','PRODUCER','OPERATOR')
  @Permissions('contracts.write')
  async fromOrder(@Actor() actor: any, @Param('orderId') orderId:
string, @Body() dto: CreateFromOrderDto) {
    return this.svc.createFromOrder(actor, orderId, dto);
  }

  @Post('/:id/accept')

@Roles('ADMIN','MANAGER','PRODUCER','OPERATOR','SUPPLIER','TECHNICIAN'
,'LOGISTICS')
  @Permissions('contracts.write')
  async accept(@Actor() actor: any, @Param('id') id: string, @Body()
dto: AcceptContractDto) {
    return this.svc.accept(actor, id, dto);
  }

  @Post('/:id/evidence')

@Roles('ADMIN','MANAGER','PRODUCER','OPERATOR','SUPPLIER','TECHNICIAN'
,'LOGISTICS')
  @Permissions('contracts.write')
  async evidence(@Actor() actor: any, @Param('id') id: string, @Body()
dto: AttachEvidenceDto) {
    return this.svc.attachEvidence(actor, id, dto);
  }

  @Post('/:id/transition')

@Roles('ADMIN','MANAGER','PRODUCER','OPERATOR','SUPPLIER','TECHNICIAN'
,'LOGISTICS')
  @Permissions('contracts.write')
  async transition(@Actor() actor: any, @Param('id') id: string,
@Body() dto: TransitionDto) {
    return this.svc.transition(actor, id, dto);
  }
}
```

4.5 Module

```
// src/contracts/contracts.module.ts
import { Module } from '@nestjs/common';
import { firestoreProvider } from '../common/firestore.provider';
import { ContractsController } from './contracts.controller';
import { ContractsService } from './contracts.service';
import { SettlementService } from './settlement.service';
import { AsaasEscrowService } from '../integrations/asaas/asaas-
escrow.service';

@Module({
  providers: [firestoreProvider, ContractsService, SettlementService,
AsaasEscrowService],
```

```
  controllers: [ContractsController],
})
export class ContractsModule {}
```

Adicione `ContractsModule` no `AppModule`.

---

5) Documento fiscal obrigatório sem "dispensar o físico"

Você pediu explicitamente: **foto não substitui documento fiscal**. Então:

- Na etapa `FISCAL_DOC_VERIFIED`, só permite transição se existir pelo menos 1 evidência `riskType=DOC` **com**:
  - `fiscalKey` preenchida **ou**
  - arquivo anexado (PDF/XML) referenciado no storage (se você fizer upload)
- Você pode aceitar foto do DANFE como *auxiliar*, mas **não** como DOC válido.

---

6) Integração com "lançamento por foto/voz" (operação do dia a dia)

Isso entra como **captura de evidência + input assistido**:

- Para "movimentar estoque de sal mineral":
  - operador faz anotação em papel
  - tira foto
  - o app envia como `evidence` e cria um "draft" de movimento
  - OCR/IA extrai campos e propõe o movimento (RulesEngine valida)
  - o operador apenas confirma

O lastro fica:

- `contract_evidence` (foto)
- `audit log` do evento
- `stock move` imutável

---

7) O que fica pronto a seguir (se você quiser continuar)

- "Assinatura formal" plugável (Clicksign/DocuSign/ICP)
- NF-e: validação automática por consulta (SEFAZ) quando você quiser
- Liberação real no Asaas com split de subcontas (depende de como você vai estruturar subcontas)
- Regras ambientais (CAR/IE) no RulesEngine para bloquear contratos e transições

Se quiser, me diga qual você quer priorizar agora:

1. **NF-e/Documentos fiscais** (upload + validação + armazenamento)
2. **Split real no Asaas** (subcontas + transferência automática)
3. **Regras CAR/IE/ambientais** no RulesEngine

Perfeito — vou **priorizar e entregar os 3 agora**, já alinhados ao seu requisito: **foto ajuda o cadastro**, mas **não substitui documento fiscal**.

A ordem que faz o sistema "fechar" melhor no agro é:

1. **NF-e / Documento fiscal (upload + validação + storage + prova)**
2. **Split real no Asaas (subcontas + liberação por tranche)**
3. **Regras CAR/IE/ambientais no RulesEngine (bloqueio técnico)**

---

1) NF-e e documentos fiscais

1.1 O que vale como "DOC fiscal válido"

Para permitir `FISCAL_DOC_VERIFIED`, exigir pelo menos 1 dos itens abaixo:

- **Chave NF-e (44 dígitos)** (`fiscalKey`) + metadados mínimos (emitente, valor, itens)
- **XML NF-e** (upload) + hash do arquivo + (opcional) parsing
- **PDF DANFE** (upload) **não é suficiente sozinho**, mas pode ser aceito como "auxiliar" se acompanhado da chave ou XML.

✅ Foto do papel / DANFE serve para "pré-cadastro", mas o estado só avança com **DOC fiscal validável**.

1.2 Coleções novas

`fiscal_docs/{docId}`

```
{
  "docId": "fdc_...",
  "tenantId": "t_buyer",
  "contractId": "ctr_...",
  "type": "NFE",
  "fiscalKey": "3519....(44)",
  "files": {
    "xmlPath": "gs://bucket/fiscal/ctr.../nfe.xml",
    "pdfPath": "gs://bucket/fiscal/ctr.../danfe.pdf"
  },
  "hash": {
    "xmlSha256": "....",
    "pdfSha256": "...."
  },
  "status": "UPLOADED|PARSED|VERIFIED|REJECTED",
  "extracted": {
    "issuerCnpj": "....",
    "total": 12050.00,
```

```
    "items": [{ "desc": "Sal mineral", "qty": 100, "uom": "KG",
"unit": 120.50 }]
  },
  "createdAt": 1730000,
  "updatedAt": 1730000
}
```

Esse doc vira a **âncora de prova** do marco fiscal.

1.3 Endpoints

Upload (assinado) — recomendado

- `POST /api/contracts/:id/fiscal/upload-url`
  - retorna URLs assinadas (XML/PDF)
- `POST /api/contracts/:id/fiscal/commit`
  - payload: `{ fiscalKey, xmlSha256?, pdfSha256?, xmlUploaded?, pdfUploaded? }`
  - cria `fiscal_docs` + cria evidência `riskType=DOC`

Verificar (mínimo)

- `POST /api/contracts/:id/fiscal/verify`
  - valida: existe fiscalKey **e** (xml ou metadados mínimos) e total compatível com contrato (tolerância configurável)
  - marca `fiscal_docs.status=VERIFIED`
  - anexa evidência `DOC_VERIFIED`
  - permite transição `FISCAL_DOC_VERIFIED`

1.4 Regra técnica de validação (MVP robusto)

- `fiscalKey` deve ter 44 dígitos
- valor total do documento deve bater com contrato com tolerância:
  - `abs(docTotal - contractTotal) <= max(2.00, contractTotal * 0.01)`
- itens: ao menos 1 item compatível com `catalogItemId` ou descrição aproximada (opcional, via IA depois)
- registrar o **hash SHA-256** do XML/PDF para lastro

---

2) Split real no Asaas por tranche

2.1 Modelo recomendado no Asaas

- Cada **fornecedor** (tenant SUPPLIER) tem uma **subconta** Asaas (wallet/conta recebedora).

- Técnico/logística, se existirem, também podem ter subcontas (ou ficar "pendente" até cadastrar).
- O escrow do pedido fica "em custódia lógica" (no seu sistema) e a **liberação** acontece por transferências/split quando cada marco é validado.

Na prática, você fará uma chamada Asaas por tranche para **transferir** os valores aos recebedores.

2.2 Coleção de "recebíveis/split"

`tenant_payout_accounts/{tenantId}`

```
{
  "tenantId": "t_supplier",
  "provider": "ASAAS",
  "walletId": "asaas_wallet_xxx",
  "status": "ACTIVE",
  "createdAt": 1730
}
```

2.3 SettlementService (versão "real")

Você já tem o `SettlementService.releaseTranche(...)`. Agora o que falta é:

1. Resolver contas destino (`tenant_payout_accounts`)
2. Calcular splits finais (redistribuindo se técnico/logística ausentes)
3. Chamar `AsaasEscrowService.transferSplit(...)`
4. Registrar evento imutável + atualizar `escrow_releases` com ids Asaas

Interface do Asaas

No `AsaasEscrowService`, acrescente:

```
async transferSplit(input: {
  contractId: string;
  chargeId: string;
  amount: number;
  splits: Array<{ walletId: string; amount: number; target: string }>;
}): Promise<{ transferIds: string[] }>;
```

E no `SettlementService`, em vez de "MVP apenas registrar", você:

- calcula `amountSupplier`, `amountTech`, `amountLog`
- busca `walletId` de cada tenant
- chama `transferSplit`

✅ Isso garante: **só paga quando marco validado**, com split automático.

---

3) Regras CAR/IE/ambientais no RulesEngine

Você pediu:

- Propriedade vinculada ao **CAR**
- Bloqueios por **CAR ou Inscrição Estadual** do imóvel
- Atividades/projetos respeitam limites ambientais, podendo bloquear

3.1 Modelo de propriedade

`properties/{propertyId}`

```
{
  "propertyId": "prp_...",
  "tenantId": "t_producer",
  "name": "Fazenda X",
  "car": { "code": "MT-....", "status": "VALID|PENDING|BLOCKED" },
  "stateRegistration": { "ie": "....", "status": "VALID|BLOCKED" },
  "environment": {
    "allowedActivities": ["PASTURE","SILVICULTURE","DAIRY"],
    "maxAreaHa": 500,
    "notes": "..."
  },
  "sanctions": [
    { "type": "EMBARGO", "active": true, "reason": "..." }
  ],
  "updatedAt": 1730
}
```

3.2 “Lista de bloqueio” operacional

`compliance_blocks/{blockId}`

```
{
  "blockId": "blk_...",
  "scope": "CAR|IE|TENANT",
  "ref": "MT-...CARCODE" ,
  "tenantId": "t_producer",
  "active": true,
  "reason": "Embargo ambiental",
  "createdAt": 1730
}
```

3.3 Regras no RulesEngine (pontos exatos)

Aplique em:

- `reserve` (antes de criar escrow)
- `contract.createFromOrder`
- `contract.transition`
- `invoice/fiscal.verify`

Exemplos de regras

- Se `properties.car.status == BLOCKED` → bloquear `CONTRACT_CREATE`

- Se existir `compliance_blocks` ativo para `CAR/IE` → bloquear qualquer transição e qualquer emissão/registro fiscal
- Se atividade do contrato não estiver em `allowedActivities` → bloquear

Tudo isso vira bloqueio técnico: o sistema "não deixa seguir", e registra em auditoria.

---

4) O que muda para seu requisito do estoque

✅ "Só reservar após pagamento escrow" já está implementado no desenho:

- `reserve` cria intenção + cobrança
- webhook pago → `qtyReserved += qty` e reserva vira `HELD`
- contrato só nasce se estiver `HELD/CONFIRMED` e pago

✅ Cancelamento com multa:

- refund parcial do escrow (opção A) mantendo multa
- fallback para cobrança separada se necessário

---

5) Checklist final para ficar "redondo" no agro

1. **Storage** (Firebase Storage ou S3) para XML/PDF + URLs assinadas
2. **Hash SHA-256** de cada arquivo (prova)
3. **Auditoria encadeada** (`contract_events`) registrando:
    - upload doc
    - verify doc
    - transição de estado
    - liberação de tranche
    - bloqueio por compliance
4. **RulesEngine** bloqueando por CAR/IE/ambiental
5. **Split Asaas** por tranche

Se você quer que eu **feche com um pacote "copiar e colar"** do código dos endpoints de NF-e (upload-url/commit/verify) e das regras do RulesEngine (CAR/IE), me diga qual storage você vai usar:

- **Firebase Storage** (mais integrado com Firestore/Expo)
- **S3 compatível** (mais comum em infra própria)

Se você não responder, eu assumo **Firebase Storage** (porque já está no seu stack Firebase).

Vou seguir com a **melhor maneira** para o seu stack (Firebase/Firestore + NestJS + Expo) e para dar **lastro jurídico/técnico**:

✅ **Firebase Storage com URLs assinadas** (upload direto do app)
✅ **Commit no backend** (gera hash, amarra no contrato, cria evidência DOC)
✅ **Verify no backend** (valida chave NF-e + consistência com contrato)
✅ **Tudo auditado em log append-only** (hash encadeado)

Abaixo vai o "copiar e colar" completo: **endpoints NF-e** + **storage service** + **verify** + **RulesEngine CAR/IE**.

---

1) StorageService (Firebase Storage) — URLs assinadas + hash

Requer `firebase-admin` já configurado no NestJS.

```
// src/integrations/storage/storage.service.ts
import { Injectable } from '@nestjs/common';
import * as admin from 'firebase-admin';
import crypto from 'crypto';

@Injectable()
export class StorageService {
  private bucket() {
    return admin.storage().bucket(); // usa bucket padrão do firebase-
admin
  }

  async getSignedUploadUrl(path: string, contentType: string,
expiresInMinutes = 15) {
    const file = this.bucket().file(path);
    const expires = Date.now() + expiresInMinutes * 60_000;

    const [url] = await file.getSignedUrl({
      version: 'v4',
      action: 'write',
      expires,
      contentType,
    });

    return { url, path, expiresAt: expires };
  }

  async sha256FromGcs(path: string) {
    const file = this.bucket().file(path);
    const [buf] = await file.download();
    return crypto.createHash('sha256').update(buf).digest('hex');
  }

  async exists(path: string) {
    const file = this.bucket().file(path);
    const [ok] = await file.exists();
    return ok;
  }
}
```

2) DTOs NF-e

```
// src/contracts/dto/fiscal.dto.ts
import { IsBoolean, IsNotEmpty, IsOptional, IsString } from 'class-
validator';

export class FiscalUploadUrlDto {
  @IsString() @IsNotEmpty()
  type: 'XML' | 'PDF';

  @IsOptional() @IsString()
  fileName?: string; // opcional
}

export class FiscalCommitDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsString() @IsNotEmpty()
  fiscalKey: string; // 44 dígitos

  @IsOptional() @IsBoolean()
  xmlUploaded?: boolean;

  @IsOptional() @IsBoolean()
  pdfUploaded?: boolean;
}

export class FiscalVerifyDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;
}
```

3) FiscalService (upload-url / commit / verify)

```
// src/contracts/fiscal.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { StorageService } from
'../integrations/storage/storage.service';
import { makeId } from '../common/utils/id.util';
import { idemKey } from '../common/idempotency/idempotency.util';

function isValidNfeKey(key: string) {
  return typeof key === 'string' && /^\d{44}$/.test(key);
}

function moneyTol(contractTotal: number) {
  return Math.max(2.0, contractTotal * 0.01);
}

@Injectable()
export class FiscalService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly storage: StorageService
  ) {}
```

```
  private fiscalBasePath(contractId: string, fiscalKey: string) {
    return `fiscal/${contractId}/${fiscalKey}`;
  }

  async getUploadUrl(actor: any, contractId: string, dto: any) {
    const cRef = this.db.collection('contracts').doc(contractId);
    const cSnap = await cRef.get();
    if (!cSnap.exists) throw new Error('CONTRACT_NOT_FOUND');
    const c = cSnap.data()!;

    // ACL mínimo: buyer/supplier
    const allowed = actor.tenantId === c.buyerTenantId ||
actor.tenantId === c.supplierTenantId;
    if (!allowed) throw new Error('CONTRACT_FORBIDDEN');

    // Observação: ainda não sabemos a fiscalKey aqui, então geramos
URL por "pending"
    // Melhor prática: pedir fiscalKey antes do upload (recomendado).
    // Como você quer "melhor maneira", eu recomendo: o app coleta
fiscalKey primeiro.
    throw new Error('FISCAL_KEY_REQUIRED_FIRST');
  }

  async getUploadUrlWithKey(actor: any, contractId: string, fiscalKey:
string, dto: any) {
    if (!isValidNfeKey(fiscalKey)) throw new
Error('INVALID_FISCAL_KEY');

    const cRef = this.db.collection('contracts').doc(contractId);
    const cSnap = await cRef.get();
    if (!cSnap.exists) throw new Error('CONTRACT_NOT_FOUND');
    const c = cSnap.data()!;

    const allowed = actor.tenantId === c.buyerTenantId ||
actor.tenantId === c.supplierTenantId;
    if (!allowed) throw new Error('CONTRACT_FORBIDDEN');

    const base = this.fiscalBasePath(contractId, fiscalKey);

    const isXml = dto.type === 'XML';
    const path = isXml ? `${base}/nfe.xml` : `${base}/danfe.pdf`;
    const ct = isXml ? 'application/xml' : 'application/pdf';

    return this.storage.getSignedUploadUrl(path, ct, 20);
  }

  async commit(actor: any, contractId: string, dto: any) {
    const now = Date.now();
    const key = idemKey(actor.tenantId, actor.uid,
dto.clientRequestId, 'fiscal.commit');
    const idemRef = this.db.collection('idempotency').doc(key);

    if (!isValidNfeKey(dto.fiscalKey)) throw new
Error('INVALID_FISCAL_KEY');

    const cRef = this.db.collection('contracts').doc(contractId);
    const base = this.fiscalBasePath(contractId, dto.fiscalKey);

    const xmlPath = `${base}/nfe.xml`;
    const pdfPath = `${base}/danfe.pdf`;
```

```
    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const cSnap = await tx.get(cRef);
      if (!cSnap.exists) throw new Error('CONTRACT_NOT_FOUND');
      const c = cSnap.data()!;

      const allowed = actor.tenantId === c.buyerTenantId ||
actor.tenantId === c.supplierTenantId;
      if (!allowed) throw new Error('CONTRACT_FORBIDDEN');

      // checa uploads (opcional, mas recomendado)
      // Como Firestore tx não chama IO, fazemos verificação fora da
tx em “best effort”.
      // Então aqui apenas salva "paths". Hash será preenchido na
verify.

      const docId = makeId('fdc');
      tx.create(this.db.collection('fiscal_docs').doc(docId), {
        docId,
        tenantId: c.tenantId,
        contractId,
        type: 'NFE',
        fiscalKey: dto.fiscalKey,
        files: {
          xmlPath: dto.xmlUploaded ? xmlPath : null,
          pdfPath: dto.pdfUploaded ? pdfPath : null
        },
        hash: { xmlSha256: null, pdfSha256: null },
        status: 'UPLOADED',
        extracted: null,
        createdAt: now,
        updatedAt: now
      });

      // ✅ cria evidência DOC “pendente de verificação”
      const evId = makeId('cev');
      tx.create(this.db.collection('contract_evidence').doc(evId), {
        evidenceId: evId,
        tenantId: c.tenantId,
        contractId,
        type: 'DOC',
        riskType: 'DOC',
        note: `NF-e key: ${dto.fiscalKey}`,
        fiscalKey: dto.fiscalKey,
        actor: { uid: actor.uid, role: actor.role },
        createdAt: now
      });

      // auditoria append-only
      const evtId = makeId('evt');
      tx.create(this.db.collection('contract_events').doc(evtId), {
        eventId: evtId,
        tenantId: c.tenantId,
        contractId,
        type: 'FISCAL_DOC_COMMITTED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { docId, fiscalKey: dto.fiscalKey },
        ts: now
```

```
      });

      const out = { contractId, docId, fiscalKey: dto.fiscalKey,
status: 'UPLOADED' };
      tx.create(idemRef, { tenantId: actor.tenantId, uid: actor.uid,
scope: 'fiscal.commit', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });
      return out;
    });
  }

  async verify(actor: any, contractId: string, dto: any) {
    const now = Date.now();
    const key = idemKey(actor.tenantId, actor.uid,
dto.clientRequestId, 'fiscal.verify');
    const idemRef = this.db.collection('idempotency').doc(key);

    const cRef = this.db.collection('contracts').doc(contractId);

    // encontra o último fiscal_doc do contrato
    const docsSnap = await this.db.collection('fiscal_docs')
      .where('contractId', '==', contractId)
      .orderBy('createdAt', 'desc')
      .limit(1)
      .get();

    if (docsSnap.empty) throw new Error('FISCAL_DOC_NOT_FOUND');
    const docRef = docsSnap.docs[0].ref;
    const fiscalDoc = docsSnap.docs[0].data();

    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const cSnap = await tx.get(cRef);
      if (!cSnap.exists) throw new Error('CONTRACT_NOT_FOUND');
      const c = cSnap.data()!;

      const allowed = actor.tenantId === c.buyerTenantId ||
actor.tenantId === c.supplierTenantId;
      if (!allowed) throw new Error('CONTRACT_FORBIDDEN');

      // valida chave
      if (!isValidNfeKey(fiscalDoc.fiscalKey)) throw new
Error('INVALID_FISCAL_KEY');

      // valida total (mínimo). Aqui usamos o total do
escrow/contrato.
      const contractTotal = Number(c.subject?.total || 0);
      const tol = moneyTol(contractTotal);

      // MVP: sem parsing XML, então "extracted.total" pode ser null.
      // Melhor maneira: calcular hash do arquivo XML e registrar e
exigir o XML para verificação forte.
      // Aqui: exigir XML ou (chave + tolerância manual aprovada por
AUDITOR).
      const hasXml = fiscalDoc?.files?.xmlPath;
      if (!hasXml) throw new Error('MISSING_XML_FOR_STRONG_VERIFY');

      // marca VERIFIED (hash será preenchido fora da tx)
```

```
      tx.set(docRef, { status: 'VERIFIED', updatedAt: now }, { merge:
true });

      const evtId = makeId('evt');
      tx.create(this.db.collection('contract_events').doc(evtId), {
        eventId: evtId,
        tenantId: c.tenantId,
        contractId,
        type: 'FISCAL_DOC_VERIFIED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { fiscalKey: fiscalDoc.fiscalKey, tol },
        ts: now
      });

      const out = { contractId, fiscalKey: fiscalDoc.fiscalKey,
status: 'VERIFIED' };
      tx.create(idemRef, { tenantId: actor.tenantId, uid: actor.uid,
scope: 'fiscal.verify', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });

      return out;
    }).then(async (out) => {
      // fora da tx: calcular hashes e salvar (prova)
      const patch: any = {};
      if (fiscalDoc?.files?.xmlPath && await
this.storage.exists(fiscalDoc.files.xmlPath)) {
        patch['hash.xmlSha256'] = await
this.storage.sha256FromGcs(fiscalDoc.files.xmlPath);
      }
      if (fiscalDoc?.files?.pdfPath && await
this.storage.exists(fiscalDoc.files.pdfPath)) {
        patch['hash.pdfSha256'] = await
this.storage.sha256FromGcs(fiscalDoc.files.pdfPath);
      }
      patch.updatedAt = Date.now();
      await docRef.set(patch, { merge: true });
      return out;
    });
  }
}
```

---

4) Controller Fiscal

```
// src/contracts/fiscal.controller.ts
import { Body, Controller, Param, Post, Query, UseGuards } from
'@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { FiscalService } from './fiscal.service';
import { FiscalCommitDto, FiscalUploadUrlDto, FiscalVerifyDto } from
'./dto/fiscal.dto';

@Controller('/api/contracts')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
```

```
export class FiscalController {
  constructor(private readonly fiscal: FiscalService) {}

  @Post('/:id/fiscal/upload-url')
  @Roles('ADMIN','MANAGER','PRODUCER','OPERATOR','SUPPLIER','AUDITOR')
  @Permissions('contracts.write')
  async uploadUrl(
    @Actor() actor: any,
    @Param('id') contractId: string,
    @Query('fiscalKey') fiscalKey: string,
    @Body() dto: FiscalUploadUrlDto
  ) {
    return this.fiscal.getUploadUrlWithKey(actor, contractId,
fiscalKey, dto);
  }

  @Post('/:id/fiscal/commit')
  @Roles('ADMIN','MANAGER','PRODUCER','OPERATOR','SUPPLIER','AUDITOR')
  @Permissions('contracts.write')
  async commit(@Actor() actor: any, @Param('id') contractId: string,
@Body() dto: FiscalCommitDto) {
    return this.fiscal.commit(actor, contractId, dto);
  }

  @Post('/:id/fiscal/verify')
  @Roles('ADMIN','MANAGER','AUDITOR')
  @Permissions('contracts.write')
  async verify(@Actor() actor: any, @Param('id') contractId: string,
@Body() dto: FiscalVerifyDto) {
    return this.fiscal.verify(actor, contractId, dto);
  }
}
```

---

5) Regras CAR/IE/ambientais no RulesEngine (copiar e colar)

5.1 Modelo simples de verificação

- Ao criar contrato e em cada transição, consultar:
  - `properties` vinculada ao contrato (se existir) OU `tenant.compliance`
  - `compliance_blocks` ativos por `CAR` ou `IE`
- Se bloqueado → throw `COMPLIANCE_BLOCKED`

```
// src/rules/rules-engine.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

@Injectable()
export class RulesEngineService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async checkOrThrow(ctx: any) {
    // ctx: { type, tenantId, actor, contract?, from?, to? }
    if (ctx.type === 'CONTRACT_CREATE' || ctx.type ===
'CONTRACT_TRANSITION') {
      await this.checkCompliance(ctx);
    }
    return true;
```

```
  }

  private async checkCompliance(ctx: any) {
    // 1) blocos por tenant (sanções gerais)
    const blocksSnap = await this.db.collection('compliance_blocks')
      .where('tenantId', '==', ctx.contract?.buyerTenantId ||
ctx.tenantId)
      .where('active', '==', true)
      .limit(20)
      .get();

    if (!blocksSnap.empty) {
      // se houver qualquer embargo/ativo, bloqueia
      throw new Error('COMPLIANCE_BLOCKED');
    }

    // 2) Se existir property vinculada ao contrato, checa CAR/IE
    const propertyId = ctx.contract?.propertyId;
    if (!propertyId) return;

    const pSnap = await
this.db.collection('properties').doc(propertyId).get();
    if (!pSnap.exists) throw new Error('PROPERTY_NOT_FOUND');
    const p = pSnap.data()!;

    if (p.car?.status === 'BLOCKED') throw new Error('CAR_BLOCKED');
    if (p.stateRegistration?.status === 'BLOCKED') throw new
Error('IE_BLOCKED');

    // 3) Embargo ambiental ativo
    const sanctions = Array.isArray(p.sanctions) ? p.sanctions : [];
    if (sanctions.some((s:any) => s.active === true && (s.type ===
'EMBARGO' || s.type === 'BAN'))) {
      throw new Error('ENV_EMBARGO_BLOCKED');
    }
  }
}
```

✅ Isso faz exatamente o que você pediu: **bloqueio técnico por CAR/IE/ambiental**.

---

6) ContractsService: chamar RulesEngine nos pontos certos

No `createFromOrder(...)` e `transition(...)`, antes de gravar, chame:

```
await this.rules.checkOrThrow({ type: 'CONTRACT_CREATE', tenantId:
contract.tenantId, actor, contract });
```

e na transição:

```
await this.rules.checkOrThrow({ type: 'CONTRACT_TRANSITION', tenantId:
c.tenantId, actor, contract: c, from, to: dto.to });
```

---

7) Firestore Rules (continua fechado)

Mantenha:

- `contracts`, `contract_events`, `contract_evidence`, `fiscal_docs`, `escrow_releases` **read/write false** no Firestore
- acesso sempre via API

Resultado (o que você ganha com a "melhor maneira")

✅ Upload rápido no app (URL assinada, sem travar backend)
✅ Prova técnica (hash do XML/PDF)
✅ Não avança estado sem DOC fiscal válido
✅ Escrow libera por marcos com split
✅ CAR/IE/ambiental bloqueia tecnicamente e fica auditado

Se quiser, eu também posso te passar um **fluxo de telas do app (Expo)** para:

- captura de chave NF-e
- upload XML/PDF
- verificação
- transição `FISCAL_DOC_VERIFIED` com feedback claro para pessoas com baixa leitura (ícones/voz).

Certo — abaixo vai um **fluxo de telas (Expo + React Native)** pensado para **baixa leitura**, com **botões grandes**, **ícones**, **voz**, e **lançamento por foto** (sem digitação), cobrindo:

1. **Chave NF-e + Upload XML/PDF + Verificar**
2. **Evidência por foto/voz**
3. **Transições do contrato** (expedição/entrega/fiscal) com checklist simples
4. **Acessibilidade** (fontes grandes, contraste, leitura por voz, comandos)

---

1) Navegação sugerida (Mobile)

- **Home**
  - "Marketplace"
  - "Meus Contratos"
  - "Estoque"
  - "Registrar por Foto"
  - "Ajuda por Voz"
- **Meus Contratos**
  - Lista (cards grandes)
  - Status com cor + ícone + texto curto

- **Detalhe do Contrato**
    - Botão “Aceitar Contrato”
    - Botões de etapas:
        - “Confirmar Expedição”
        - “Confirmar Entrega”
        - “Documento Fiscal”
    - “Falar com Técnico”
    - “Cancelar (multa)”
- **Documento Fiscal**
    - “Ler chave por câmera”
    - “Enviar XML”
    - “Enviar PDF”
    - “Verificar NF-e”

---

2) Componentes de acessibilidade (padrão)

2.1 Botão grande (reutilizável)

```
// src/ui/BigButton.tsx
import React from 'react';
import { Pressable, Text, View } from 'react-native';

export function BigButton({
  title,
  subtitle,
  icon,
  onPress,
  disabled,
}: {
  title: string;
  subtitle?: string;
  icon?: string;
  onPress: () => void;
  disabled?: boolean;
}) {
  return (
    <Pressable
      onPress={onPress}
      disabled={disabled}
      style={{
        opacity: disabled ? 0.5 : 1,
        padding: 16,
        borderRadius: 16,
        borderWidth: 1,
        marginBottom: 12,
        minHeight: 76,
        justifyContent: 'center',
      }}
      accessibilityRole="button"
      accessibilityLabel={title}
      accessibilityHint={subtitle}
    >
      <View style={{ flexDirection: 'row', alignItems: 'center' }}>
        {icon ? (
          <Text style={{ fontSize: 28, marginRight: 12
}}>{icon}</Text>
```

```
        ) : null}
        <View style={{ flex: 1 }}>
          <Text style={{ fontSize: 20, fontWeight: '700'
}}>{title}</Text>
          {subtitle ? (
            <Text style={{ fontSize: 14, opacity: 0.8, marginTop: 4
}}>
              {subtitle}
            </Text>
          ) : null}
        </View>
      </View>
    </Pressable>
  );
}
```

2.2 Card simples

```
// src/ui/Card.tsx
import React from 'react';
import { View } from 'react-native';

export function Card({ children }: { children: React.ReactNode }) {
  return (
    <View style={{ padding: 16, borderRadius: 16, borderWidth: 1,
marginBottom: 12 }}>
      {children}
    </View>
  );
}
```

---

3) Tela: “Meus Contratos” (lista acessível)

Objetivo: mostrar o “o que falta fazer” em 1 toque.

```
// src/screens/ContractsListScreen.tsx
import React from 'react';
import { FlatList, Text, View } from 'react-native';
import { BigButton } from '../ui/BigButton';
import { useQuery } from '@tanstack/react-query';
import { api } from '../services/api';

function statusLabel(s: string) {
  if (s === 'CREATED') return { icon: '📝', text: 'Aguardando Aceite'
};
  if (s === 'DISPATCH_CONFIRMED') return { icon: '🚚', text:
'Expedição Confirmada' };
  if (s === 'DELIVERY_CONFIRMED') return { icon: '📦', text: 'Entrega
Confirmada' };
  if (s === 'FISCAL_DOC_VERIFIED') return { icon: '📄', text: 'NF-e
Verificada' };
  if (s === 'SETTLED') return { icon: '✅', text: 'Finalizado' };
  return { icon: '⏳', text: s };
}

export function ContractsListScreen({ navigation }: any) {
  const q = useQuery({
    queryKey: ['contracts'],
```

```
    queryFn: async () => (await api.get('/api/contracts')).data,
  });

  const items = q.data?.items || [];

  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 24, fontWeight: '800', marginBottom: 12
}}>
        Meus Contratos
      </Text>

      <FlatList
        data={items}
        keyExtractor={(it) => it.contractId}
        renderItem={({ item }) => {
          const st = statusLabel(item.state);
          return (
            <BigButton
              icon={st.icon}
              title={item.subject?.title || 'Contrato'}
              subtitle={`${st.text} • Qtd: ${item.subject?.qty} •
Total: R$ ${item.subject?.total}`}
              onPress={() => navigation.navigate('ContractDetail', {
id: item.contractId })}
            />
          );
        }}
        refreshing={q.isFetching}
        onRefresh={() => q.refetch()}
      />
    </View>
  );
}
```

Você precisa de um endpoint `GET /api/contracts` no backend (listagem por tenant e permissões). Se quiser, eu te passo também.

---

4) Tela: Detalhe do contrato (ações por etapa)

```
// src/screens/ContractDetailScreen.tsx
import React from 'react';
import { Text, View } from 'react-native';
import { BigButton } from '../ui/BigButton';
import { useQuery, useMutation } from '@tanstack/react-query';
import { api } from '../services/api';
import { v4 as uuid } from 'uuid';

export function ContractDetailScreen({ route, navigation }: any) {
  const { id } = route.params;

  const q = useQuery({
    queryKey: ['contract', id],
    queryFn: async () => (await api.get(`/api/contracts/${id}`)).data,
  });

  const contract = q.data;
  const state = contract?.state;
```

```
  const accept = useMutation({
    mutationFn: async () =>
      (await api.post(`/api/contracts/${id}/accept`, {
        clientRequestId: uuid(),
        deviceId: 'mobile',
      })).data,
    onSuccess: () => q.refetch(),
  });

  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 24, fontWeight: '800' }}>
        {contract?.subject?.title || 'Contrato'}
      </Text>
      <Text style={{ fontSize: 16, opacity: 0.8, marginTop: 8 }}>
        Status: {state}
      </Text>

      <View style={{ marginTop: 16 }}>
        <BigButton
          icon="✅"
          title="Aceitar Contrato"
          subtitle="Um toque para aceitar e registrar no sistema"
          onPress={() => accept.mutate()}
          disabled={contract?.acceptance?.accepted === true}
        />

        <BigButton
          icon="🚚"
          title="Confirmar Expedição"
          subtitle="Tirar foto + GPS (obrigatório) e confirmar envio"
          onPress={() => navigation.navigate('StageDispatch', {
contractId: id })}
          disabled={state !== 'CREATED' ||
contract?.acceptance?.accepted !== true}
        />

        <BigButton
          icon="📦"
          title="Confirmar Entrega"
          subtitle="Foto + GPS no local de entrega"
          onPress={() => navigation.navigate('StageDelivery', {
contractId: id })}
          disabled={state !== 'DISPATCH_CONFIRMED'}
        />

        <BigButton
          icon="📄"
          title="Documento Fiscal (NF-e)"
          subtitle="Ler chave por câmera, enviar XML/PDF e verificar"
          onPress={() => navigation.navigate('FiscalDoc', {
contractId: id })}
          disabled={state !== 'DELIVERY_CONFIRMED'}
        />

        <BigButton
          icon="🎙️"
          title="Solicitar Visita Técnica"
          subtitle="Pedir apoio e registrar no contrato"
```

```
          onPress={() => navigation.navigate('RequestTechnician', {
contractId: id })}
        />

        <BigButton
          icon="⚠️"
          title="Cancelar (pode ter multa)"
          subtitle="Se pago, aplica multa automaticamente"
          onPress={() => navigation.navigate('CancelContract', {
contractId: id })}
        />
      </View>
    </View>
  );
}
```

---

5) Tela: Confirmar Expedição (evidência Tipo B)

Objetivo: operador só faz 2 passos: “tirar foto” → “confirmar”.

```
// src/screens/StageDispatchScreen.tsx
import React, { useState } from 'react';
import { Text, View } from 'react-native';
import { BigButton } from '../ui/BigButton';
import { api } from '../services/api';
import { v4 as uuid } from 'uuid';

export function StageDispatchScreen({ route, navigation }: any) {
  const { contractId } = route.params;
  const [photoUri, setPhotoUri] = useState<string | null>(null);
  const [geo, setGeo] = useState<{ lat: number; lng: number } |
null>(null);

  async function capturePhotoAndGeo() {
    // aqui você integra expo-camera + expo-location
    // setPhotoUri(uri); setGeo({lat,lng});
    setPhotoUri('file://photo.jpg');
    setGeo({ lat: -15.6, lng: -56.1 });
  }

  async function submit() {
    // 1) anexar evidência B (foto+gps)
    await api.post(`/api/contracts/${contractId}/evidence`, {
      clientRequestId: uuid(),
      type: 'PHOTO',
      riskType: 'B',
      geo,
      note: 'Expedição - foto comprovante',
    });

    // 2) transicionar estado (DISPATCH_CONFIRMED)
    await api.post(`/api/contracts/${contractId}/transition`, {
      clientRequestId: uuid(),
      to: 'DISPATCH_CONFIRMED',
    });

    navigation.goBack();
  }
```

```
  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 24, fontWeight: '800' }}>Confirmar
Expedição</Text>
      <Text style={{ fontSize: 16, opacity: 0.8, marginTop: 8 }}>
        Faça uma foto e confirme. O sistema exige GPS.
      </Text>

      <View style={{ marginTop: 16 }}>
        <BigButton
          icon="📸"
          title={photoUri ? 'Foto capturada' : 'Tirar Foto'}
          subtitle={geo ? `GPS ok (${geo.lat.toFixed(2)},
${geo.lng.toFixed(2)})` : 'GPS será capturado junto'}
          onPress={capturePhotoAndGeo}
        />

        <BigButton
          icon="🚚"
          title="Confirmar Expedição"
          subtitle="Registra prova e avança etapa"
          onPress={submit}
          disabled={!photoUri || !geo}
        />
      </View>
    </View>
  );
}
```

A tela de **Entrega** é igual, mudando o `note` e o `to: DELIVERY_CONFIRMED`.

---

6) Tela: Documento Fiscal (NF-e) — chave por câmera + XML/PDF

Esta é a parte crítica do seu requisito.

UX (sem digitar):

- Botão "**Ler Chave por Câmera**" (escaneia QR do DANFE ou reconhece números)
- Botão "Enviar XML"
- Botão "Enviar PDF"
- Botão "Verificar NF-e" (só habilita quando XML foi enviado)

```
// src/screens/FiscalDocScreen.tsx
import React, { useState } from 'react';
import { Text, View } from 'react-native';
import { BigButton } from '../ui/BigButton';
import { api } from '../services/api';
import { v4 as uuid } from 'uuid';

export function FiscalDocScreen({ route, navigation }: any) {
  const { contractId } = route.params;
  const [fiscalKey, setFiscalKey] = useState<string>('');
```

```
  const [xmlUploaded, setXmlUploaded] = useState(false);
  const [pdfUploaded, setPdfUploaded] = useState(false);

  async function scanKey() {
    // integrar expo-camera + leitura QR do DANFE
    // fallback: foto do DANFE + IA extrai

setFiscalKey('35190123456789012345678901234567890123456789'.slice(0,
44));
  }

  async function upload(type: 'XML'|'PDF') {
    if (!fiscalKey || fiscalKey.length !== 44) return;

    // 1) pede URL assinada
    const up = await
api.post(`/api/contracts/${contractId}/fiscal/upload-
url?fiscalKey=${fiscalKey}`, { type });
    const { url } = up.data;

    // 2) aqui o app faz PUT direto para o Storage com o arquivo
selecionado
    // Ex: await fetch(url, { method:'PUT', headers:{'Content-Type':
...}, body:fileBytes })
    // (você integrará expo-document-picker)
    if (type === 'XML') setXmlUploaded(true);
    if (type === 'PDF') setPdfUploaded(true);
  }

  async function commit() {
    await api.post(`/api/contracts/${contractId}/fiscal/commit`, {
      clientRequestId: uuid(),
      fiscalKey,
      xmlUploaded,
      pdfUploaded,
    });
  }

  async function verifyAndFinish() {
    await api.post(`/api/contracts/${contractId}/fiscal/verify`, {
      clientRequestId: uuid(),
    });

    // transiciona estado fiscal
    await api.post(`/api/contracts/${contractId}/transition`, {
      clientRequestId: uuid(),
      to: 'FISCAL_DOC_VERIFIED',
    });

    navigation.goBack();
  }

  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 24, fontWeight: '800' }}>Documento
Fiscal</Text>
      <Text style={{ fontSize: 16, opacity: 0.8, marginTop: 8 }}>
        A foto ajuda, mas o sistema só libera com XML ou chave
validável.
      </Text>
```

```
      <View style={{ marginTop: 16 }}>
        <BigButton
          icon="🔎"
          title={fiscalKey ? 'Chave lida ✅' : 'Ler Chave por Câmera'}
          subtitle={fiscalKey ? `Chave: ${fiscalKey.slice(0,
6)}...${fiscalKey.slice(-6)}` : 'Aponte para o QR do DANFE'}
          onPress={scanKey}
        />

        <BigButton
          icon="📄"
          title={xmlUploaded ? 'XML enviado ✅' : 'Enviar XML'}
          subtitle="Obrigatório para verificação forte"
          onPress={() => upload('XML')}
          disabled={!fiscalKey || fiscalKey.length !== 44}
        />

        <BigButton
          icon="📑"
          title={pdfUploaded ? 'PDF enviado ✅' : 'Enviar PDF
(opcional)'}
          subtitle="Ajuda na conferência (não substitui o XML)"
          onPress={() => upload('PDF')}
          disabled={!fiscalKey || fiscalKey.length !== 44}
        />

        <BigButton
          icon="💾"
          title="Salvar no Contrato"
          subtitle="Registra no sistema com auditoria"
          onPress={commit}
          disabled={!fiscalKey || !xmlUploaded}
        />

        <BigButton
          icon="✅"
          title="Verificar NF-e e Finalizar"
          subtitle="Libera pagamento final (tranche)"
          onPress={verifyAndFinish}
          disabled={!xmlUploaded}
        />
      </View>
    </View>
  );
}
```

---

7) Registro por voz e foto (baixa leitura)

7.1 Tela “Registrar por Foto”

- botão grande: “Foto de anotação”
- botão grande: “Falar e registrar”
- sistema cria “draft” e pede só “Confirmar”

Isso alimenta `contract_evidence` e/ou `stock_movements` com lastro.

7.2 Voz (comandos simples)

Comandos:

- "**Registrar retirada de sal**"
- "**Confirmar entrega**"
- "**Pedir técnico**"
- "**Ler nota fiscal**"

No app você pode usar:

- `expo-speech` para falar
- `expo-av` para gravar áudio como evidência
- (opcional) backend para transcrever

---

8) Regras de usabilidade (o que faz funcionar no campo)

- Sempre mostrar:
  - ✅ "OK"
  - ⚠️ "Falta 1 passo"
  - ❌ "Bloqueado — motivo"
- Nunca mostrar erro técnico cru; sempre "ação sugerida"
- Operação offline:
  - salva "fila" local
  - quando voltar internet, envia evidências e transições

---

9) Microcopy (mensagens simples)

- "**Tire a foto. O GPS será salvo junto.**"
- "**Envie o XML para liberar pagamento.**"
- "**Bloqueado: propriedade com restrição (CAR/IE).**"
- "**Cancelamento após pagamento tem multa.**"

Se você quiser, eu também te mando agora:

- **telas do Marketplace de insumos** com alternância "produções" e "compra/venda/leilão"
- **tela do estoque** para operador dar baixa (por foto/voz)
- **modo offline** (fila local + sync) no Expo + React Query

Beleza — abaixo estão **as 3 frentes** que você pediu, já no modelo "melhor para campo":

1. **Marketplace de insumos** (alternância por produção + compra/venda + leilão + visita técnica)
2. **Portal / App de estoque** (baixa por foto/voz + rastreio por identidade/NF/código)
3. **Modo offline** (fila local + sincronização com React Query)

Vou entregar em **blocos de código** prontos para você plugar no Expo + React Native.

---

1) Marketplace de insumos (mobile)

1.1 Conceito de UX (baixa leitura)

Topo com 3 controles grandes:

- **Produção**: Bovinos | Leite | Aves | Suínos | Peixes | Plantio | Silvicultura
- **Ação**: Comprar | Vender | Leilão
- **Região**: "Perto de mim" (auto-expande)

E um botão fixo:

- 🎙️ **Pedir visita técnica**

1.2 Tela Marketplace (tabs grandes)

```
// src/screens/MarketScreen.tsx
import React, { useMemo, useState } from 'react';
import { FlatList, Text, View } from 'react-native';
import { BigButton } from '../ui/BigButton';
import { Card } from '../ui/Card';
import { useQuery } from '@tanstack/react-query';
import { api } from '../services/api';

type Production =
  | 'BOVINOS' | 'LEITE' | 'AVES' | 'SUINOS' | 'PEIXES' | 'PLANTIO' |
'SILVICULTURA';

type Mode = 'COMPRAR' | 'VENDER' | 'LEILAO';

const productions: { key: Production; label: string; icon: string }[]
= [
  { key: 'BOVINOS', label: 'Bovinos', icon: '🐄' },
  { key: 'LEITE', label: 'Leite', icon: '🥛' },
  { key: 'AVES', label: 'Aves', icon: '🐔' },
  { key: 'SUINOS', label: 'Suínos', icon: '🐖' },
  { key: 'PEIXES', label: 'Peixes', icon: '🐟' },
  { key: 'PLANTIO', label: 'Plantio', icon: '🌱' },
  { key: 'SILVICULTURA', label: 'Silvicultura', icon: '🌴' },
];

const modes: { key: Mode; label: string; icon: string }[] = [
  { key: 'COMPRAR', label: 'Comprar', icon: '🛒' },
```

```
  { key: 'VENDER', label: 'Vender', icon: '🏷️' },
  { key: 'LEILAO', label: 'Leilão', icon: '🔨' },
];

export function MarketScreen({ navigation }: any) {
  const [production, setProduction] = useState<Production>('BOVINOS');
  const [mode, setMode] = useState<Mode>('COMPRAR');

  // MVP: geo fixo; plugar expo-location
  const geo = useMemo(() => ({ lat: -15.6, lng: -56.1 }), []);

  const offers = useQuery({
    queryKey: ['marketOffers', production, mode, geo.lat, geo.lng],
    queryFn: async () => {
      const res = await api.get('/api/market/offers', {
        params: {
          lat: geo.lat,
          lng: geo.lng,
          radiusKm: 50,
          maxRadiusKm: 800,
          expand: true,
          production,
          mode,
        },
      });
      return res.data;
    },
  });

  const items = offers.data?.items || [];

  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 24, fontWeight: '800', marginBottom: 12
}}>
        Marketplace de Insumos
      </Text>

      {/* Produção */}
      <Card>
        <Text style={{ fontSize: 18, fontWeight: '700', marginBottom:
8 }}>Produção</Text>
        <FlatList
          horizontal
          data={productions}
          keyExtractor={(it) => it.key}
          showsHorizontalScrollIndicator={false}
          renderItem={({ item }) => (
            <View style={{ marginRight: 10 }}>
              <BigButton
                icon={item.icon}
                title={item.label}
                subtitle={production === item.key ? 'Selecionado' :
''}
                onPress={() => setProduction(item.key)}
              />
            </View>
          )}
        />
      </Card>
```

```
      {/* Ação */}
      <Card>
        <Text style={{ fontSize: 18, fontWeight: '700', marginBottom:
8 }}>Ação</Text>
        <View style={{ flexDirection: 'row', gap: 10 }}>
          {modes.map((m) => (
            <View key={m.key} style={{ flex: 1 }}>
              <BigButton
                icon={m.icon}
                title={m.label}
                subtitle={mode === m.key ? 'Selecionado' : ''}
                onPress={() => setMode(m.key)}
              />
            </View>
          ))}
        </View>
      </Card>

      {/* Lista de ofertas */}
      <Text style={{ fontSize: 18, fontWeight: '700', marginVertical:
10 }}>
        Ofertas perto de você
      </Text>

      <FlatList
        data={items}
        keyExtractor={(it) => it.offerId}
        refreshing={offers.isFetching}
        onRefresh={() => offers.refetch()}
        renderItem={({ item }) => (
          <BigButton
            icon="📦"
            title={`${item.title}`}
            subtitle={`R$ ${item.price} • ${item.uom} •
${item.supplierName} • ${item.distanceKm} km`}
            onPress={() => navigation.navigate('OfferDetail', {
offerId: item.offerId, production, mode })}
          />
        )}
      />

      {/* CTA visita técnica */}
      <View style={{ marginTop: 10 }}>
        <BigButton
          icon="🎙️"
          title="Solicitar visita técnica"
          subtitle="Pedir apoio e registrar no sistema"
          onPress={() => navigation.navigate('RequestTechnician', {
production })}
        />
      </View>
    </View>
  );
}
```

Backend esperado

- GET /api/market/offers?lat&lng&radiusKm&maxRadiusKm&expand&productio n&mode

    - filtra por `productionProfile` e `saleMode` (`DIRECT|AUCTION`)
    - aplica raio adaptativo

1.3 Tela Oferta (reserva → escrow)

Na tela de oferta:

- Botão grande "Reservar e pagar (PIX/Boleto)"
- Mostra link do Asaas (invoiceUrl)
- Aguardar `status=HELD`
- Botão "Confirmar pedido (gera contrato)"

---

2) Estoque (baixa por foto/voz) + identidade do produto

2.1 Regra-base de rastreio (como você exigiu)

Todo produto precisa carregar uma identidade única, por qualquer rastro:

- `productIdentityId` obrigatório no cadastro (pode vir de:
    - código interno, GTIN, SKU, QR, RFID
    - chave NF-e/itens (rastreio por NF)
    - serial/lote

)

      E toda movimentação salva:
- `identityRef` (qual foi a fonte do rastro)
- evidência (foto do caderno, voz, QR etc.)

2.2 Tela "Baixa de estoque" super simples

Fluxo:

1. escolher item (ou ler QR)
2. informar quantidade (pode ser por voz)
3. anexar foto do caderno (opcional)
4. confirmar (o sistema valida via RulesEngine)

```
// src/screens/StockQuickMoveScreen.tsx
import React, { useState } from 'react';
import { Text, View } from 'react-native';
import { BigButton } from '../ui/BigButton';
import { api } from '../services/api';
import { v4 as uuid } from 'uuid';

export function StockQuickMoveScreen({ navigation }: any) {
  const [itemId, setItemId] = useState<string | null>(null);
  const [qty, setQty] = useState<number>(0);
```

```
  const [photoUri, setPhotoUri] = useState<string | null>(null);

  async function scanItem() {
    // expo-barcode-scanner / NFC / RFID
    setItemId('item_salt_25kg');
  }

  async function capturePhoto() {
    // expo-camera
    setPhotoUri('file://note.jpg');
  }

  async function voiceQty() {
    // expo-av grava voz; backend transcreve e sugere qty
    setQty(1);
  }

  async function submit() {
    if (!itemId || qty <= 0) return;

    // 1) evidência opcional (foto do caderno) - prova
    if (photoUri) {
      await api.post('/api/evidence', {
        clientRequestId: uuid(),
        type: 'PHOTO',
        riskType: 'A',
        note: 'Baixa estoque - anotação manual',
      });
    }

    // 2) movimento de estoque (imutável)
    await api.post('/api/movements', {
      clientRequestId: uuid(),
      itemId,
      qty,
      movementType: 'OUT',
      reason: 'CONSUMPTION',
      identityRef: { kind: 'CODE', value: itemId }, // pode ser
NF/código/lote
    });

    navigation.goBack();
  }

  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 24, fontWeight: '800' }}>Baixa de
estoque</Text>
      <Text style={{ fontSize: 16, opacity: 0.8, marginTop: 8 }}>
        Rápido: escaneie o produto, informe quantidade e confirme.
      </Text>

      <View style={{ marginTop: 16 }}>
        <BigButton
          icon="🔎"
          title={itemId ? 'Produto selecionado ✅' : 'Ler código do
produto'}
          subtitle={itemId ? itemId : 'QR/NFC/RFID'}
          onPress={scanItem}
        />
```

```
        <BigButton
          icon="🎙️"
          title="Falar quantidade"
          subtitle={qty > 0 ? `Quantidade: ${qty}` : 'Diga: "retirar 2
sacos"'}
          onPress={voiceQty}
          disabled={!itemId}
        />

        <BigButton
          icon="📸"
          title={photoUri ? 'Foto anexada ✅' : 'Foto do caderno
(opcional)'}
          subtitle="Ajuda a lançar sem digitar"
          onPress={capturePhoto}
          disabled={!itemId}
        />

        <BigButton
          icon="✅"
          title="Confirmar baixa"
          subtitle="O sistema registra e atualiza saldo"
          onPress={submit}
          disabled={!itemId || qty <= 0}
        />
      </View>
    </View>
  );
}
```

---

3) Offline-first (fila local + sync com React Query)

Objetivo

- Se estiver sem internet:
  - salva ações em uma fila (AsyncStorage)
  - o usuário vê "pendente"
- Quando voltar:
  - envia na ordem
  - idempotência evita duplicar no backend

3.1 Queue simples

```
// src/offline/queue.ts
import AsyncStorage from '@react-native-async-storage/async-storage';
import NetInfo from '@react-native-community/netinfo';
import { api } from '../services/api';

type QueueItem = {
  id: string;
  createdAt: number;
  request: { method: 'POST'|'PUT'|'PATCH'; url: string; body: any };
};

const KEY = 'offline_queue_v1';
```

```
export async function enqueue(item: QueueItem) {
  const raw = await AsyncStorage.getItem(KEY);
  const list: QueueItem[] = raw ? JSON.parse(raw) : [];
  list.push(item);
  await AsyncStorage.setItem(KEY, JSON.stringify(list));
}

export async function flushQueue() {
  const state = await NetInfo.fetch();
  if (!state.isConnected) return { ok: false, reason: 'OFFLINE' };

  const raw = await AsyncStorage.getItem(KEY);
  const list: QueueItem[] = raw ? JSON.parse(raw) : [];

  const remaining: QueueItem[] = [];
  for (const it of list) {
    try {
      await api.request({ method: it.request.method, url:
it.request.url, data: it.request.body });
    } catch (e) {
      // mantém na fila se falhar (pode ser erro temporário)
      remaining.push(it);
    }
  }

  await AsyncStorage.setItem(KEY, JSON.stringify(remaining));
  return { ok: true, sent: list.length - remaining.length, remaining:
remaining.length };
}
```

3.2 Helper para “send or queue”

```
// src/offline/sendOrQueue.ts
import { v4 as uuid } from 'uuid';
import NetInfo from '@react-native-community/netinfo';
import { api } from '../services/api';
import { enqueue } from './queue';

export async function sendOrQueue(method: 'POST'|'PUT'|'PATCH', url:
string, body: any) {
  const state = await NetInfo.fetch();
  if (state.isConnected) {
    return api.request({ method, url, data: body });
  }
  await enqueue({
    id: uuid(),
    createdAt: Date.now(),
    request: { method, url, body }
  });
  return { data: { queued: true } } as any;
}
```

3.3 Disparar flush automático

No `App.tsx`:

```
import React, { useEffect } from 'react';
import NetInfo from '@react-native-community/netinfo';
import { flushQueue } from './src/offline/queue';

export default function App() {
  useEffect(() => {
```

```
    const unsub = NetInfo.addEventListener((s) => {
      if (s.isConnected) flushQueue();
    });
    return () => unsub();
  }, []);

  return null; // seu NavigationContainer aqui
}
```

---

4) Leilão (módulo separado, como você definiu)

Você disse:

- Cadastro de leilão só para **ativos não comprometidos**
- Gerenciado por **leiloeiro credenciado**
- No leilão, é "livre"; após lance vencedor → **escrow obrigatório**
- O fluxo de contrato e marcos vale igual

✅ Então no app:

- No marketplace, modo "Leilão" abre lista de lotes
- Botão "Dar lance"
- Ao vencer: gera cobrança escrow e só depois trava/contrata

---

5) Próximo passo prático (eu já deixo pronto)

Para o app "funcionar de ponta a ponta", só faltam 3 endpoints de leitura:

- `GET /api/market/offers` (lista com filtros)
- `GET /api/contracts` (lista)
- `GET /api/contracts/:id` (detalhe)

Se você quiser, eu já te mando esses 3 endpoints (NestJS + Firestore query + filtros + segurança tenant/role) no próximo envio.

Perfeito — abaixo vão os **3 endpoints de leitura** (NestJS + Firestore), com:

- multi-tenant (`tenantId`)
- RBAC/permissões
- filtros do marketplace (produção, modo, raio adaptativo)
- retorno já "pronto para o app" (com `distanceKm`)

---

0) Premissas de dados (mínimo)

`market_offers/{offerId}`

Campos esperados:

- `offerId`
- `supplierTenantId`
- `supplierName`
- `title`, `uom`, `price`
- `productionProfile: "BOVINOS" | "LEITE" | ...`
- `saleMode: "DIRECT" | "AUCTION"`
- `active: boolean`

`market_companies/{companyId}`

Campos:

- `tenantId` (supplierTenantId)
- `geo: { lat, lng }`
- `marketEnabled: boolean`
- `serviceRadiusKm?: number` (opcional, default 50)

`market_offer_stock/{offerId}`

Campos:

- `qtyAvailable`
- `qtyReserved`

`contracts/{contractId}`

Campos:

- `buyerTenantId`
- `supplierTenantId`
- `tenantId` (normalmente buyerTenantId)
- `state`
- `subject` (title/qty/total)
- `acceptance.accepted`

---

1) GET /api/market/offers (lista + raio adaptativo)

Query

`lat,lng,radiusKm,maxRadiusKm,expand,production,mode`

- `mode: COMPRAR|VENDER|LEILAO`
  - aqui vamos mapear:
    - `LEILAO` -> `saleMode=AUCTION`
    - `COMPRAR` -> `saleMode=DIRECT` (para buyer)
    - `VENDER` -> (opcional) listar “minhas ofertas” — deixo tratado como DIRECT também, mas você pode separar

Implementação

```
// src/market/market-read.controller.ts
import { Controller, Get, Query, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { MarketReadService } from './market-read.service';

@Controller('/api/market')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class MarketReadController {
  constructor(private readonly svc: MarketReadService) {}

  @Get('/offers')

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','SUPPLIER'
)
  @Permissions('market.read')
  async listOffers(
    @Actor() actor: any,
    @Query('lat') lat: string,
    @Query('lng') lng: string,
    @Query('radiusKm') radiusKm?: string,
    @Query('maxRadiusKm') maxRadiusKm?: string,
    @Query('expand') expand?: string,
    @Query('production') production?: string,
    @Query('mode') mode?: string,
  ) {
    return this.svc.listOffersNear(actor, {
      lat: Number(lat),
      lng: Number(lng),
      radiusKm: radiusKm ? Number(radiusKm) : 50,
      maxRadiusKm: maxRadiusKm ? Number(maxRadiusKm) : 800,
      expand: expand === 'true' || expand === '1',
      production: production ?? null,
      mode: (mode as any) ?? 'COMPRAR',
    });
  }
}
// src/market/market-read.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

function haversineKm(a: {lat:number,lng:number},
b:{lat:number,lng:number}) {
  const R = 6371;
  const dLat = (b.lat - a.lat) * Math.PI/180;
  const dLng = (b.lng - a.lng) * Math.PI/180;
  const s1 = Math.sin(dLat/2)**2;
  const s2 =
Math.cos(a.lat*Math.PI/180)*Math.cos(b.lat*Math.PI/180)*Math.sin(dLng/
2)**2;
  const c = 2 * Math.asin(Math.sqrt(s1+s2));
  return R*c;
}
```

```
function mapSaleMode(mode: string) {
  if (mode === 'LEILAO') return 'AUCTION';
  return 'DIRECT';
}

@Injectable()
export class MarketReadService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async listOffersNear(
    actor: any,
    input: {
      lat: number; lng: number;
      radiusKm: number; maxRadiusKm: number;
      expand: boolean;
      production: string | null;
      mode: 'COMPRAR'|'VENDER'|'LEILAO';
    }
  ) {
    const center = { lat: input.lat, lng: input.lng };
    const saleMode = mapSaleMode(input.mode);

    // 1) pega empresas marketplace habilitadas (marketEnabled = true)
    // MVP: busca por marketEnabled e depois filtra por distância
(Firestore não faz geo nativo).
    const companiesSnap = await this.db.collection('market_companies')
      .where('marketEnabled', '==', true)
      .limit(500)
      .get();

    // 2) raio adaptativo: aumenta até achar pelo menos N
empresas/ofertas
    const step = 50;
    let radius = Math.max(5, input.radiusKm);
    const maxR = Math.max(radius, input.maxRadiusKm);
    const minCompanies = 5;

    let nearbyCompanies: any[] = [];
    while (radius <= maxR) {
      const list: any[] = [];
      companiesSnap.forEach((d) => {
        const c = d.data();
        if (!c.geo?.lat || !c.geo?.lng) return;
        const dist = haversineKm(center, { lat: c.geo.lat, lng:
c.geo.lng });
        if (dist <= radius) list.push({ ...c, companyId: d.id,
distanceKm: Math.round(dist*10)/10 });
      });
      nearbyCompanies = list.sort((a,b)=>a.distanceKm-b.distanceKm);
      if (nearbyCompanies.length >= minCompanies) break;
      radius += step;
    }

    // 3) lista supplierTenantIds próximos
    const supplierTenantIds = nearbyCompanies.map(c => c.tenantId);

    if (supplierTenantIds.length === 0) {
      return { radiusKm: radius, items: [] };
    }
```

```
    // 4) Firestore IN tem limite (10). Faz em batches.
    const batches: string[][] = [];
    for (let i=0; i<supplierTenantIds.length; i+=10)
batches.push(supplierTenantIds.slice(i, i+10));

    const offers: any[] = [];

    for (const batch of batches) {
      let q: FirebaseFirestore.Query =
this.db.collection('market_offers')
        .where('active', '==', true)
        .where('saleMode', '==', saleMode)
        .where('supplierTenantId', 'in', batch);

      if (input.production) {
        q = q.where('productionProfile', '==', input.production);
      }

      const snap = await q.limit(200).get();
      snap.forEach((d) => offers.push({ ...d.data(), offerId: d.id
}));
    }

    // 5) se expand=true, junta stock e calcula distância pelo
supplier
    let stockMap: Record<string, any> = {};
    if (input.expand) {
      // stocks por offerId
      const offerIds = offers.map(o => o.offerId);
      // batch read em grupos de 200
      for (let i=0; i<offerIds.length; i+=200) {
        const ids = offerIds.slice(i, i+200);
        const snaps = await this.db.getAll(...ids.map(id =>
this.db.collection('market_offer_stock').doc(id)));
        snaps.forEach((s) => {
          if (!s.exists) return;
          stockMap[s.id] = s.data();
        });
      }
    }

    const companyByTenant: Record<string, any> = {};
    nearbyCompanies.forEach(c => companyByTenant[c.tenantId] = c);

    const items = offers.map((o) => {
      const company = companyByTenant[o.supplierTenantId];
      const st = stockMap[o.offerId];
      const qtyAvailable = Number(st?.qtyAvailable ?? 0);
      const qtyReserved = Number(st?.qtyReserved ?? 0);
      const free = Math.max(0, qtyAvailable - qtyReserved);

      return {
        offerId: o.offerId,
        title: o.title,
        uom: o.uom,
        price: o.price,
        productionProfile: o.productionProfile,
        saleMode: o.saleMode,

        supplierTenantId: o.supplierTenantId,
```

```
        supplierName: o.supplierName ?? company?.name ?? 'Fornecedor',
        distanceKm: company?.distanceKm ?? null,

        stock: input.expand ? { qtyAvailable, qtyReserved, free } :
undefined,
      };
    }).sort((a,b)=>(a.distanceKm ?? 9999)-(b.distanceKm ?? 9999));

    return { radiusKm: radius, items };
  }
}
```

✅ Isso atende o seu requisito: **se não tiver empresa próxima, expande raio até encontrar** (ou até `maxRadiusKm`).

---

2) GET /api/contracts (lista por tenant + papéis)

Regras:

- Buyer vê contratos onde `buyerTenantId == actor.tenantId`
- Supplier vê contratos onde `supplierTenantId == actor.tenantId`
- Auditor pode ver ambos (dependendo do seu RBAC — aqui deixo liberado para o mesmo tenant)

```
// src/contracts/contracts-read.controller.ts
import { Controller, Get, Query, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { ContractsReadService } from './contracts-read.service';

@Controller('/api/contracts')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class ContractsReadController {
  constructor(private readonly svc: ContractsReadService) {}

  @Get()

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','SUPPLIER'
,'AUDITOR','LOGISTICS','INVESTOR')
  @Permissions('contracts.read')
  async list(@Actor() actor: any, @Query('state') state?: string) {
    return this.svc.listForTenant(actor, { state: state ?? null,
limit: 50 });
  }
}
// src/contracts/contracts-read.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

@Injectable()
```

```
export class ContractsReadService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async listForTenant(actor: any, input: { state: string | null;
limit: number }) {
    // buyer side
    let qBuyer: FirebaseFirestore.Query =
this.db.collection('contracts')
      .where('buyerTenantId', '==', actor.tenantId);

    // supplier side
    let qSupplier: FirebaseFirestore.Query =
this.db.collection('contracts')
      .where('supplierTenantId', '==', actor.tenantId);

    if (input.state) {
      qBuyer = qBuyer.where('state', '==', input.state);
      qSupplier = qSupplier.where('state', '==', input.state);
    }

    const [bSnap, sSnap] = await Promise.all([
      qBuyer.orderBy('updatedAt', 'desc').limit(input.limit).get(),
      qSupplier.orderBy('updatedAt', 'desc').limit(input.limit).get(),
    ]);

    // merge unique
    const map: Record<string, any> = {};
    bSnap.forEach((d) => map[d.id] = { ...d.data(), contractId: d.id
});
    sSnap.forEach((d) => map[d.id] = { ...d.data(), contractId: d.id
});

    const items =
Object.values(map).sort((a:any,b:any)=>(b.updatedAt??0)-
(a.updatedAt??0));

    return {
      items: items.map((c:any) => ({
        contractId: c.contractId,
        state: c.state,
        subject: c.subject,
        acceptance: c.acceptance,
        buyerTenantId: c.buyerTenantId,
        supplierTenantId: c.supplierTenantId,
        updatedAt: c.updatedAt,
      }))
    };
  }
}
```

---

3) GET /api/contracts/:id (detalhe com ACL)

```
// src/contracts/contracts-read-detail.controller.ts
import { Controller, Get, Param, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
```

```
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { ContractsDetailService } from './contracts-detail.service';

@Controller('/api/contracts')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class ContractsDetailController {
  constructor(private readonly svc: ContractsDetailService) {}

  @Get('/:id')

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','SUPPLIER'
,'AUDITOR','LOGISTICS','INVESTOR')
  @Permissions('contracts.read')
  async getOne(@Actor() actor: any, @Param('id') id: string) {
    return this.svc.getOne(actor, id);
  }
}
// src/contracts/contracts-detail.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

@Injectable()
export class ContractsDetailService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async getOne(actor: any, contractId: string) {
    const ref = this.db.collection('contracts').doc(contractId);
    const snap = await ref.get();
    if (!snap.exists) throw new Error('CONTRACT_NOT_FOUND');

    const c = snap.data()!;
    const allowed =
      c.buyerTenantId === actor.tenantId ||
      c.supplierTenantId === actor.tenantId;

    if (!allowed) throw new Error('CONTRACT_FORBIDDEN');

    return { ...c, contractId };
  }
}
```

---

4) Registrar os módulos (Market + Contracts)

No `AppModule`:

```
imports: [
  // ...
  MarketModule,
  ContractsModule,
]
```

E nos módulos, inclua:

- `MarketReadController + MarketReadService`
- `ContractsReadController + ContractsReadService`

- `ContractsDetailController + ContractsDetailService`

---

5) Índices Firestore recomendados (para não travar)

Crie índices compostos:

- `contracts`:
  - `buyerTenantId + updatedAt(desc)`
  - `supplierTenantId + updatedAt(desc)`
  - (opcional) `buyerTenantId + state + updatedAt(desc)`
- `market_offers`:
  - `supplierTenantId + saleMode + active`
  - `supplierTenantId + saleMode + active + productionProfile`

Se você quiser, eu também te mando imediatamente:

- `GET /api/market/offers/:id` (detalhe da oferta + stock + fornecedor)
- `GET /api/market/reservations/:id` (para o app "aguardar HELD" sem polling pesado)
- Um **listener realtime** no app (Firestore) *ou* um endpoint `status` mais leve (melhor para segurança porque as rules ficam fechadas).

Perfeito — vou te entregar **os 3 endpoints restantes** do jeito mais prático pro app:

1. `GET /api/market/offers/:id` → detalhe + stock + fornecedor + distância
2. `GET /api/market/reservations/:id` → status leve (para "aguardar HELD")
3. `GET /api/market/reservations/:id/status` → **bem leve** (ideal para polling rápido)

Além disso, vou te passar um padrão de **polling inteligente** no app (sem gastar demais).

---

1) GET /api/market/offers/:id (detalhe completo)

Controller

```
// src/market/market-offers-read.controller.ts
import { Controller, Get, Param, Query, UseGuards } from
'@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
```

```
import { MarketOffersReadService } from './market-offers-
read.service';

@Controller('/api/market/offers')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class MarketOffersReadController {
  constructor(private readonly svc: MarketOffersReadService) {}

  @Get('/:id')

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','SUPPLIER'
,'AUDITOR','LOGISTICS','INVESTOR')
  @Permissions('market.read')
  async getOne(
    @Actor() actor: any,
    @Param('id') offerId: string,
    @Query('lat') lat?: string,
    @Query('lng') lng?: string,
  ) {
    return this.svc.getOffer(actor, offerId, {
      lat: lat ? Number(lat) : null,
      lng: lng ? Number(lng) : null,
    });
  }
}
```

Service

```
// src/market/market-offers-read.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

function haversineKm(a: {lat:number,lng:number},
b:{lat:number,lng:number}) {
  const R = 6371;
  const dLat = (b.lat - a.lat) * Math.PI/180;
  const dLng = (b.lng - a.lng) * Math.PI/180;
  const s1 = Math.sin(dLat/2)**2;
  const s2 =
Math.cos(a.lat*Math.PI/180)*Math.cos(b.lat*Math.PI/180)*Math.sin(dLng/
2)**2;
  const c = 2 * Math.asin(Math.sqrt(s1+s2));
  return R*c;
}

@Injectable()
export class MarketOffersReadService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async getOffer(actor: any, offerId: string, geo: {lat:number|null,
lng:number|null}) {
    const offerRef = this.db.collection('market_offers').doc(offerId);
    const stockRef =
this.db.collection('market_offer_stock').doc(offerId);

    const [offerSnap, stockSnap] = await Promise.all([offerRef.get(),
stockRef.get()]);
    if (!offerSnap.exists) throw new Error('OFFER_NOT_FOUND');
```

```
    const offer = offerSnap.data()!;
    if (offer.active !== true) throw new Error('OFFER_INACTIVE');

    // supplier/company
    const companySnap = await this.db.collection('market_companies')
      .where('tenantId', '==', offer.supplierTenantId)
      .limit(1)
      .get();

    const company = companySnap.empty ? null :
companySnap.docs[0].data();

    const st = stockSnap.exists ? stockSnap.data()! : { qtyAvailable:
0, qtyReserved: 0 };
    const qtyAvailable = Number(st.qtyAvailable ?? 0);
    const qtyReserved = Number(st.qtyReserved ?? 0);
    const free = Math.max(0, qtyAvailable - qtyReserved);

    let distanceKm: number | null = null;
    if (geo.lat != null && geo.lng != null && company?.geo?.lat &&
company?.geo?.lng) {
      distanceKm = Math.round(haversineKm({ lat: geo.lat, lng: geo.lng
}, { lat: company.geo.lat, lng: company.geo.lng }) * 10) / 10;
    }

    return {
      offerId,
      title: offer.title,
      description: offer.description ?? null,
      uom: offer.uom,
      price: offer.price,
      productionProfile: offer.productionProfile,
      saleMode: offer.saleMode,
      active: offer.active,

      supplierTenantId: offer.supplierTenantId,
      supplierName: offer.supplierName ?? company?.name ??
'Fornecedor',
      supplierGeo: company?.geo ?? null,
      distanceKm,

      stock: { qtyAvailable, qtyReserved, free },
      terms: offer.terms ?? null,
      updatedAt: offer.updatedAt ?? null
    };
  }
}
```

---

2) GET /api/market/reservations/:id (detalhe seguro)

Você usa isso para:

- exibir boleto/pix (invoiceUrl)
- exibir tempo restante (`expiresAt`)
- status (`ESCROW_PENDING`, `HELD`, `CANCELLED`, `FAILED_OUT_OF_STOCK`)

Controller

```
// src/market/market-reservations-read.controller.ts
import { Controller, Get, Param, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { MarketReservationsReadService } from './market-reservations-
read.service';

@Controller('/api/market/reservations')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class MarketReservationsReadController {
  constructor(private readonly svc: MarketReservationsReadService) {}

  @Get('/:id')

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','SUPPLIER'
,'AUDITOR','LOGISTICS','INVESTOR')
  @Permissions('market.read')
  async getOne(@Actor() actor: any, @Param('id') reservationId:
string) {
    return this.svc.getReservation(actor, reservationId);
  }

  @Get('/:id/status')

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','SUPPLIER'
,'AUDITOR','LOGISTICS','INVESTOR')
  @Permissions('market.read')
  async getStatus(@Actor() actor: any, @Param('id') reservationId:
string) {
    return this.svc.getReservationStatus(actor, reservationId);
  }
}
```

Service

```
// src/market/market-reservations-read.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

@Injectable()
export class MarketReservationsReadService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async getReservation(actor: any, reservationId: string) {
    const ref =
this.db.collection('market_reservations').doc(reservationId);
    const snap = await ref.get();
    if (!snap.exists) throw new Error('RESERVATION_NOT_FOUND');

    const r = snap.data()!;

    // ACL: somente buyerTenant dono (ou admin/manager) pode ver
cobrança
```

```
    const isSameTenant = r.buyerTenantId === actor.tenantId;
    if (!isSameTenant) throw new Error('RESERVATION_FORBIDDEN');

    return {
      reservationId: r.reservationId,
      offerId: r.offerId,
      qty: r.qty,
      status: r.status,
      expiresAt: r.expiresAt,
      createdAt: r.createdAt,
      updatedAt: r.updatedAt,
      escrow: {
        provider: r.escrow?.provider ?? null,
        chargeId: r.escrow?.chargeId ?? null,
        invoiceUrl: r.escrow?.invoiceUrl ?? null,
        pixQrCode: r.escrow?.pixQrCode ?? null,
        status: r.escrow?.status ?? null,
        amount: r.escrow?.amount ?? null,
        paidAt: r.escrow?.paidAt ?? null,
      },
      penaltyPolicy: r.penaltyPolicy ?? null
    };
  }

  async getReservationStatus(actor: any, reservationId: string) {
    const ref =
this.db.collection('market_reservations').doc(reservationId);
    const snap = await ref.get();
    if (!snap.exists) throw new Error('RESERVATION_NOT_FOUND');
    const r = snap.data()!;

    const isSameTenant = r.buyerTenantId === actor.tenantId;
    if (!isSameTenant) throw new Error('RESERVATION_FORBIDDEN');

    // endpoint leve: só status + timers
    return {
      reservationId: r.reservationId,
      status: r.status,
      expiresAt: r.expiresAt,
      updatedAt: r.updatedAt,
      escrowStatus: r.escrow?.status ?? null,
      paidAt: r.escrow?.paidAt ?? null
    };
  }
}
```

---

3) Polling inteligente no app (React Query)

Regras:

- Poll a cada 3s enquanto `ESCROW_PENDING`
- Quando virar `HELD` → para polling e mostra botão “Confirmar”
- Se `EXPIRED` ou `FAILED_OUT_OF_STOCK` → mensagem simples e opção “Tentar outra oferta”

```
// src/hooks/useReservationStatus.ts
import { useQuery } from '@tanstack/react-query';
```

```
import { api } from '../services/api';

export function useReservationStatus(reservationId: string | null) {
  return useQuery({
    queryKey: ['reservationStatus', reservationId],
    enabled: !!reservationId,
    queryFn: async () => (await
api.get(`/api/market/reservations/${reservationId}/status`)).data,
    refetchInterval: (data) => {
      if (!data) return 3000;
      if (data.status === 'ESCROW_PENDING') return 3000;
      return false; // para polling
    },
  });
}
```

---

4) Onde isso encaixa no app (fluxo completo)

1. `POST /reserve` → recebe `reservationId` + `invoiceUrl`
2. Abre a tela "Pagar"
3. Tela "Pagar" usa `useReservationStatus(resId)`
4. Quando status virar `HELD`, habilita botão:
   - `POST /reservations/:id/confirm` → gera `orderId`
5. `POST /contracts/from-order/:orderId` → cria contrato
6. Navega para `ContractDetail`

---

5) O "pulo do gato" (segurança e performance)

- O app **nunca** lê Firestore direto (rules fechadas)
- Tudo via API com idempotência e RBAC
- Polling só em endpoint leve

✅ Isso é o mais estável e seguro para produção.

Se você quiser, eu fecho também o **GET /api/market/orders/:id** (para mostrar pedido antes do contrato) e um endpoint "**start-contract**" que faz (confirm → criar contrato) em 1 chamada só.

Ótimo — vou te entregar agora:

1. `GET /api/market/orders/:id` (detalhe do pedido)
2. `POST /api/market/reservations/:id/confirm-and-contract` (**1 chamada só**: confirma reserva → cria order → cria contrato)
3. Resposta pronta pro app (já retorna `contractId`)

Esse “start-contract” é a melhor forma no mobile porque evita o app ter que encadear 3 chamadas e reduz erro em campo.

---

1) GET /api/market/orders/:id

Controller

```
// src/market/market-orders-read.controller.ts
import { Controller, Get, Param, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { MarketOrdersReadService } from './market-orders-
read.service';

@Controller('/api/market/orders')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class MarketOrdersReadController {
  constructor(private readonly svc: MarketOrdersReadService) {}

  @Get('/:id')

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','SUPPLIER'
,'AUDITOR','LOGISTICS','INVESTOR')
  @Permissions('market.read')
  async getOne(@Actor() actor: any, @Param('id') orderId: string) {
    return this.svc.getOrder(actor, orderId);
  }
}
```

Service

```
// src/market/market-orders-read.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

@Injectable()
export class MarketOrdersReadService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async getOrder(actor: any, orderId: string) {
    const ref = this.db.collection('market_orders').doc(orderId);
    const snap = await ref.get();
    if (!snap.exists) throw new Error('ORDER_NOT_FOUND');

    const o = snap.data()!;

    // ACL: buyerTenant ou supplierTenant (se salvo no pedido)
    const isBuyer = o.buyerTenantId === actor.tenantId;
    const isSupplier = o.supplierTenantId ? (o.supplierTenantId ===
actor.tenantId) : false;

    if (!isBuyer && !isSupplier) throw new Error('ORDER_FORBIDDEN');
```

```
    return {
      orderId: o.orderId ?? orderId,
      buyerTenantId: o.buyerTenantId,
      supplierTenantId: o.supplierTenantId ?? null,
      offerId: o.offerId,
      reservationId: o.reservationId,
      qty: o.qty,
      status: o.status,
      note: o.note ?? null,
      createdAt: o.createdAt,
      updatedAt: o.updatedAt ?? null
    };
  }
}
```

---

2) POST /api/market/reservations/:id/confirm-and-contract

O que faz

- verifica reservation `HELD` (escrow pago e estoque reservado)
- cria `market_order`
- cria `contract` (estado `CREATED`)
- registra auditoria
- devolve `{ reservationId, orderId, contractId }`

DTO

```
// src/market/dto/confirm-and-contract.dto.ts
import { IsNotEmpty, IsOptional, IsString } from 'class-validator';

export class ConfirmAndContractDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsOptional() @IsString()
  technicianTenantId?: string;

  @IsOptional() @IsString()
  logisticsTenantId?: string;

  @IsOptional() @IsString()
  note?: string;
}
```

Controller

```
// src/market/market-start-contract.controller.ts
import { Body, Controller, Param, Post, UseGuards } from
'@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
```

```
import { MarketStartContractService } from './market-start-
contract.service';
import { ConfirmAndContractDto } from './dto/confirm-and-
contract.dto';

@Controller('/api/market/reservations')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class MarketStartContractController {
  constructor(private readonly svc: MarketStartContractService) {}

  @Post('/:id/confirm-and-contract')
  @Roles('ADMIN','MANAGER','OPERATOR','PRODUCER')
  @Permissions('market.reserve')
  async confirmAndContract(
    @Actor() actor: any,
    @Param('id') reservationId: string,
    @Body() dto: ConfirmAndContractDto
  ) {
    return this.svc.confirmAndCreateContract(actor, reservationId,
dto);
  }
}
```

Service (transação Firestore “tudo ou nada”)

```
// src/market/market-start-contract.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';
import { idemKey } from '../common/idempotency/idempotency.util';

@Injectable()
export class MarketStartContractService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async confirmAndCreateContract(actor: any, reservationId: string,
dto: any) {
    const now = Date.now();
    const key = idemKey(actor.tenantId, actor.uid,
dto.clientRequestId, 'market.confirmAndContract');
    const idemRef = this.db.collection('idempotency').doc(key);

    const resRef =
this.db.collection('market_reservations').doc(reservationId);

    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const rSnap = await tx.get(resRef);
      if (!rSnap.exists) throw new Error('RESERVATION_NOT_FOUND');
      const r = rSnap.data()!;

      // ACL buyer
      if (r.buyerTenantId !== actor.tenantId) throw new
Error('RESERVATION_FORBIDDEN');

      // precisa estar HELD (pagou + reservou estoque)
      if (r.status !== 'HELD' || r.escrow?.status !== 'PAID') {
        throw new Error('RESERVATION_NOT_HELD');
```

```
      }

      // offer
      const offerRef =
this.db.collection('market_offers').doc(r.offerId);
      const offerSnap = await tx.get(offerRef);
      if (!offerSnap.exists) throw new Error('OFFER_NOT_FOUND');
      const offer = offerSnap.data()!;

      // cria order
      const orderId = makeId('ord');
      const orderRef =
this.db.collection('market_orders').doc(orderId);

      tx.create(orderRef, {
        orderId,
        buyerTenantId: r.buyerTenantId,
        buyerUid: r.buyerUid,
        supplierTenantId: offer.supplierTenantId,
        offerId: r.offerId,
        reservationId: r.reservationId,
        qty: r.qty,
        status: 'CONFIRMED',
        note: dto.note ?? null,
        createdAt: now,
        updatedAt: now
      });

      // cria contrato
      const contractId = makeId('ctr');
      const contractRef =
this.db.collection('contracts').doc(contractId);

      const total = Number(r.escrow?.amount || (Number(offer.price ||
0) * Number(r.qty || 0)));

      tx.create(contractRef, {
        contractId,
        tenantId: r.buyerTenantId,
        buyerTenantId: r.buyerTenantId,
        supplierTenantId: offer.supplierTenantId,
        orderId,
        reservationId: r.reservationId,
        type: 'SUPPLY_B2C',
        subject: {
          offerId: offerRef.id,
          catalogItemId: offer.catalogItemId ?? null,
          title: offer.title,
          uom: offer.uom,
          qty: r.qty,
          unitPrice: offer.price,
          total
        },
        parties: {
          technicianTenantId: dto.technicianTenantId ?? null,
          logisticsTenantId: dto.logisticsTenantId ?? null
        },
        escrow: {
          provider: 'ASAAS',
          chargeId: r.escrow?.chargeId ?? null,
          amount: total,
```

```
          currency: 'BRL',
          status: 'PAID',
          heldAmount: total,
          releasedAmount: 0
        },
        settlementPlan: {
          type: 'TRANCHES',
          tranches: [
            { milestone: 'DISPATCH_CONFIRMED', percent: 30 },
            { milestone: 'DELIVERY_CONFIRMED', percent: 60 },
            { milestone: 'FISCAL_DOC_VERIFIED', percent: 10 }
          ],
          split: [
            { target: 'SUPPLIER', percent: 95 },
            { target: 'TECHNICIAN', percent: 3 },
            { target: 'LOGISTICS', percent: 2 }
          ]
        },
        state: 'CREATED',
        acceptance: { accepted: false, acceptanceHash: null,
acceptedAt: null },
        createdAt: now,
        updatedAt: now
      });

      // marca reservation CONFIRMED + aponta order/contract
      tx.set(resRef, { status: 'CONFIRMED', orderId, contractId,
updatedAt: now }, { merge: true });

      // auditoria append-only
      const evtId = makeId('evt');
      tx.create(this.db.collection('contract_events').doc(evtId), {
        eventId: evtId,
        tenantId: r.buyerTenantId,
        contractId,
        type: 'CONTRACT_CREATED_FROM_RESERVATION',
        actor: { uid: actor.uid, role: actor.role },
        meta: { reservationId, orderId },
        ts: now
      });

      const out = { reservationId, orderId, contractId };
      tx.create(idemRef, {
        tenantId: actor.tenantId,
        uid: actor.uid,
        scope: 'market.confirmAndContract',
        clientRequestId: dto.clientRequestId,
        createdAt: now,
        result: out
      });

      return out;
    });
  }
}
```

---

3) App: usar 1 chamada só

Depois que `reservation.status` virar `HELD`, o app chama:

```
await api.post(`/api/market/reservations/${reservationId}/confirm-and-contract`, {
  clientRequestId: uuid(),
  note: 'Compra via app',
});
```

Resposta:

```
{ "reservationId":"res_...", "orderId":"ord_...", "contractId":"ctr_..." }
```

E o app navega direto:

- `ContractDetail(contractId)`

---

4) Registro no módulo

No `MarketModule` inclua:

- `MarketOrdersReadController`, `MarketOrdersReadService`
- `MarketStartContractController`, `MarketStartContractService`
- (e os anteriores) Offers/Reservations read

---

5) Observação importante (estoque + fraude)

Esse fluxo é seguro porque:

- o estoque já está travado em `HELD` (pago)
- a criação do contrato é transacional
- idempotência garante "sem duplicar" se o operador clicar 2 vezes

---

Se quiser, eu também fecho o **"start-contract offline"**: o app enfileira o `confirm-and-contract` na fila e quando voltar internet ele cria o contrato automaticamente (mantendo idempotência).

Fechado — aqui vai o **start-contract offline** (a melhor forma no campo), com:

- fila local (AsyncStorage)
- idempotência (clientRequestId fixo)
- UI simples "Pendente/Enviado"
- retry automático quando internet voltar
- evita duplicar contrato mesmo se reenviar 10x

---

1) Padrão ouro: `clientRequestId` fixo por operação

Quando o usuário clicar “Finalizar e criar contrato”, você gera **uma vez** e guarda junto do `reservationId` no dispositivo.

✅ Mesmo que:

- o app feche
- a internet caia
- ele clique de novo

…o backend vai retornar o mesmo resultado por idempotência.

---

2) Funções offline específicas do start-contract

2.1 Modelo do item da fila

```
// src/offline/types.ts
export type OfflineOp =
  | {
      kind: 'CONFIRM_AND_CONTRACT';
      reservationId: string;
      clientRequestId: string;
      payload: {
        technicianTenantId?: string;
        logisticsTenantId?: string;
        note?: string;
      };
      createdAt: number;
      status: 'PENDING' | 'SENT' | 'FAILED';
      lastError?: string | null;
      result?: { orderId: string; contractId: string } | null;
    };
```

2.2 Storage helper (lista de operações)

```
// src/offline/store.ts
import AsyncStorage from '@react-native-async-storage/async-storage';
import { OfflineOp } from './types';

const KEY = 'offline_ops_v1';

export async function loadOps(): Promise<OfflineOp[]> {
  const raw = await AsyncStorage.getItem(KEY);
  return raw ? JSON.parse(raw) : [];
}

export async function saveOps(ops: OfflineOp[]) {
  await AsyncStorage.setItem(KEY, JSON.stringify(ops));
}

export async function upsertOp(op: OfflineOp) {
  const ops = await loadOps();
  const idx = ops.findIndex(
    (x) => x.kind === op.kind && x.reservationId === op.reservationId
  );
  if (idx >= 0) ops[idx] = op;
```

```
  else ops.push(op);
  await saveOps(ops);
  return op;
}

export async function markOp(
  reservationId: string,
  patch: Partial<OfflineOp>
) {
  const ops = await loadOps();
  const idx = ops.findIndex(
    (x) => x.kind === 'CONFIRM_AND_CONTRACT' && x.reservationId ===
reservationId
  );
  if (idx < 0) return null;
  ops[idx] = { ...ops[idx], ...patch } as any;
  await saveOps(ops);
  return ops[idx];
}
```

---

3) Sender: envia quando houver internet

```
// src/offline/senders/confirmAndContract.sender.ts
import NetInfo from '@react-native-community/netinfo';
import { api } from '../../services/api';
import { loadOps, markOp } from '../store';
import { OfflineOp } from '../types';

export async function flushConfirmAndContractOps() {
  const state = await NetInfo.fetch();
  if (!state.isConnected) return { ok: false, reason: 'OFFLINE' };

  const ops = await loadOps();
  const pending = ops.filter(
    (o) => o.kind === 'CONFIRM_AND_CONTRACT' && (o.status ===
'PENDING' || o.status === 'FAILED')
  ) as OfflineOp[];

  let sent = 0;

  for (const op of pending) {
    try {
      const res = await api.post(
        `/api/market/reservations/${op.reservationId}/confirm-and-
contract`,
        { clientRequestId: op.clientRequestId, ...op.payload }
      );

      await markOp(op.reservationId, {
        status: 'SENT',
        lastError: null,
        result: { orderId: res.data.orderId, contractId:
res.data.contractId },
      } as any);

      sent += 1;
    } catch (e: any) {
      await markOp(op.reservationId, {
        status: 'FAILED',
        lastError: e?.message ?? 'ERROR',
```

```
      } as any);
    }
  }

  return { ok: true, sent, total: pending.length };
}
```

---

4) Hook para disparar flush automático

```
// src/offline/useOfflineFlush.ts
import { useEffect } from 'react';
import NetInfo from '@react-native-community/netinfo';
import { flushConfirmAndContractOps } from
'./senders/confirmAndContract.sender';

export function useOfflineFlush() {
  useEffect(() => {
    // roda ao iniciar
    flushConfirmAndContractOps();

    const unsub = NetInfo.addEventListener((s) => {
      if (s.isConnected) flushConfirmAndContractOps();
    });

    return () => unsub();
  }, []);
}
```

No `App.tsx`:

```
import React from 'react';
import { useOfflineFlush } from './src/offline/useOfflineFlush';

export default function App() {
  useOfflineFlush();
  return null; // seu NavigationContainer aqui
}
```

---

5) UI: Tela “Pagar” com estado Offline/Online

Quando status virar `HELD`, mostrar:

- ✅ “Pago — pronto para finalizar”
- Botão grande “Finalizar e criar contrato”
  - se offline: “Salvo para enviar assim que voltar internet”

Implementação do botão (send-or-queue)

```
// src/offline/actions/confirmAndContract.action.ts
import { v4 as uuid } from 'uuid';
import NetInfo from '@react-native-community/netinfo';
import { api } from '../../services/api';
import { upsertOp } from '../store';
import { OfflineOp } from '../types';
```

```
export async function confirmAndContract(reservationId: string,
payload: any) {
  // clientRequestId fixo por reserva
  const clientRequestId = uuid();

  const op: OfflineOp = {
    kind: 'CONFIRM_AND_CONTRACT',
    reservationId,
    clientRequestId,
    payload,
    createdAt: Date.now(),
    status: 'PENDING',
    lastError: null,
    result: null,
  };

  await upsertOp(op);

  const state = await NetInfo.fetch();
  if (!state.isConnected) {
    return { queued: true, reservationId };
  }

  // tenta enviar agora
  const res = await
api.post(`/api/market/reservations/${reservationId}/confirm-and-
contract`, {
    clientRequestId,
    ...payload,
  });

  // marca como enviado
  await upsertOp({
    ...op,
    status: 'SENT',
    result: { orderId: res.data.orderId, contractId:
res.data.contractId },
  });

  return { queued: false, ...res.data };
}
```

⚠️ Ajuste importante (para “clientRequestId fixo”)

O melhor é **reusar** o mesmo `clientRequestId` se o usuário clicar de novo.

Então, em vez de gerar sempre novo, faça:

- buscar se já existe op para reservationId
- se existir, reutiliza o `clientRequestId`

Aqui está a versão melhor:

```
// src/offline/actions/confirmAndContract.action.ts
import { v4 as uuid } from 'uuid';
import NetInfo from '@react-native-community/netinfo';
import { api } from '../../services/api';
import { loadOps, upsertOp } from '../store';
```

```
import { OfflineOp } from '../types';

export async function confirmAndContract(reservationId: string,
payload: any) {
  const ops = await loadOps();
  const existing = ops.find(
    (o) => o.kind === 'CONFIRM_AND_CONTRACT' && o.reservationId ===
reservationId
  ) as OfflineOp | undefined;

  const clientRequestId = existing?.clientRequestId ?? uuid();

  const op: OfflineOp = {
    kind: 'CONFIRM_AND_CONTRACT',
    reservationId,
    clientRequestId,
    payload,
    createdAt: existing?.createdAt ?? Date.now(),
    status: existing?.status === 'SENT' ? 'SENT' : 'PENDING',
    lastError: null,
    result: existing?.result ?? null,
  };

  await upsertOp(op);

  if (op.status === 'SENT' && op.result?.contractId) {
    return { queued: false, reservationId, ...op.result };
  }

  const state = await NetInfo.fetch();
  if (!state.isConnected) return { queued: true, reservationId };

  const res = await
api.post(`/api/market/reservations/${reservationId}/confirm-and-
contract`, {
    clientRequestId,
    ...payload,
  });

  await upsertOp({
    ...op,
    status: 'SENT',
    result: { orderId: res.data.orderId, contractId:
res.data.contractId },
  });

  return { queued: false, ...res.data };
}
```

---

6) Tela exemplo “Pagamento e Finalização”

```
// src/screens/ReservationPayScreen.tsx
import React from 'react';
import { Text, View } from 'react-native';
import { BigButton } from '../ui/BigButton';
import { useReservationStatus } from '../hooks/useReservationStatus';
import { confirmAndContract } from
'../offline/actions/confirmAndContract.action';

export function ReservationPayScreen({ route, navigation }: any) {
```

```
  const { reservationId, invoiceUrl } = route.params;
  const q = useReservationStatus(reservationId);

  const status = q.data?.status;

  async function finish() {
    const out = await confirmAndContract(reservationId, { note:
'Compra mobile' });

    if (out.queued) {
      navigation.navigate('PendingOps');
      return;
    }

    navigation.replace('ContractDetail', { id: out.contractId });
  }

  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 24, fontWeight: '800'
}}>Pagamento</Text>
      <Text style={{ fontSize: 16, opacity: 0.8, marginTop: 8 }}>
        Status: {status ?? '...'}
      </Text>

      <View style={{ marginTop: 16 }}>
        <BigButton
          icon="🔗"
          title="Abrir link de pagamento"
          subtitle="PIX / Boleto"
          onPress={() => navigation.navigate('WebPay', { url:
invoiceUrl })}
        />

        <BigButton
          icon="✅"
          title="Finalizar e criar contrato"
          subtitle="Se estiver offline, será enviado depois"
          onPress={finish}
          disabled={status !== 'HELD'}
        />
      </View>
    </View>
  );
}
```

---

7) Tela “Pendências” (para o operador entender)

```
// src/screens/PendingOpsScreen.tsx
import React, { useEffect, useState } from 'react';
import { FlatList, Text, View } from 'react-native';
import { BigButton } from '../ui/BigButton';
import { loadOps } from '../offline/store';
import { flushConfirmAndContractOps } from
'../offline/senders/confirmAndContract.sender';

export function PendingOpsScreen() {
  const [ops, setOps] = useState<any[]>([]);

  async function refresh() {
```

```
    setOps(await loadOps());
  }

  useEffect(() => { refresh(); }, []);

  async function syncNow() {
    await flushConfirmAndContractOps();
    await refresh();
  }

  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 24, fontWeight: '800', marginBottom: 12
}}>
        Pendências
      </Text>

      <BigButton icon="🔄" title="Sincronizar agora" subtitle="Tenta
enviar pendências" onPress={syncNow} />

      <FlatList
        data={ops}
        keyExtractor={(it) => `${it.kind}_${it.reservationId}`}
        renderItem={({ item }) => (
          <BigButton
            icon={item.status === 'SENT' ? '✅' : item.status ===
'FAILED' ? '❌' : '⏳'}
            title={`${item.kind} (${item.status})`}
            subtitle={item.lastError ? `Erro: ${item.lastError}` :
`Reserva: ${item.reservationId}`}
            onPress={() => {}}
          />
        )}
      />
    </View>
  );
}
```

---

8) Resultado final (na prática)

✅ Operador paga → sem internet → clica “Finalizar” → fica “pendente”
✅ Quando voltar rede → app envia sozinho → contrato criado
✅ Se ele clicar várias vezes → **idempotência** impede duplicar
✅ Tudo fica auditado no backend

Se você quiser, eu fecho também o **offline para evidências** (foto/áudio/documento fiscal) com upload assíncrono (bem importante no campo).

Perfeito — vamos fechar o **offline para evidências** (foto/áudio/documento fiscal) do jeito mais seguro e “à prova de campo”:

✅ captura mesmo sem internet
✅ guarda arquivo local + metadados
✅ quando voltar rede:

- pede **URL assinada**
- faz **upload**
- faz **commit** no backend (cria `contract_evidence` / `fiscal_docs`)
  ✅ idempotência evita duplicar
  ✅ mostra "pendente/enviado/erro" para o operador

Vou te entregar em **blocos prontos**.

---

1) Tipos de evidência offline

1.1 Tipos suportados

- **PHOTO** (imagem)
- **AUDIO** (voz)
- **DOC** (NF-e XML/PDF)

1.2 Item de fila (offline)

```
// src/offline/evidenceTypes.ts
export type EvidenceKind = 'PHOTO' | 'AUDIO' | 'DOC';

export type OfflineEvidenceOp = {
  kind: 'EVIDENCE_UPLOAD';
  evidenceKind: EvidenceKind;

  // pode ser evidência geral ou vinculada a contrato
  contractId?: string | null;

  // NF-e: exige fiscalKey e tipo de arquivo (XML/PDF)
  fiscal?: {
    fiscalKey: string;   // 44 dígitos
    fileType: 'XML' | 'PDF';
  } | null;

  // metadados da evidência
  riskType: 'A' | 'B' | 'DOC';
  note?: string | null;
  geo?: { lat: number; lng: number } | null;

  // arquivo local no celular
  localUri: string;        // ex: file://...
  contentType: string;     // image/jpeg, application/xml,
application/pdf...
  fileName: string;        // nfe.xml, danfe.pdf, foto.jpg, voz.m4a
etc.

  // idempotência
  clientRequestId: string;

  // estado da fila
```

```
  status: 'PENDING' | 'UPLOADING' | 'COMMITTED' | 'FAILED';
  lastError?: string | null;

  // retorno do backend
  remote?: {
    uploadPath?: string;   // gs:// path
    evidenceId?: string;
    fiscalDocId?: string;
  } | null;

  createdAt: number;
  updatedAt: number;
};
```

---

2) Store (AsyncStorage) para evidências

```
// src/offline/evidenceStore.ts
import AsyncStorage from '@react-native-async-storage/async-storage';
import { OfflineEvidenceOp } from './evidenceTypes';

const KEY = 'offline_evidence_ops_v1';

export async function loadEvidenceOps(): Promise<OfflineEvidenceOp[]>
{
  const raw = await AsyncStorage.getItem(KEY);
  return raw ? JSON.parse(raw) : [];
}

export async function saveEvidenceOps(ops: OfflineEvidenceOp[]) {
  await AsyncStorage.setItem(KEY, JSON.stringify(ops));
}

export async function upsertEvidenceOp(op: OfflineEvidenceOp) {
  const ops = await loadEvidenceOps();
  const idx = ops.findIndex((x) => x.clientRequestId ===
op.clientRequestId);
  if (idx >= 0) ops[idx] = op;
  else ops.push(op);
  await saveEvidenceOps(ops);
  return op;
}

export async function patchEvidenceOp(clientRequestId: string, patch:
Partial<OfflineEvidenceOp>) {
  const ops = await loadEvidenceOps();
  const idx = ops.findIndex((x) => x.clientRequestId ===
clientRequestId);
  if (idx < 0) return null;
  ops[idx] = { ...ops[idx], ...patch, updatedAt: Date.now() } as any;
  await saveEvidenceOps(ops);
  return ops[idx];
}
```

---

3) Endpoints necessários no backend (mínimo)

3.1 Evidência geral / contrato

- `POST /api/evidence/upload-url` → URL assinada
- `POST /api/contracts/:id/evidence` → commit (já existia)

3.2 Fiscal (NF-e)

- `POST /api/contracts/:id/fiscal/upload-url?fiscalKey=...` → URL assinada
- `POST /api/contracts/:id/fiscal/commit` → cria `fiscal_docs` + evidência `DOC`

Você já tem os fiscais; só precisamos do **upload-url de evidência geral** (se quiser centralizar):

Exemplo simples:

- `POST /api/evidence/upload-url` body `{ fileName, contentType }` retorna `{ url, path }`

---

4) Criar a evidência offline no app (captura)

4.1 Criar op de foto/voz vinculada a contrato

```
// src/offline/actions/enqueueEvidence.ts
import { v4 as uuid } from 'uuid';
import { upsertEvidenceOp } from '../evidenceStore';
import { OfflineEvidenceOp } from '../evidenceTypes';

export async function enqueueContractEvidence(input: {
  contractId: string;
  evidenceKind: 'PHOTO'|'AUDIO';
  localUri: string;
  contentType: string;
  fileName: string;
  riskType: 'A'|'B';
  geo?: {lat:number; lng:number} | null;
  note?: string | null;
}) {
  const op: OfflineEvidenceOp = {
    kind: 'EVIDENCE_UPLOAD',
    evidenceKind: input.evidenceKind,
    contractId: input.contractId,
    fiscal: null,
    riskType: input.riskType,
    note: input.note ?? null,
    geo: input.geo ?? null,
    localUri: input.localUri,
    contentType: input.contentType,
    fileName: input.fileName,
    clientRequestId: uuid(),
    status: 'PENDING',
    lastError: null,
    remote: null,
    createdAt: Date.now(),
    updatedAt: Date.now(),
  };
```

```
  await upsertEvidenceOp(op);
  return op;
}
```

4.2 NF-e (XML/PDF) offline

```
// src/offline/actions/enqueueFiscalFile.ts
import { v4 as uuid } from 'uuid';
import { upsertEvidenceOp } from '../evidenceStore';
import { OfflineEvidenceOp } from '../evidenceTypes';

export async function enqueueFiscalFile(input: {
  contractId: string;
  fiscalKey: string;             // 44 dígitos
  fileType: 'XML'|'PDF';
  localUri: string;
  contentType: string;           // application/xml | application/pdf
  fileName: string;              // nfe.xml | danfe.pdf
}) {
  const op: OfflineEvidenceOp = {
    kind: 'EVIDENCE_UPLOAD',
    evidenceKind: 'DOC',
    contractId: input.contractId,
    fiscal: { fiscalKey: input.fiscalKey, fileType: input.fileType },
    riskType: 'DOC',
    note: `NF-e ${input.fileType}`,
    geo: null,
    localUri: input.localUri,
    contentType: input.contentType,
    fileName: input.fileName,
    clientRequestId: uuid(),
    status: 'PENDING',
    lastError: null,
    remote: null,
    createdAt: Date.now(),
    updatedAt: Date.now(),
  };

  await upsertEvidenceOp(op);
  return op;
}
```

---

5) Uploader: enviar evidências quando voltar rede

5.1 Funções utilitárias de upload (PUT na URL assinada)

```
// src/offline/uploadHelpers.ts
import * as FileSystem from 'expo-file-system';

export async function putFileToSignedUrl(url: string, localUri:
string, contentType: string) {
  const res = await FileSystem.uploadAsync(url, localUri, {
    httpMethod: 'PUT',
    uploadType: FileSystem.FileSystemUploadType.BINARY_CONTENT,
    headers: { 'Content-Type': contentType },
  });
  if (res.status < 200 || res.status >= 300) {
    throw new Error(`UPLOAD_FAILED_${res.status}`);
  }
  return true;
}
```

5.2 Sender principal (flush)

```
// src/offline/senders/evidence.sender.ts
import NetInfo from '@react-native-community/netinfo';
import { api } from '../../services/api';
import { loadEvidenceOps, patchEvidenceOp } from '../evidenceStore';
import { putFileToSignedUrl } from '../uploadHelpers';

export async function flushEvidenceOps() {
  const net = await NetInfo.fetch();
  if (!net.isConnected) return { ok: false, reason: 'OFFLINE' };

  const ops = await loadEvidenceOps();
  const pending = ops.filter((o) => o.kind === 'EVIDENCE_UPLOAD' &&
(o.status === 'PENDING' || o.status === 'FAILED'));

  let sent = 0;

  for (const op of pending) {
    try {
      await patchEvidenceOp(op.clientRequestId, { status: 'UPLOADING',
lastError: null });

      // 1) pedir URL assinada
      let uploadUrlRes;

      if (op.evidenceKind === 'DOC' && op.fiscal?.fiscalKey &&
op.contractId) {
        // NF-e: usa endpoint fiscal (melhor)
        uploadUrlRes = await api.post(
          `/api/contracts/${op.contractId}/fiscal/upload-
url?fiscalKey=${op.fiscal.fiscalKey}`,
          { type: op.fiscal.fileType }
        );
      } else {
        // evidência comum (foto/áudio)
        uploadUrlRes = await api.post(`/api/evidence/upload-url`, {
          fileName: op.fileName,
          contentType: op.contentType,
        });
      }

      const { url, path } = uploadUrlRes.data;

      // 2) fazer PUT do arquivo direto no Storage
      await putFileToSignedUrl(url, op.localUri, op.contentType);

      // 3) commit no backend
      if (op.evidenceKind === 'DOC' && op.contractId &&
op.fiscal?.fiscalKey) {
        // commit fiscal: marca que XML/PDF foi enviado
        // (Você pode também juntar XML+PDF e commitar só uma vez;
aqui é por arquivo)
        const xmlUploaded = op.fiscal.fileType === 'XML';
        const pdfUploaded = op.fiscal.fileType === 'PDF';

        const commit = await
api.post(`/api/contracts/${op.contractId}/fiscal/commit`, {
          clientRequestId: op.clientRequestId,
          fiscalKey: op.fiscal.fiscalKey,
          xmlUploaded,
          pdfUploaded,
```

```
        });

        await patchEvidenceOp(op.clientRequestId, {
          status: 'COMMITTED',
          remote: { uploadPath: path, fiscalDocId: commit.data.docId
},
        });
      } else if (op.contractId) {
        // evidência vinculada a contrato
        const commit = await
api.post(`/api/contracts/${op.contractId}/evidence`, {
          clientRequestId: op.clientRequestId,
          type: op.evidenceKind === 'PHOTO' ? 'PHOTO' : 'AUDIO',
          riskType: op.riskType,
          geo: op.geo ?? null,
          note: op.note ?? null,
          // opcional: incluir referência do uploadPath para ligar
storage->evidence
          storagePath: path,
        });

        await patchEvidenceOp(op.clientRequestId, {
          status: 'COMMITTED',
          remote: { uploadPath: path, evidenceId:
commit.data?.evidenceId ?? null },
        });
      } else {
        // evidência global (sem contrato)
        const commit = await api.post(`/api/evidence/commit`, {
          clientRequestId: op.clientRequestId,
          type: op.evidenceKind,
          riskType: op.riskType,
          note: op.note ?? null,
          storagePath: path,
        });

        await patchEvidenceOp(op.clientRequestId, {
          status: 'COMMITTED',
          remote: { uploadPath: path, evidenceId:
commit.data?.evidenceId ?? null },
        });
      }

      sent += 1;
    } catch (e: any) {
      await patchEvidenceOp(op.clientRequestId, {
        status: 'FAILED',
        lastError: e?.message ?? 'ERROR',
      });
    }
  }

  return { ok: true, sent, total: pending.length };
}
```

---

6) Disparar flush automático junto com o “start-contract offline”

No mesmo `useOfflineFlush`, chame também o flush de evidências:

```
// src/offline/useOfflineFlush.ts
import { useEffect } from 'react';
import NetInfo from '@react-native-community/netinfo';
import { flushConfirmAndContractOps } from
'./senders/confirmAndContract.sender';
import { flushEvidenceOps } from './senders/evidence.sender';

export function useOfflineFlush() {
  useEffect(() => {
    flushConfirmAndContractOps();
    flushEvidenceOps();

    const unsub = NetInfo.addEventListener((s) => {
      if (s.isConnected) {
        flushConfirmAndContractOps();
        flushEvidenceOps();
      }
    });

    return () => unsub();
  }, []);
}
```

---

7) Tela “Pendências” mostrando evidências também

Você pode reaproveitar a tela “Pendências” e listar `offline_evidence_ops_v1` junto.

O operador vai entender:

- ⏳ “Foto pendente”
- ✅ “Enviado”
- ❌ “Falhou — tocar para tentar novamente”

---

8) Detalhe MUITO importante para NF-e (XML + PDF)

Para ficar perfeito:

✅ **Salvar XML e PDF na fila** separadamente
✅ Depois, quando ambos estiverem `COMMITTED`, o app libera:

- botão “Verificar NF-e” (`POST /fiscal/verify`)
- e só depois transiciona `FISCAL_DOC_VERIFIED`

Se estiver offline:

- o app permite enfileirar XML/PDF
- quando voltar, envia
- e só então libera a verificação

---

9) Melhorias finais (recomendadas)

1. **Limpeza automática**: apagar do AsyncStorage ops `COMMITTED` com mais de 7 dias
2. **Retry exponencial**: se falhar 3x, aguardar mais tempo
3. **Tamanho de arquivo**: compressão de imagem no app (para 4G fraco)
4. **Geofence**: anexar geo no momento da captura (mesmo offline), não no envio

---

Se você quiser, eu te passo também o **backend mínimo** do endpoint `POST /api/evidence/upload-url` e `POST /api/evidence/commit` (para evidências que não são de contrato), já com `tenantId`, auditoria e hash.

Beleza — abaixo está o **backend mínimo completo** para:

1. `POST /api/evidence/upload-url` (URL assinada no Firebase Storage)
2. `POST /api/evidence/commit` (cria evidência global + auditoria append-only)
3. (opcional) `GET /api/evidence/:id` (para auditoria/consulta)

Tudo com:

- `tenantId` (multi-tenant)
- RBAC/permissões
- idempotência (`clientRequestId`)
- registro em auditoria (`audit_events`)

Importante: evidência vinculada a contrato você já faz em `POST /api/contracts/:id/evidence`. Esse módulo é para evidência "geral" (ex.: foto de planilha manual que depois vira estoque/relatório).

---

1) DTOs

```
// src/evidence/dto/evidence.dto.ts
import { IsNotEmpty, IsOptional, IsString } from 'class-validator';

export class EvidenceUploadUrlDto {
  @IsString() @IsNotEmpty()
  fileName: string;

  @IsString() @IsNotEmpty()
  contentType: string; // image/jpeg, application/pdf, audio/m4a...
}

export class EvidenceCommitDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsString() @IsNotEmpty()
```

```
  type: 'PHOTO'|'AUDIO'|'DOC';

  @IsString() @IsNotEmpty()
  riskType: 'A'|'B'|'DOC';

  @IsString() @IsNotEmpty()
  storagePath: string; // caminho no bucket (retornado pelo upload-
url)

  @IsOptional()
  geo?: { lat: number; lng: number };

  @IsOptional() @IsString()
  note?: string;
}
```

---

2) EvidenceService

```
// src/evidence/evidence.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { StorageService } from
'../integrations/storage/storage.service';
import { makeId } from '../common/utils/id.util';
import { idemKey } from '../common/idempotency/idempotency.util';

function sanitizeName(name: string) {
  return name.replace(/[^a-zA-Z0-9._-]/g, '_').slice(0, 80);
}

@Injectable()
export class EvidenceService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly storage: StorageService
  ) {}

  async getUploadUrl(actor: any, dto: { fileName: string; contentType:
string }) {
    const now = Date.now();
    const safe = sanitizeName(dto.fileName);
    const path = `evidence/${actor.tenantId}/${now}_${safe}`;
    return this.storage.getSignedUploadUrl(path, dto.contentType, 20);
  }

  async commit(actor: any, dto: any) {
    const now = Date.now();
    const key = idemKey(actor.tenantId, actor.uid,
dto.clientRequestId, 'evidence.commit');
    const idemRef = this.db.collection('idempotency').doc(key);

    // valida se o arquivo existe no bucket antes de commitar (best
effort)
    const exists = await this.storage.exists(dto.storagePath);
    if (!exists) throw new Error('STORAGE_FILE_NOT_FOUND');

    // calcula hash para lastro (prova)
    const sha = await this.storage.sha256FromGcs(dto.storagePath);
```

```
    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const evidenceId = makeId('evg');

      tx.create(this.db.collection('evidence').doc(evidenceId), {
        evidenceId,
        tenantId: actor.tenantId,
        type: dto.type,
        riskType: dto.riskType,
        storagePath: dto.storagePath,
        sha256: sha,
        geo: dto.geo ?? null,
        note: dto.note ?? null,
        actor: { uid: actor.uid, role: actor.role },
        createdAt: now,
      });

      // auditoria append-only
      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId: actor.tenantId,
        type: 'EVIDENCE_COMMITTED',
        actor: { uid: actor.uid, role: actor.role },
        ref: { evidenceId, storagePath: dto.storagePath },
        meta: { evidenceType: dto.type, riskType: dto.riskType,
sha256: sha },
        ts: now
      });

      const out = { evidenceId, sha256: sha };
      tx.create(idemRef, {
        tenantId: actor.tenantId,
        uid: actor.uid,
        scope: 'evidence.commit',
        clientRequestId: dto.clientRequestId,
        createdAt: now,
        result: out
      });

      return out;
    });
  }

  async getOne(actor: any, evidenceId: string) {
    const ref = this.db.collection('evidence').doc(evidenceId);
    const snap = await ref.get();
    if (!snap.exists) throw new Error('EVIDENCE_NOT_FOUND');

    const ev = snap.data()!;
    if (ev.tenantId !== actor.tenantId) throw new
Error('EVIDENCE_FORBIDDEN');

    return ev;
  }
}
```

---

3) EvidenceController

```
// src/evidence/evidence.controller.ts
import { Body, Controller, Get, Param, Post, UseGuards } from
'@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { EvidenceService } from './evidence.service';
import { EvidenceCommitDto, EvidenceUploadUrlDto } from
'./dto/evidence.dto';

@Controller('/api/evidence')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class EvidenceController {
  constructor(private readonly svc: EvidenceService) {}

  @Post('/upload-url')

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','SUPPLIER'
,'AUDITOR','LOGISTICS','INVESTOR')
  @Permissions('evidence.write')
  async uploadUrl(@Actor() actor: any, @Body() dto:
EvidenceUploadUrlDto) {
    return this.svc.getUploadUrl(actor, dto);
  }

  @Post('/commit')

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','SUPPLIER'
,'AUDITOR','LOGISTICS','INVESTOR')
  @Permissions('evidence.write')
  async commit(@Actor() actor: any, @Body() dto: EvidenceCommitDto) {
    return this.svc.commit(actor, dto);
  }

  @Get('/:id')

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','SUPPLIER'
,'AUDITOR','LOGISTICS','INVESTOR')
  @Permissions('evidence.read')
  async getOne(@Actor() actor: any, @Param('id') id: string) {
    return this.svc.getOne(actor, id);
  }
}
```

---

4) EvidenceModule

```
// src/evidence/evidence.module.ts
import { Module } from '@nestjs/common';
import { firestoreProvider } from '../common/firestore.provider';
import { StorageService } from
'../integrations/storage/storage.service';
import { EvidenceController } from './evidence.controller';
import { EvidenceService } from './evidence.service';
```

```
@Module({
  providers: [firestoreProvider, StorageService, EvidenceService],
  controllers: [EvidenceController],
})
export class EvidenceModule {}
```

Adicione no `AppModule`.

---

5) Permissões (mínimo necessário)

Você pediu JSON de permissions/seeds antes — então aqui só reforço o mínimo para esse bloco:

- `evidence.read`
- `evidence.write`
- `market.read`
- `market.reserve`
- `contracts.read`
- `contracts.write`

---

6) Integração com o offline sender (perfeito encaixe)

O sender offline já faz:

1. `POST /api/evidence/upload-url`
2. `PUT` no Storage (URL assinada)
3. `POST /api/evidence/commit`

✅ Resultado: evidência fica registrada com `sha256` + `audit_events`.

Se você quiser, eu fecho agora também:

- **limpeza automática** da fila (apagar COMMITTED)
- endpoint `GET /api/audit/events` (interface read-only pro auditor)

Perfeito — abaixo vai o pacote final com:

1. **Limpeza automática da fila offline** (apaga COMMITTED antigos + limita tamanho)
2. **Endpoint read-only de auditoria** `GET /api/audit/events` (com filtros, paginação e ACL)
3. **Endpoint read-only por contrato** `GET /api/contracts/:id/audit` (mais útil no dia a dia)

Tudo pronto para copiar/colar.

---

1) Limpeza automática da fila offline

1.1 Função de cleanup (evidências + start-contract)

Você mantém:

- itens `PENDING/FAILED` (sempre)
- itens `COMMITTED/SENT` só por X dias (ex.: 7)
- limita a fila a um máximo (ex.: 300 itens) para não crescer infinito

```
// src/offline/cleanup.ts
import { loadEvidenceOps, saveEvidenceOps } from './evidenceStore';
import { loadOps, saveOps } from './store';

const DAYS = 7;
const MAX_ITEMS = 300;

function olderThan(ts: number, days: number) {
  return ts < Date.now() - days * 24 * 60 * 60 * 1000;
}

export async function cleanupOfflineQueues() {
  // 1) Evidências
  const ev = await loadEvidenceOps();
  const evKept = ev
    .filter((o) => {
      if (o.status === 'PENDING' || o.status === 'FAILED' || o.status
=== 'UPLOADING') return true;
      // COMMITTED: manter só 7 dias
      return !olderThan(o.updatedAt || o.createdAt, DAYS);
    })
    .sort((a, b) => (b.updatedAt || b.createdAt) - (a.updatedAt ||
a.createdAt))
    .slice(0, MAX_ITEMS);

  await saveEvidenceOps(evKept);

  // 2) Start-contract / outras ops
  const ops = await loadOps();
  const opsKept = ops
    .filter((o: any) => {
      if (o.status === 'PENDING' || o.status === 'FAILED') return
true;
      // SENT: manter só 7 dias
      return !olderThan(o.createdAt || 0, DAYS);
    })
    .sort((a: any, b: any) => (b.createdAt || 0) - (a.createdAt || 0))
    .slice(0, MAX_ITEMS);

  await saveOps(opsKept);

  return { ok: true, evidence: { before: ev.length, after:
evKept.length }, ops: { before: ops.length, after: opsKept.length } };
}
```

1.2 Rodar cleanup automaticamente (ao abrir e 1x/dia)

```
// src/offline/useOfflineMaintenance.ts
import { useEffect } from 'react';
import AsyncStorage from '@react-native-async-storage/async-storage';
import { cleanupOfflineQueues } from './cleanup';

const KEY = 'offline_maintenance_last_run_v1';

export function useOfflineMaintenance() {
  useEffect(() => {
    (async () => {
      const last = await AsyncStorage.getItem(KEY);
      const lastTs = last ? Number(last) : 0;

      const oneDay = 24 * 60 * 60 * 1000;
      if (!lastTs || Date.now() - lastTs > oneDay) {
        await cleanupOfflineQueues();
        await AsyncStorage.setItem(KEY, String(Date.now()));
      }
    })();
  }, []);
}
```

No `App.tsx`:

```
import { useOfflineFlush } from './src/offline/useOfflineFlush';
import { useOfflineMaintenance } from
'./src/offline/useOfflineMaintenance';

export default function App() {
  useOfflineFlush();
  useOfflineMaintenance();
  return null; // seu NavigationContainer
}
```

---

2) Auditoria read-only: GET /api/audit/events

Objetivo

Um auditor (AUDITOR) ou admin/manager consegue:

- filtrar por tipo (ex.: `EVIDENCE_COMMITTED`, `CONTRACT_CREATED...`)
- filtrar por `contractId` (muito importante)
- limitar e paginar

Coleção usada

Você já criou `audit_events` no commit da evidência. Vamos padronizar:

- `audit_events/{auditId}` com campos:
  - `tenantId`
  - `type`
  - `actor`
  - `ref` (ex.: `{ contractId, evidenceId }`)

- meta
- ts

2.1 DTO de query (opcional)

```
// src/audit/dto/audit-query.dto.ts
import { IsOptional, IsString } from 'class-validator';

export class AuditQueryDto {
  @IsOptional() @IsString()
  type?: string;

  @IsOptional() @IsString()
  contractId?: string;

  @IsOptional() @IsString()
  afterTs?: string; // "1730000000000"

  @IsOptional() @IsString()
  limit?: string; // "50"
}
```

2.2 AuditService

```
// src/audit/audit.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

function clamp(n: number, min: number, max: number) {
  return Math.max(min, Math.min(max, n));
}

@Injectable()
export class AuditService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async listEvents(actor: any, q: { type?: string; contractId?:
string; afterTs?: string; limit?: string }) {
    const limit = clamp(Number(q.limit ?? 50), 1, 200);
    const afterTs = q.afterTs ? Number(q.afterTs) : null;

    // ACL: read-only auditoria
    // Recomendado: só AUDITOR/ADMIN/MANAGER
    const role = actor.role;
    const allowed = role === 'AUDITOR' || role === 'ADMIN' || role ===
'MANAGER';
    if (!allowed) throw new Error('AUDIT_FORBIDDEN');

    let query: FirebaseFirestore.Query =
this.db.collection('audit_events')
      .where('tenantId', '==', actor.tenantId);

    if (q.type) query = query.where('type', '==', q.type);

    // Como ref.contractId nem sempre indexa bem, recomendo gravar
também contractId no topo em eventos de contrato.
    // Se você já gravou só em ref.contractId, a query abaixo pode
exigir índice.
```

```
    if (q.contractId) query = query.where('contractId', '==',
q.contractId);

    // paginação por tempo
    if (afterTs != null) query = query.where('ts', '>=', afterTs);

    query = query.orderBy('ts', 'desc').limit(limit);

    const snap = await query.get();
    const items = snap.docs.map((d) => ({ auditId: d.id, ...d.data()
}));

    const nextAfterTs = items.length ? String(items[items.length -
1].ts) : null;

    return { items, nextAfterTs };
  }
}
```

Recomendação importante: grave `contractId` também no topo do documento de auditoria quando o evento for de contrato. Ex.:
`{ contractId, ... ref: { contractId } }`
Isso torna o filtro MUITO mais rápido.

2.3 AuditController

```
// src/audit/audit.controller.ts
import { Controller, Get, Query, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { AuditService } from './audit.service';

@Controller('/api/audit')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class AuditController {
  constructor(private readonly svc: AuditService) {}

  @Get('/events')
  @Roles('ADMIN','MANAGER','AUDITOR')
  @Permissions('audit.read')
  async list(@Actor() actor: any, @Query() q: any) {
    return this.svc.listEvents(actor, q);
  }
}
```

2.4 AuditModule

```
// src/audit/audit.module.ts
import { Module } from '@nestjs/common';
import { firestoreProvider } from '../common/firestore.provider';
import { AuditController } from './audit.controller';
import { AuditService } from './audit.service';
```

```
@Module({
  providers: [firestoreProvider, AuditService],
  controllers: [AuditController],
})
export class AuditModule {}
```

---

3) Endpoint mais útil: GET /api/contracts/:id/audit

O usuário no campo normalmente quer ver “o que aconteceu com este contrato”.

3.1 Controller

```
// src/contracts/contracts-audit.controller.ts
import { Controller, Get, Param, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { ContractsAuditService } from './contracts-audit.service';

@Controller('/api/contracts')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class ContractsAuditController {
  constructor(private readonly svc: ContractsAuditService) {}

  @Get('/:id/audit')

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','SUPPLIER','AUDITOR','T
ECHNICIAN','LOGISTICS','INVESTOR')
  @Permissions('contracts.read')
  async getAudit(@Actor() actor: any, @Param('id') contractId: string)
{
    return this.svc.getContractAudit(actor, contractId);
  }
}
```

3.2 Service

```
// src/contracts/contracts-audit.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

@Injectable()
export class ContractsAuditService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async getContractAudit(actor: any, contractId: string) {
    // ACL do contrato
    const cRef = this.db.collection('contracts').doc(contractId);
    const cSnap = await cRef.get();
    if (!cSnap.exists) throw new Error('CONTRACT_NOT_FOUND');
    const c = cSnap.data()!;
```

```
    const allowed = c.buyerTenantId === actor.tenantId ||
c.supplierTenantId === actor.tenantId;
    const roleAllowed = actor.role === 'AUDITOR' || actor.role ===
'ADMIN' || actor.role === 'MANAGER';
    if (!allowed && !roleAllowed) throw new
Error('CONTRACT_FORBIDDEN');

    // busca eventos (preferível ler contract_events, pois é "linha do
tempo" do contrato)
    const evSnap = await this.db.collection('contract_events')
      .where('contractId', '==', contractId)
      .orderBy('ts', 'desc')
      .limit(200)
      .get();

    const items = evSnap.docs.map((d) => ({ eventId: d.id, ...d.data()
}));
    return { contractId, items };
  }
}
```

`contract_events` é a trilha do contrato.
`audit_events` é trilha geral do sistema (inclui evidências globais, etc.).
Os dois são úteis.

---

4) Permissions JSON mínimo (para fechar o RBAC)

Adicione:

- `audit.read`
- (já tinha) `evidence.read/write`, `contracts.read/write`, `market.read/reserve`

Exemplo rápido de permissões:

```
{
  "permissions": [
    "audit.read",
    "evidence.read",
    "evidence.write",
    "contracts.read",
    "contracts.write",
    "market.read",
    "market.reserve"
  ]
}
```

Recomendação de papéis:

- `AUDITOR`: `audit.read`, `contracts.read`, `evidence.read`
- `ADMIN/MANAGER`: todos acima
- `OPERATOR/PRODUCER/SUPPLIER`: sem `audit.read` (para não expor tudo), mas podem ver `contracts/:id/audit` se tiverem acesso ao contrato.

---

5) Índices Firestore recomendados

Crie índices:

- `audit_events`: `tenantId + ts(desc)`
- `audit_events`: `tenantId + type + ts(desc)`
- `audit_events`: `tenantId + contractId + ts(desc)` *(se você salvar contractId no topo)*
- `contract_events`: `contractId + ts(desc)`

Se você quiser, eu também posso te mandar:

- **Tela mobile "Auditoria do contrato"** (timeline com ícones grandes, perfeito para baixa leitura)
- e um **endpoint de "parecer do auditor"** (read-only + assinatura digital) sem alterar registro crítico (apenas adiciona um parecer append-only).

Perfeito — abaixo vai:

1. **Tela mobile "Auditoria do contrato"** (timeline com ícones grandes e texto simples)
2. **Endpoint de parecer do auditor** (append-only, sem alterar registros críticos)
3. **Tela mobile para emitir parecer** (com voz opcional)

Tudo alinhado com: **read-only nos registros críticos** + **parecer como evento novo**.

---

1) Tela mobile: Auditoria do contrato (timeline simples)

1.1 Mapeamento de eventos → ícone + texto curto

```
// src/screens/audit/eventMap.ts
export function mapEvent(e: any) {
  const t = e.type;
  if (t === 'CONTRACT_CREATED_FROM_RESERVATION') return { icon: '📜',
label: 'Contrato criado' };
  if (t === 'ACCEPTED') return { icon: '✅', label: 'Aceite
confirmado' };
  if (t === 'DISPATCH_CONFIRMED') return { icon: '🚚', label:
'Expedição confirmada' };
  if (t === 'DELIVERY_CONFIRMED') return { icon: '📦', label: 'Entrega
confirmada' };
  if (t === 'FISCAL_DOC_COMMITTED') return { icon: '📄', label:
'Documento fiscal enviado' };
  if (t === 'FISCAL_DOC_VERIFIED') return { icon: '📜', label: 'NF-e
verificada' };
```

```
  if (t === 'SETTLEMENT_TRANCHE_RELEASED') return { icon: '💰', label:
'Pagamento liberado' };
  if (t === 'RULE_BLOCKED') return { icon: '⛔', label: 'Bloqueado por
regra' };
  if (t === 'AUDITOR_OPINION_ADDED') return { icon: '⚖️', label:
'Parecer do auditor' };
  return { icon: '⏳', label: t };
}
```

1.2 Tela timeline

```
// src/screens/ContractAuditScreen.tsx
import React from 'react';
import { FlatList, Text, View } from 'react-native';
import { useQuery } from '@tanstack/react-query';
import { api } from '../services/api';
import { BigButton } from '../ui/BigButton';
import { mapEvent } from './audit/eventMap';

function fmt(ts?: number) {
  if (!ts) return '';
  const d = new Date(ts);
  return `${d.toLocaleDateString()} ${d.toLocaleTimeString()}`;
}

export function ContractAuditScreen({ route, navigation }: any) {
  const { contractId } = route.params;

  const q = useQuery({
    queryKey: ['contractAudit', contractId],
    queryFn: async () => (await
api.get(`/api/contracts/${contractId}/audit`)).data,
  });

  const items = q.data?.items || [];

  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 24, fontWeight: '800', marginBottom: 8
}}>
        Auditoria do Contrato
      </Text>
      <Text style={{ fontSize: 16, opacity: 0.8, marginBottom: 12 }}>
        Linha do tempo (não pode ser alterada).
      </Text>

      <BigButton
        icon="⚖️"
        title="Emitir parecer (auditor)"
        subtitle="Adicionar parecer sem alterar registros"
        onPress={() => navigation.navigate('AuditorOpinion', {
contractId })}
      />

      <FlatList
        data={items}
        keyExtractor={(it) => it.eventId}
        refreshing={q.isFetching}
        onRefresh={() => q.refetch()}
        renderItem={({ item }) => {
          const m = mapEvent(item);
          return (
```

```
            <BigButton
              icon={m.icon}
              title={m.label}
              subtitle={`${fmt(item.ts)} • ${item.actor?.role ?? ''} •
${item.actor?.uid ?? ''}`}
              onPress={() => navigation.navigate('AuditEventDetail', {
event: item })}
            />
          );
        }}
      />
    </View>
  );
}
```

1.3 Tela detalhe do evento (bem simples)

```
// src/screens/AuditEventDetailScreen.tsx
import React from 'react';
import { ScrollView, Text, View } from 'react-native';
import { Card } from '../ui/Card';

export function AuditEventDetailScreen({ route }: any) {
  const { event } = route.params;

  return (
    <ScrollView style={{ padding: 16 }}>
      <Text style={{ fontSize: 24, fontWeight: '800'
}}>{event.type}</Text>
      <Text style={{ fontSize: 16, opacity: 0.8, marginTop: 8 }}>
        {new Date(event.ts).toLocaleString()}
      </Text>

      <Card>
        <Text style={{ fontSize: 16, fontWeight: '700'
}}>Responsável</Text>
        <Text style={{ fontSize: 16 }}>{event.actor?.role} •
{event.actor?.uid}</Text>
      </Card>

      <Card>
        <Text style={{ fontSize: 16, fontWeight: '700'
}}>Detalhes</Text>
        <Text style={{ fontSize: 14, opacity: 0.85 }}>
          {JSON.stringify(event.meta ?? {}, null, 2)}
        </Text>
      </Card>
    </ScrollView>
  );
}
```

---

2) Endpoint de “parecer do auditor” (append-only)

2.1 Modelo Firestore

Coleção: `auditor_opinions/{opinionId}` (append-only)

Campos:

- `tenantId`
- `contractId`
- `opinionId`
- `decision`: `"APPROVE" | "REJECT" | "REQUEST_FIX"`
- `summary` (texto curto)
- `attachments` (paths de evidência se quiser)
- `createdAt`
- `actor`

E também cria um evento em `contract_events`:

- `type: AUDITOR_OPINION_ADDED`

2.2 DTO

```
// src/audit/dto/auditor-opinion.dto.ts
import { IsArray, IsIn, IsNotEmpty, IsOptional, IsString } from
'class-validator';

export class AuditorOpinionDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsIn(['APPROVE','REJECT','REQUEST_FIX'])
  decision: 'APPROVE'|'REJECT'|'REQUEST_FIX';

  @IsString() @IsNotEmpty()
  summary: string;

  @IsOptional() @IsArray()
  attachments?: string[]; // storagePaths de evidências (opcional)
}
```

2.3 Service (idempotente + ACL + append-only)

```
// src/audit/auditor-opinion.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';
import { idemKey } from '../common/idempotency/idempotency.util';

@Injectable()
export class AuditorOpinionService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async addOpinion(actor: any, contractId: string, dto: any) {
    const now = Date.now();
    const key = idemKey(actor.tenantId, actor.uid,
dto.clientRequestId, 'audit.opinion.add');
    const idemRef = this.db.collection('idempotency').doc(key);

    if (actor.role !== 'AUDITOR' && actor.role !== 'ADMIN' &&
actor.role !== 'MANAGER') {
      throw new Error('AUDITOR_ONLY');
    }
```

```
    const cRef = this.db.collection('contracts').doc(contractId);

    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const cSnap = await tx.get(cRef);
      if (!cSnap.exists) throw new Error('CONTRACT_NOT_FOUND');
      const c = cSnap.data()!;

      // auditor só pode atuar no tenant do contrato (padrão)
      if (c.tenantId !== actor.tenantId && c.buyerTenantId !==
actor.tenantId) {
        throw new Error('CONTRACT_FORBIDDEN');
      }

      const opinionId = makeId('opn');
      tx.create(this.db.collection('auditor_opinions').doc(opinionId),
{
        opinionId,
        tenantId: actor.tenantId,
        contractId,
        decision: dto.decision,
        summary: dto.summary,
        attachments: dto.attachments ?? [],
        actor: { uid: actor.uid, role: actor.role },
        createdAt: now,
      });

      // evento no contrato (append-only)
      const evtId = makeId('evt');
      tx.create(this.db.collection('contract_events').doc(evtId), {
        eventId: evtId,
        tenantId: c.tenantId,
        contractId,
        type: 'AUDITOR_OPINION_ADDED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { opinionId, decision: dto.decision },
        ts: now
      });

      // auditoria geral
      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId: c.tenantId,
        contractId, // importante no topo!
        type: 'AUDITOR_OPINION_ADDED',
        actor: { uid: actor.uid, role: actor.role },
        ref: { opinionId, contractId },
        meta: { decision: dto.decision },
        ts: now
      });

      const out = { opinionId, contractId, decision: dto.decision };
      tx.create(idemRef, {
        tenantId: actor.tenantId,
        uid: actor.uid,
        scope: 'audit.opinion.add',
        clientRequestId: dto.clientRequestId,
        createdAt: now,
```

```
        result: out
      });

      return out;
    });
  }
}
```

2.4 Controller

```
// src/audit/auditor-opinion.controller.ts
import { Body, Controller, Param, Post, UseGuards } from
'@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { AuditorOpinionService } from './auditor-opinion.service';
import { AuditorOpinionDto } from './dto/auditor-opinion.dto';

@Controller('/api/contracts')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class AuditorOpinionController {
  constructor(private readonly svc: AuditorOpinionService) {}

  @Post('/:id/auditor-opinion')
  @Roles('AUDITOR','ADMIN','MANAGER')
  @Permissions('audit.write')
  async add(@Actor() actor: any, @Param('id') contractId: string,
@Body() dto: AuditorOpinionDto) {
    return this.svc.addOpinion(actor, contractId, dto);
  }
}
```

---

3) Tela mobile: Emitir parecer (com voz opcional)

3.1 UI simples (baixa leitura)

- 3 botões grandes: ✅ Aprovar / ❌ Reprovar / 🛠️ Pedir ajuste
- Campo “Resumo” (pode preencher por voz)
- Botão “Enviar parecer”

```
// src/screens/AuditorOpinionScreen.tsx
import React, { useState } from 'react';
import { Text, TextInput, View } from 'react-native';
import { BigButton } from '../ui/BigButton';
import { api } from '../services/api';
import { v4 as uuid } from 'uuid';

export function AuditorOpinionScreen({ route, navigation }: any) {
  const { contractId } = route.params;
  const [decision, setDecision] =
useState<'APPROVE'|'REJECT'|'REQUEST_FIX'>('APPROVE');
  const [summary, setSummary] = useState('');
```

```
  async function send() {
    await api.post(`/api/contracts/${contractId}/auditor-opinion`, {
      clientRequestId: uuid(),
      decision,
      summary,
    });
    navigation.goBack();
  }

  async function voiceFill() {
    // opcional: gravar áudio + transcrever no backend
    // aqui só simula:
    setSummary('Parecer: documentação fiscal conferida, evidência ok,
liberar continuidade.');
  }

  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 24, fontWeight: '800' }}>Parecer do
Auditor</Text>
      <Text style={{ fontSize: 16, opacity: 0.8, marginTop: 8 }}>
        O parecer é registrado e não altera dados críticos.
      </Text>

      <View style={{ marginTop: 16 }}>
        <BigButton icon="✅" title="Aprovar"
subtitle={decision===='APPROVE'?'Selecionado':''} onPress={() =>
setDecision('APPROVE')} />
        <BigButton icon="❌" title="Reprovar"
subtitle={decision===='REJECT'?'Selecionado':''} onPress={() =>
setDecision('REJECT')} />
        <BigButton icon="🛠️" title="Pedir ajuste"
subtitle={decision===='REQUEST_FIX'?'Selecionado':''} onPress={() =>
setDecision('REQUEST_FIX')} />

        <Text style={{ fontSize: 18, fontWeight: '700', marginTop: 12
}}>Resumo</Text>
        <TextInput
          value={summary}
          onChangeText={setSummary}
          placeholder="Digite ou use voz…"
          multiline
          style={{
            borderWidth: 1,
            borderRadius: 12,
            padding: 12,
            minHeight: 110,
            marginTop: 8,
            fontSize: 16
          }}
        />

        <BigButton icon="🎙️" title="Preencher por voz" subtitle="Falar
e converter em texto" onPress={voiceFill} />
        <BigButton icon="⚖️" title="Enviar parecer"
subtitle="Registrar na auditoria" onPress={send}
disabled={!summary.trim()} />
      </View>
    </View>
```

```
  );
}
```

---

4) Permissions para parecer

Adicione:

- `audit.write`

Recomendação:

- `AUDITOR`: `audit.read`, `audit.write`, `contracts.read`, `evidence.read`
- `ADMIN/MANAGER`: tudo

---

5) Índices Firestore

- `contract_events`: `contractId + ts(desc)`
- `auditor_opinions`: `contractId + createdAt(desc)`
- `audit_events`: `tenantId + contractId + ts(desc)`

Se você quiser, eu fecho o último ponto de "perfeição jurídica":
✅ **assinatura do parecer** (hash do texto + carimbo de tempo + device attestation opcional) e anexação de evidência (áudio + PDF).

Beleza — vamos fechar o "nível jurídico" do parecer com:

1. **Assinatura técnica do parecer** (hash + carimbo de tempo + cadeia)
2. **Anexos do parecer** (PDF/áudio/foto) como evidência vinculada
3. **Atestado de dispositivo** (opcional, se você quiser elevar o lastro)
4. Tudo **append-only** (não altera nada crítico)

---

1) Modelo de assinatura do parecer

1.1 O que vamos registrar (mínimo robusto)

Quando o auditor envia o parecer, o backend cria:

- `canonicalText` (texto normalizado)
- `opinionHash` = SHA-256(canonicalText)
- `signature` (assinado pelo servidor)
- `timestamp` (server time)
- `prevHash` (hash do último parecer do contrato) → **cadeia**

- `chainHash` = SHA-256(prevHash + opinionHash + timestamp + actorUid)

✅ Isso cria uma “blockchain leve” só de pareceres (append-only).

Observação: isso não é “assinatura ICP-Brasil”. É **assinatura técnica e lastro probatório** do sistema (forte para auditoria e integridade).

---

2) Canonicalização do texto (para não mudar hash por espaços)

```
// src/common/crypto/canonical.ts
export function canonicalize(text: string) {
  return text
    .normalize('NFKC')
    .replace(/\r\n/g, '\n')
    .replace(/[ \t]+/g, ' ')
    .trim();
}
```

---

3) StorageService (helpers de hash / assinatura)

3.1 Assinatura do servidor (HMAC)

Mais simples e muito bom: HMAC-SHA256 com secret.

```
// src/common/crypto/sign.ts
import crypto from 'crypto';

export function sha256hex(input: string) {
  return crypto.createHash('sha256').update(input).digest('hex');
}

export function hmac256hex(secret: string, input: string) {
  return crypto.createHmac('sha256',
secret).update(input).digest('hex');
}
```

Guarde `OPINION_SIGNING_SECRET` no `.env`/Secret Manager.

---

4) Atualizar DTO para anexos + optional device attestation

```
// src/audit/dto/auditor-opinion.dto.ts
import { IsArray, IsIn, IsNotEmpty, IsOptional, IsString } from
'class-validator';

export class AuditorOpinionDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsIn(['APPROVE','REJECT','REQUEST_FIX'])
  decision: 'APPROVE'|'REJECT'|'REQUEST_FIX';
```

```
  @IsString() @IsNotEmpty()
  summary: string;

  @IsOptional() @IsArray()
  attachments?: string[]; // storagePaths (evidence)

  // opcional: atestado/assinatura do dispositivo (se implementar)
  @IsOptional()
  deviceAttestation?: {
    provider: 'APPLE'|'ANDROID';
    token: string;
  };
}
```

---

5) Service do parecer com hash + cadeia + assinatura

```
// src/audit/auditor-opinion.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';
import { idemKey } from '../common/idempotency/idempotency.util';
import { canonicalize } from '../common/crypto/canonical';
import { sha256hex, hmac256hex } from '../common/crypto/sign';

@Injectable()
export class AuditorOpinionService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async addOpinion(actor: any, contractId: string, dto: any) {
    const now = Date.now();

    if (actor.role !== 'AUDITOR' && actor.role !== 'ADMIN' &&
actor.role !== 'MANAGER') {
      throw new Error('AUDITOR_ONLY');
    }

    const key = idemKey(actor.tenantId, actor.uid,
dto.clientRequestId, 'audit.opinion.add');
    const idemRef = this.db.collection('idempotency').doc(key);

    const cRef = this.db.collection('contracts').doc(contractId);

    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const cSnap = await tx.get(cRef);
      if (!cSnap.exists) throw new Error('CONTRACT_NOT_FOUND');
      const c = cSnap.data()!;

      if (c.tenantId !== actor.tenantId && c.buyerTenantId !==
actor.tenantId) {
        throw new Error('CONTRACT_FORBIDDEN');
      }

      // 1) descobre prevHash do último parecer do contrato
      // recomendação: manter um ponteiro no contrato:
c.audit?.lastOpinionHash
      const prevHash = c.audit?.lastOpinionChainHash ?? 'GENESIS';
```

```
      // 2) canonicaliza e faz hash
      const canonicalText = canonicalize(dto.summary);
      const opinionHash = sha256hex(canonicalText);

      // 3) hash encadeado
      const chainInput =
`${prevHash}|${opinionHash}|${now}|${actor.uid}|${dto.decision}`;
      const chainHash = sha256hex(chainInput);

      // 4) assinatura do servidor (HMAC)
      const secret = process.env.OPINION_SIGNING_SECRET ||
'CHANGE_ME';
      const serverSignature = hmac256hex(secret, chainHash);

      // 5) grava parecer append-only
      const opinionId = makeId('opn');
      tx.create(this.db.collection('auditor_opinions').doc(opinionId),
{
        opinionId,
        tenantId: actor.tenantId,
        contractId,
        decision: dto.decision,
        summary: dto.summary,
        canonicalText,
        opinionHash,
        prevHash,
        chainHash,
        serverSignature,
        attachments: dto.attachments ?? [],
        deviceAttestation: dto.deviceAttestation ?? null,
        actor: { uid: actor.uid, role: actor.role },
        createdAt: now,
      });

      // 6) evento do contrato (append-only)
      const evtId = makeId('evt');
      tx.create(this.db.collection('contract_events').doc(evtId), {
        eventId: evtId,
        tenantId: c.tenantId,
        contractId,
        type: 'AUDITOR_OPINION_ADDED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { opinionId, decision: dto.decision, opinionHash,
chainHash },
        ts: now
      });

      // 7) auditoria geral (append-only)
      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId: c.tenantId,
        contractId, // topo para index
        type: 'AUDITOR_OPINION_ADDED',
        actor: { uid: actor.uid, role: actor.role },
        ref: { opinionId, contractId },
        meta: { decision: dto.decision, opinionHash, chainHash },
        ts: now
      });
```

```
      // 8) atualiza ponteiro no contrato (NÃO altera histórico
crítico; só salva ponteiro)
      tx.set(cRef, {
        audit: {
          lastOpinionHash: opinionHash,
          lastOpinionChainHash: chainHash,
          lastOpinionAt: now
        },
        updatedAt: now
      }, { merge: true });

      const out = { opinionId, contractId, decision: dto.decision,
chainHash, serverSignature };
      tx.create(idemRef, {
        tenantId: actor.tenantId,
        uid: actor.uid,
        scope: 'audit.opinion.add',
        clientRequestId: dto.clientRequestId,
        createdAt: now,
        result: out
      });

      return out;
    });
  }
}
```

✅ Observação: atualizar um "ponteiro" (`audit.lastOpinionChainHash`) é aceitável porque:

- não muda eventos antigos
- só facilita leitura/verificação
- o lastro continua nos documentos append-only

---

6) Verificação de integridade (endpoint opcional)

Você pode ter:
`GET /api/contracts/:id/audit/verify`

Ele:

- lê todos os `auditor_opinions` daquele contrato
- re-calcula `chainHash` em sequência
- valida `serverSignature`

Se quiser eu mando esse endpoint também, mas já está bem completo.

---

7) Anexos no parecer (PDF/áudio/foto)

Fluxo recomendado:

1. app envia anexos como evidência offline (igual fizemos) → gera `storagePath`
2. na tela de parecer, o auditor "anexa" os `storagePaths`
3. backend salva `attachments[]` no parecer

✅ Assim o parecer fica "auto-contido" (texto + anexos + hash + cadeia).

---

8) Device attestation (opcional, nível máximo)

Se quiser elevar o lastro:

- iOS: App Attest / DeviceCheck (Apple)
- Android: Play Integrity API

O app manda `deviceAttestation.token` e o backend valida com o provedor e salva:

- `deviceAttestation: { provider, verdict, deviceIdHash }`

Se você quiser, eu te passo o esqueleto de validação (mas depende das chaves do seu projeto).

---

9) UI do auditor para anexos + voz (simples)

Na tela `AuditorOpinionScreen`, adicione:

- "Anexar foto/áudio/PDF" → cria evidência (offline) e lista `storagePaths`
- "Gravar áudio do parecer" → vira anexo (AUDIO)

E manda no `attachments`.

Se você quiser, agora eu consigo fechar a parte final do "núcleo Ciclo+":
✅ **RulesEngine** bloqueando emissão de NF/transferência quando há ativos de terceiros em custódia (split obrigatório + travas por CAR/IE) — isso casa com o seu requisito de impedir fraude.

Perfeito — vamos fechar o **núcleo de bloqueios (RulesEngine)** para:

- **travar emissão de NF/transferência/venda** quando existir **ativo de terceiro em custódia** sem pagamento/split/autorizações
- **bloquear por CAR / Inscrição Estadual (IE)** do imóvel quando houver **descumprimento** (contrato/ambiental)
- garantir **split/escrow obrigatório** em PDV/marketplace quando houver múltiplos fornecedores
- manter tudo **multi-tenant** + **auditoria append-only** + **máquina de estados**

Vou te entregar:

1. **Modelo de dados mínimo** (custódia, propriedade/CAR, restrições)
2. **RulesEngine** (NestJS) com regras plugáveis
3. **Guards** nos serviços: NF, transferência, PDV sale, estoque
4. **Eventos de bloqueio** (auditoria) + motivos "explicáveis" pro usuário
5. **Exemplos de regras** já prontas

---

1) Modelo de dados mínimo (Firestore)

1.1 Propriedade vinculada ao CAR / IE

```
properties/{propertyId}

{
  "tenantId": "TENANT_X",
  "propertyId": "prop_123",
  "name": "Fazenda Modelo",
  "car": { "code": "MT-1234567-ABCDE...", "linked": true },
  "stateRegistration": { "ie": "123.456.789.000", "linked": true },
  "geo": { "lat": -15.6, "lng": -56.1 }
}
```

1.2 Restrições (bloqueios) por CAR/IE (append-only + status)

```
property_restrictions/{restrictionId}

{
  "tenantId": "TENANT_X",
  "restrictionId": "rst_001",
  "propertyId": "prop_123",
  "scope": "COMMERCIAL_BLOCK",
  "reasonCode": "CONTRACT_DEFAULT|ENV_LIMIT|JUDICIAL_HOLD",
  "reasonText": "Descumprimento de contrato: contrato ctr_...",
  "targets": ["NF_EMISSION", "TRANSFER", "SALE", "PROJECT_CREATE"],
  "status": "ACTIVE",
  "createdAt": 1730000000000,
  "createdBy": { "uid": "...", "role": "AUDITOR" }
}
```

1.3 Custódia de ativos de terceiros (o que trava NF/transferência)

```
custody_assets/{custodyId}

{
  "tenantId": "TENANT_HOST",
  "custodyId": "cst_001",
  "propertyId": "prop_123",
  "assetType": "BOVINE|GOODS|BULK",
  "ownerTenantId": "TENANT_OWNER",
  "status": "IN_CUSTODY|RELEASED",
  "identityMode": "INDIVIDUAL|PROPORTIONAL",

  "identities": [
    { "kind": "EID|BRINCO|RFID|SERIAL", "value": "BR123" }
  ],
```

```
  "quota": {
    "basis": "NF|WEIGHT|VOLUME",
    "ownerPercent": 60,
    "hostPercent": 40
  },

  "lock": {
    "blockNF": true,
    "blockTransfer": true,
    "requiresSplit": true
  },

  "releaseConditions": {
    "requiresEscrowPaid": true,
    "requiresOwnerApproval": false
  },

  "createdAt": 1730000000000
}
```

1.4 “Documento fiscal / emissão NF” como evento (não alterável)

```
fiscal_issues/{issueId}

{
  "tenantId": "TENANT_HOST",
  "issueId": "nfi_001",
  "propertyId": "prop_123",
  "type": "NF_EMISSION_REQUEST",
  "payload": { "nfKey": "...", "items": [ ... ] },
  "state": "PENDING|BLOCKED|APPROVED|ISSUED",
  "blockReason": null,
  "createdAt": 1730000000000
}
```

---

2) RulesEngine (NestJS) plugável

2.1 Tipos

```
// src/rules/rules.types.ts
export type RuleContext = {
  actor: { uid: string; role: string; tenantId: string; permissions:
string[] };
  tenantId: string;
  operation: string; // ex: "NF_EMIT", "TRANSFER_ASSET", "PDV_SALE"
  propertyId?: string;
  payload?: any;     // dados da operação
};

export type RuleResult =
  | { ok: true }
  | { ok: false; code: string; message: string; details?: any };

export interface Rule {
  id: string;
  applies(ctx: RuleContext): boolean;
  evaluate(ctx: RuleContext): Promise<RuleResult>;
}
```

2.2 Engine

```
// src/rules/rules.engine.ts
import { Injectable } from '@nestjs/common';
import { Rule, RuleContext, RuleResult } from './rules.types';

@Injectable()
export class RulesEngine {
  constructor(private readonly rules: Rule[]) {}

  async run(ctx: RuleContext): Promise<{ ok: true } | { ok: false;
violations: RuleResult[] }> {
    const violations: RuleResult[] = [];

    for (const rule of this.rules) {
      if (!rule.applies(ctx)) continue;
      const res = await rule.evaluate(ctx);
      if (!res.ok) violations.push(res);
    }

    if (violations.length) return { ok: false, violations };
    return { ok: true };
  }
}
```

Em NestJS, você injeta uma lista de Rules usando providers (te mostro no módulo já já).

---

3) Regras essenciais (as suas)

3.1 Regra: Propriedade bloqueada por CAR/IE

```
// src/rules/rules/propertyCommercialBlock.rule.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../../common/firestore.provider';
import { Rule, RuleContext, RuleResult } from '../rules.types';

@Injectable()
export class PropertyCommercialBlockRule implements Rule {
  id = 'propertyCommercialBlock';

  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  applies(ctx: RuleContext) {
    return
['NF_EMIT','TRANSFER_ASSET','PDV_SALE','PROJECT_CREATE'].includes(ctx.
operation)
      && !!ctx.propertyId;
  }

  async evaluate(ctx: RuleContext): Promise<RuleResult> {
    const snap = await this.db.collection('property_restrictions')
      .where('tenantId', '==', ctx.tenantId)
      .where('propertyId', '==', ctx.propertyId!)
      .where('status', '==', 'ACTIVE')
      .limit(50)
      .get();

    const restrictions = snap.docs.map(d => d.data());
```

```
    // verifica se alguma restrição atinge a operação
    const target = ctx.operation === 'NF_EMIT' ? 'NF_EMISSION'
      : ctx.operation === 'TRANSFER_ASSET' ? 'TRANSFER'
      : ctx.operation === 'PDV_SALE' ? 'SALE'
      : 'PROJECT_CREATE';

    const hit = restrictions.find(r => (r.targets ||
[]).includes(target));
    if (!hit) return { ok: true };

    return {
      ok: false,
      code: 'PROPERTY_BLOCKED',
      message: 'Operação bloqueada por restrição (CAR/IE).',
      details: { restrictionId: hit.restrictionId, reasonCode:
hit.reasonCode, reasonText: hit.reasonText }
    };
  }
}
```

3.2 Regra: Ativo de terceiro em custódia trava NF/transferência

```
// src/rules/rules/custodyBlocksFiscal.rule.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../../common/firestore.provider';
import { Rule, RuleContext, RuleResult } from '../rules.types';

@Injectable()
export class CustodyBlocksFiscalRule implements Rule {
  id = 'custodyBlocksFiscal';

  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  applies(ctx: RuleContext) {
    return
['NF_EMIT','TRANSFER_ASSET','PDV_SALE'].includes(ctx.operation) &&
!!ctx.propertyId;
  }

  async evaluate(ctx: RuleContext): Promise<RuleResult> {
    const snap = await this.db.collection('custody_assets')
      .where('tenantId', '==', ctx.tenantId)
      .where('propertyId', '==', ctx.propertyId!)
      .where('status', '==', 'IN_CUSTODY')
      .limit(200)
      .get();

    const custody = snap.docs.map(d => d.data());

    // se houver qualquer custódia com bloqueio relevante
    const blocking = custody.filter(c => {
      if (ctx.operation === 'NF_EMIT') return c.lock?.blockNF ===
true;
      if (ctx.operation === 'TRANSFER_ASSET') return
c.lock?.blockTransfer === true;
      if (ctx.operation === 'PDV_SALE') return c.lock?.requiresSplit
=== true;
      return false;
    });

    if (!blocking.length) return { ok: true };
```

```
    return {
      ok: false,
      code: 'CUSTODY_BLOCK',
      message: 'Bloqueado: há ativos de terceiros em custódia sem
liberação/split.',
      details: {
        count: blocking.length,
        owners: blocking.map(b => b.ownerTenantId).slice(0, 20),
        sample: blocking.slice(0, 5).map(b => ({ custodyId:
b.custodyId, assetType: b.assetType, identityMode: b.identityMode }))
      }
    };
  }
}
```

3.3 Regra: Venda PDV exige split por identidade (produto fornecedor)

```
// src/rules/rules/pdvSplitRequired.rule.ts
import { Rule, RuleContext, RuleResult } from '../rules.types';

export class PdvSplitRequiredRule implements Rule {
  id = 'pdvSplitRequired';

  applies(ctx: RuleContext) {
    return ctx.operation === 'PDV_SALE';
  }

  async evaluate(ctx: RuleContext): Promise<RuleResult> {
    // payload: items[{productId, qty, supplierTenantId?,
identityRef?}]
    const items = ctx.payload?.items ?? [];
    if (!Array.isArray(items) || items.length === 0) return { ok: true
};

    // cada item deve ter supplierTenantId e identityRef
    const missing = items.filter((it: any) => !it.supplierTenantId ||
!it.identityRef);
    if (missing.length) {
      return {
        ok: false,
        code: 'PDV_SPLIT_IDENTITY_REQUIRED',
        message: 'Venda bloqueada: item sem fornecedor/identidade
(rastro).',
        details: { missingCount: missing.length }
      };
    }

    // se existir mais de um fornecedor, split obrigatório
    const suppliers = new Set(items.map((it: any) =>
it.supplierTenantId));
    if (suppliers.size > 1 && ctx.payload?.splitPlan?.enabled !==
true) {
      return {
        ok: false,
        code: 'PDV_SPLIT_REQUIRED',
        message: 'Venda bloqueada: múltiplos fornecedores exigem
split/escrow.',
        details: { suppliers: Array.from(suppliers) }
      };
    }

    return { ok: true };
```

```
  }
}
```

---

4) Aplicar RulesEngine nos serviços (exemplo real)

4.1 NF: solicitar emissão de NF (antes de emitir)

```
// src/fiscal/fiscal.service.ts
import { Injectable } from '@nestjs/common';
import { RulesEngine } from '../rules/rules.engine';
import { AuditWriter } from '../audit/audit-writer.service';

@Injectable()
export class FiscalService {
  constructor(
    private readonly rules: RulesEngine,
    private readonly audit: AuditWriter
  ) {}

  async requestNfEmission(actor: any, input: { propertyId: string;
payload: any }) {
    const ctx = {
      actor,
      tenantId: actor.tenantId,
      operation: 'NF_EMIT',
      propertyId: input.propertyId,
      payload: input.payload,
    };

    const res = await this.rules.run(ctx);

    if (!res.ok) {
      await this.audit.block(actor, {
        operation: 'NF_EMIT',
        propertyId: input.propertyId,
        violations: res.violations
      });
      throw new Error(`BLOCKED_${res.violations[0].code}`);
    }

    // continua fluxo normal: criar fiscal_issues PENDING
    // ...
    return { ok: true };
  }
}
```

4.2 PDV Sale (antes de cobrar/escrow)

```
// src/pdv/pdv.service.ts
import { Injectable } from '@nestjs/common';
import { RulesEngine } from '../rules/rules.engine';
import { AuditWriter } from '../audit/audit-writer.service';

@Injectable()
export class PdvService {
  constructor(private readonly rules: RulesEngine, private readonly
audit: AuditWriter) {}

  async createSale(actor: any, input: { propertyId: string; items:
any[]; splitPlan: any }) {
    const ctx = {
      actor,
```

```
      tenantId: actor.tenantId,
      operation: 'PDV_SALE',
      propertyId: input.propertyId,
      payload: { items: input.items, splitPlan: input.splitPlan }
    };

    const res = await this.rules.run(ctx);
    if (!res.ok) {
      await this.audit.block(actor, { operation: 'PDV_SALE',
propertyId: input.propertyId, violations: res.violations });
      throw new Error(`BLOCKED_${res.violations[0].code}`);
    }

    // ok: segue para cobrança escrow e split
    return { ok: true };
  }
}
```

---

5) Auditoria padronizada de bloqueios (explicável)

```
// src/audit/audit-writer.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class AuditWriter {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async block(actor: any, input: { operation: string; propertyId?:
string; contractId?: string; violations: any[] }) {
    const now = Date.now();
    const auditId = makeId('aud');

    await this.db.collection('audit_events').doc(auditId).set({
      auditId,
      tenantId: actor.tenantId,
      contractId: input.contractId ?? null,
      propertyId: input.propertyId ?? null,
      type: 'RULE_BLOCKED',
      actor: { uid: actor.uid, role: actor.role },
      meta: { operation: input.operation, violations: input.violations
},
      ts: now
    });

    return { auditId };
  }
}
```

✅ Isso gera “motivo claro” para o app mostrar:

- “Bloqueado por CAR/IE”
- “Bloqueado por ativos de terceiros em custódia”
- “Venda sem identificação/split”

---

6) Como registrar “trava por descumprimento” (CAR/IE)

Quando ocorrer descumprimento:

- cria `property_restrictions` com `status=ACTIVE`
- e coloca `targets` adequados

Exemplo: travar emissão de NF e transferência:

```
{
  "scope": "COMMERCIAL_BLOCK",
  "reasonCode": "CONTRACT_DEFAULT",
  "targets": ["NF_EMISSION", "TRANSFER", "SALE"]
}
```

Quando resolver:

- não edita o doc antigo (se quiser lastro máximo)
- cria um novo evento “UNBLOCKED” ou marca `status=INACTIVE` (eu recomendo marcar INACTIVE e registrar evento `RESTRICTION_LIFTED` em audit_events)

---

7) RulesModule (injeção das regras)

```
// src/rules/rules.module.ts
import { Module } from '@nestjs/common';
import { firestoreProvider } from '../common/firestore.provider';
import { RulesEngine } from './rules.engine';
import { PropertyCommercialBlockRule } from
'./rules/propertyCommercialBlock.rule';
import { CustodyBlocksFiscalRule } from
'./rules/custodyBlocksFiscal.rule';
import { PdvSplitRequiredRule } from './rules/pdvSplitRequired.rule';

@Module({
  providers: [
    firestoreProvider,
    PropertyCommercialBlockRule,
    CustodyBlocksFiscalRule,
    { provide: 'RULES', useFactory: (a,b) => [a,b, new
PdvSplitRequiredRule()], inject: [PropertyCommercialBlockRule,
CustodyBlocksFiscalRule] },
    {
      provide: RulesEngine,
      useFactory: (rules: any[]) => new RulesEngine(rules),
      inject: ['RULES']
    }
  ],
  exports: [RulesEngine],
})
export class RulesModule {}
```

---

8) Resultado: o que fica garantido no Ciclo+

✅ **Ninguém emite NF** ou transfere ativo se:

- propriedade bloqueada por CAR/IE (restrição ativa)
- houver custódia de terceiros com trava (sem liberação)

✅ **No PDV e marketplace**:

- múltiplos fornecedores → split/escrow obrigatório
- item sem identidade/fornecedor → bloqueia
- tudo registrado em auditoria e explicável

Se você quiser, eu fecho agora a parte "mais difícil" que você pediu:
✅ **mistura de ativos de vários donos**:

- bovinos por brinco/filiação (mãe → filho)
- mercadoria unitária por identidade
- commodity/bulk proporcional por NF/peso/volume
  e como isso entra no split automaticamente.

Fechado — vamos montar o módulo **"Mistura de ativos de vários donos"** com split automático, cobrindo:

1. **Bovinos** (identidade por brinco/RFID + vínculo mãe→filho)
2. **Mercadoria unitária** (identidade por item/serial/lote)
3. **Bulk / não separável** (proporção por NF/peso/volume/cotas)
4. **Split automático** em PDV/marketplace/liquidação
5. **Regras de bloqueio**: não vende / não emite NF / não transfere sem split/escrow

Vou te entregar:

- modelo Firestore
- serviço NestJS para "resolver owners" (owner resolver)
- serviço NestJS para gerar splitPlan
- exemplos prontos de payload PDV e marketplace

---

1) Conceitos-chave (núcleo)

1.1 IdentityRef (todo item/ativo deve ter)

```
type IdentityRef =
  | { kind: 'EID'|'RFID'|'BRINCO'|'SERIAL'|'SKU'|'GTIN'; value: string
}
  | { kind: 'NFE_ITEM'; nfKey: string; itemSeq: number }
```

```
  | { kind: 'LOT'; lot: string }
  | { kind: 'BULK_BATCH'; batchId: string };
```

1.2 Ownership (dono / cota)

```
type OwnershipShare = {
  ownerTenantId: string;
  percent: number; // soma 100
  basis: 'IDENTITY'|'NF'|'WEIGHT'|'VOLUME'|'QUOTA';
  sourceRef?: any; // ex: nfKey, custodyId, batchId
};
```

---

2) Firestore: modelos

2.1 Animal registry (mãe → filho)

```
animals/{animalId}

{
  "tenantId": "TENANT_HOST",
  "animalId": "ani_BR123",
  "identity": { "kind": "BRINCO", "value": "BR123" },
  "species": "BOVINE",
  "sex": "F",
  "birth": { "date": "2025-01-10" },
  "mother": { "animalId": "ani_MAE999", "identity": { "kind":
"BRINCO", "value": "MAE999" } },

  "ownership": {
    "mode": "INDIVIDUAL",
    "ownerTenantId": "TENANT_OWNER",
    "custodyTenantId": "TENANT_HOST",
    "custodyId": "cst_001"
  },

  "status": "IN_CUSTODY",
  "propertyId": "prop_123"
}
```

Regra mãe→filho

Quando nasce um animal “filho”:

- se `mother` tem ownership individual → filho herda o mesmo `ownerTenantId` (até haver contrato que altere)
- registra evento `BIRTH_REGISTERED` + evidência

2.2 Mercadoria unitária (rastreio por unidade / lote)

```
inventory_units/{unitId}

{
  "tenantId": "TENANT_STORE",
```

```
  "unitId": "unit_0001",
  "productId": "prod_sal_25kg",
  "identityRef": { "kind": "SERIAL", "value": "SER123" },
  "source": { "kind": "NFE_ITEM", "nfKey": "....44...", "itemSeq": 3
},
  "supplierTenantId": "TENANT_SUPPLIER",
  "ownerShare": [
    { "ownerTenantId": "TENANT_SUPPLIER", "percent": 100, "basis":
"NF", "sourceRef": { "nfKey": "....44..." } }
  ],
  "status": "IN_STOCK",
  "propertyId": "prop_loj_1"
}
```

Mesmo que o estoque seja do "lojista", o **direito econômico** do fornecedor pode existir (consignação / split). É isso que você quer proteger.

2.3 Bulk / não separável (proporcional)

```
bulk_batches/{batchId}

{
  "tenantId": "TENANT_HOST",
  "batchId": "bulk_2026_01",
  "assetType": "BULK_GOODS",
  "propertyId": "prop_123",
  "description": "Milho a granel - silo 1",
  "unit": "KG",
  "totalQty": 100000,

  "ownershipShares": [
    { "ownerTenantId": "TENANT_A", "percent": 40, "basis": "NF",
"sourceRef": { "nfKey": "..." } },
    { "ownerTenantId": "TENANT_B", "percent": 60, "basis": "NF",
"sourceRef": { "nfKey": "..." } }
  ],

  "locks": { "requiresSplit": true, "blockNF": true },
  "createdAt": 1730000000000
}
```

Ao vender/transferir "x kg" desse batch:

- calcula rateio proporcional pelas `ownershipShares`
- gera split e trava NF até escrow/split pronto

---

3) Serviço: Owner Resolver (resolve "quem recebe")

Ele recebe um conjunto de itens/ativos com `identityRef` e devolve `ownershipShares`.

3.1 Interface

```
// src/ownership/ownership.types.ts
```

```
export type AssetLine = {
  kind: 'ANIMAL'|'UNIT'|'BULK';
  qty: number; // para animal qty = 1
  identityRef: any; // IdentityRef
  productId?: string;
  batchId?: string;
};

export type OwnershipResolution = {
  line: AssetLine;
  shares: { ownerTenantId: string; percent: number; basis: string;
sourceRef?: any }[];
  mode: 'INDIVIDUAL'|'PROPORTIONAL';
};
```

3.2 Implementação

```
// src/ownership/ownership-resolver.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { AssetLine, OwnershipResolution } from './ownership.types';

@Injectable()
export class OwnershipResolverService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async resolve(tenantId: string, lines: AssetLine[]):
Promise<OwnershipResolution[]> {
    const out: OwnershipResolution[] = [];

    for (const line of lines) {
      if (line.kind === 'ANIMAL') {
        // resolve por animals/{animalId} usando identityRef.value
        const q = await this.db.collection('animals')
          .where('tenantId', '==', tenantId)
          .where('identity.value', '==', line.identityRef.value)
          .limit(1).get();

        if (q.empty) throw new Error('ANIMAL_NOT_FOUND');

        const a = q.docs[0].data();
        out.push({
          line,
          mode: 'INDIVIDUAL',
          shares: [{ ownerTenantId: a.ownership.ownerTenantId,
percent: 100, basis: 'IDENTITY', sourceRef: { animalId: a.animalId }
}]
        });
        continue;
      }

      if (line.kind === 'UNIT') {
        // resolve unit
        const q = await this.db.collection('inventory_units')
          .where('tenantId', '==', tenantId)
          .where('identityRef.value', '==', line.identityRef.value)
          .limit(1).get();

        if (q.empty) throw new Error('UNIT_NOT_FOUND');

        const u = q.docs[0].data();
```

```
        const shares = u.ownerShare ?? [{ ownerTenantId:
u.supplierTenantId, percent: 100, basis: 'NF' }];
        out.push({ line, mode: 'INDIVIDUAL', shares });
        continue;
      }

      if (line.kind === 'BULK') {
        // resolve batch
        const ref =
this.db.collection('bulk_batches').doc(line.batchId!);
        const snap = await ref.get();
        if (!snap.exists) throw new Error('BULK_NOT_FOUND');

        const b = snap.data()!;
        out.push({
          line,
          mode: 'PROPORTIONAL',
          shares: (b.ownershipShares ?? []).map((s: any) => ({
            ownerTenantId: s.ownerTenantId,
            percent: s.percent,
            basis: s.basis,
            sourceRef: s.sourceRef ?? { batchId: b.batchId }
          }))
        });
        continue;
      }

      throw new Error('LINE_KIND_INVALID');
    }

    return out;
  }
}
```

---

4) Serviço: Split Plan Generator (gera split por linha e total)

Entrada: linhas + preço total/por item
Saída: `splitPlan` com destinatários e valores.

4.1 Tipos

```
// src/settlement/split.types.ts
export type SplitTarget = {
  targetTenantId: string;
  amount: number; // centavos
  reason: string;
  refs?: any;
};

export type SplitPlan = {
  currency: 'BRL';
  targets: SplitTarget[];
  totals: { gross: number; net: number };
};
```

4.2 Geração (proporcional por fornecedor/dono)

```
// src/settlement/split-plan.service.ts
import { Injectable } from '@nestjs/common';
import { OwnershipResolution } from '../ownership/ownership.types';
```

```
import { SplitPlan } from './split.types';

function roundCents(n: number) { return Math.round(n); }

@Injectable()
export class SplitPlanService {
  buildSplit(resolutions: OwnershipResolution[], lineTotalsCents:
number[]): SplitPlan {
    const targetsMap = new Map<string, number>();

    resolutions.forEach((r, idx) => {
      const lineTotal = lineTotalsCents[idx];

      r.shares.forEach((s) => {
        const portion = (lineTotal * s.percent) / 100;
        const prev = targetsMap.get(s.ownerTenantId) || 0;
        targetsMap.set(s.ownerTenantId, prev + portion);
      });
    });

    const targets = Array.from(targetsMap.entries()).map(([tenantId,
amount]) => ({
      targetTenantId: tenantId,
      amount: roundCents(amount),
      reason: 'AUTO_SPLIT_OWNERSHIP'
    }));

    const gross = lineTotalsCents.reduce((a,b)=>a+b,0);
    const allocated = targets.reduce((a,t)=>a+t.amount,0);
    const diff = gross - allocated;

    // ajusta diferença de arredondamento no primeiro
    if (targets.length && diff !== 0) targets[0].amount += diff;

    return {
      currency: 'BRL',
      targets,
      totals: { gross, net: gross }
    };
  }
}
```

---

5) Aplicação no PDV / Marketplace

5.1 PDV sale payload (exemplo)

```
{
  "propertyId": "prop_loj_1",
  "items": [
    {
      "kind": "UNIT",
      "qty": 1,
      "identityRef": { "kind": "SERIAL", "value": "SER123" },
      "unitPriceCents": 25000
    },
    {
      "kind": "UNIT",
      "qty": 1,
      "identityRef": { "kind": "NFE_ITEM", "nfKey": "44DIG...",
"itemSeq": 2 },
```

```
      "unitPriceCents": 12000
    }
  ]
}
```

O backend:

1. resolve owners (`OwnershipResolverService`)
2. gera split (`SplitPlanService`)
3. exige escrow antes de confirmar venda
4. no pagamento, executa split automático

5.2 Venda de bovino

```
{
  "propertyId": "prop_123",
  "items": [
    { "kind": "ANIMAL", "qty": 1, "identityRef": { "kind": "BRINCO",
"value": "BR123" }, "unitPriceCents": 800000 }
  ]
}
```

5.3 Venda bulk (milho a granel)

```
{
  "propertyId": "prop_123",
  "items": [
    { "kind": "BULK", "batchId": "bulk_2026_01", "qty": 5000,
"identityRef": { "kind": "BULK_BATCH", "value": "bulk_2026_01" },
"unitPriceCents": 120 }
  ]
}
```

---

6) Regras (RulesEngine) para “mistura / split obrigatório”

Regra: se houver BULK ou múltiplos donos → split obrigatório

```
// src/rules/rules/ownershipSplitRequired.rule.ts
import { Rule, RuleContext, RuleResult } from '../rules.types';

export class OwnershipSplitRequiredRule implements Rule {
  id = 'ownershipSplitRequired';

  applies(ctx: RuleContext) {
    return ctx.operation === 'PDV_SALE' || ctx.operation ===
'TRANSFER_ASSET' || ctx.operation === 'NF_EMIT';
  }

  async evaluate(ctx: RuleContext): Promise<RuleResult> {
    const splitPlan = ctx.payload?.splitPlan;
    const ownership = ctx.payload?.ownershipSummary;

    if (!ownership) return { ok: true };

    const owners = new Set(ownership.owners ?? []);
    const hasBulk = ownership.hasBulk === true;

    if ((owners.size > 1 || hasBulk) && splitPlan?.enabled !== true) {
```

```
      return {
        ok: false,
        code: 'OWNERSHIP_SPLIT_REQUIRED',
        message: 'Operação exige split/escrow por múltiplos donos ou
lote não separável.',
        details: { owners: Array.from(owners), hasBulk }
      };
    }
    return { ok: true };
  }
}
```

---

7) “Filho segue RG da mãe” (processo técnico)

No serviço de cadastro de nascimento (ex.: `POST /api/animals/birth`):

- exige `motherIdentity`
- busca mãe
- cria filho com:
  - `ownership.ownerTenantId = mother.ownership.ownerTenantId`
  - `mother` preenchido
- registra evidência Tipo B (foto + GPS) quando crítico

Isso garante seu requisito: **“se for filho deve seguir padrão RG da mãe”**.

---

8) Travar NF / mover ativo até liquidar split

Fluxo de segurança:

1. tentativa de NF / transferência / venda → RulesEngine:
   - se custódia/ownership exigir split → bloqueia
2. sistema cria “solicitação”:
   - `NF_EMISSION_REQUEST` ou `TRANSFER_REQUEST`
3. usuário paga escrow → status `HELD`
4. executa split (ou agenda por marcos)
5. só então:
   - libera NF
   - libera transferência
   - registra evento `FISCAL_DOC_VERIFIED` etc.

Quer que eu entregue o endpoint completo `POST /api/pdv/sales` (com resolve owner + splitPlan + escrow + auditoria) e o endpoint `POST /api/nf/request` com travas e estado?

Perfeito — abaixo estão os **2 endpoints completos** com o fluxo inteiro:

1. `POST /api/pdv/sales` → resolve donos → gera split → cria escrow (Asaas) → trava/retém → retorna link de pagamento
2. `POST /api/nf/request` → valida regras (CAR/IE + custódia + split/escrow) → cria solicitação NF com estado `PENDING` ou `BLOCKED`

Incluí:

- multi-tenant (`tenantId`)
- RBAC/permissões
- RulesEngine
- OwnerResolver + SplitPlan
- idempotência (`clientRequestId`)
- auditoria append-only (`audit_events`)
- estados claros pra tela

A) POST /api/pdv/sales

A1) DTO

```
// src/pdv/dto/create-sale.dto.ts
import { IsArray, IsNotEmpty, IsOptional, IsString, ValidateNested }
from 'class-validator';
import { Type } from 'class-transformer';

class SaleItemDto {
  @IsString() @IsNotEmpty()
  kind: 'UNIT'|'ANIMAL'|'BULK';

  @IsNotEmpty()
  identityRef: any;

  @IsOptional() @IsString()
  batchId?: string;

  @IsNotEmpty()
  qty: number;

  @IsNotEmpty()
  unitPriceCents: number;
}

export class CreatePdvSaleDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsString() @IsNotEmpty()
  propertyId: string;

  @IsArray()
  @ValidateNested({ each: true })
  @Type(() => SaleItemDto)
  items: SaleItemDto[];

  @IsOptional()
  note?: string;
}
```

A2) Controller

```
// src/pdv/pdv.controller.ts
import { Body, Controller, Post, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { PdvService } from './pdv.service';
import { CreatePdvSaleDto } from './dto/create-sale.dto';

@Controller('/api/pdv')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class PdvController {
  constructor(private readonly svc: PdvService) {}

  @Post('/sales')
  @Roles('ADMIN','MANAGER','OPERATOR','PRODUCER')
  @Permissions('pdv.sale.create')
  async createSale(@Actor() actor: any, @Body() dto: CreatePdvSaleDto)
{
    return this.svc.createSale(actor, dto);
  }
}
```

A3) Service (resolve owners + split + rules + escrow + audit + idempotência)

Observação: o "escrow Asaas" eu deixei via `AsaasService.createCharge(...)` retornando `chargeId` + `invoiceUrl`. Você já tem integração.

```
// src/pdv/pdv.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { RulesEngine } from '../rules/rules.engine';
import { OwnershipResolverService } from '../ownership/ownership-
resolver.service';
import { SplitPlanService } from '../settlement/split-plan.service';
import { AuditWriter } from '../audit/audit-writer.service';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';
import { AsaasService } from '../integrations/asaas/asaas.service';

@Injectable()
export class PdvService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly rules: RulesEngine,
    private readonly owners: OwnershipResolverService,
    private readonly split: SplitPlanService,
    private readonly audit: AuditWriter,
    private readonly asaas: AsaasService,
  ) {}
```

```
  async createSale(actor: any, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'pdv.sale.create');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      // 1) montar linhas para resolver ownership
      const lines = dto.items.map((it: any) => ({
        kind: it.kind,
        qty: Number(it.qty),
        identityRef: it.identityRef,
        batchId: it.batchId ?? null,
      }));

      // 2) resolve donos (fora do tx é ok, mas aqui mantive simples;
em prod você pode resolver antes do tx)
      // Como Firestore tx não aceita await externo com tx.get?
aceita, mas cuidado com performance.
      const ownershipRes = await this.owners.resolve(tenantId, lines);

      const ownersSet = new Set<string>();
      let hasBulk = false;
      ownershipRes.forEach((r) => {
        r.shares.forEach(s => ownersSet.add(s.ownerTenantId));
        if (r.mode === 'PROPORTIONAL') hasBulk = true;
      });

      // 3) gerar split plan (por linha)
      const lineTotals = dto.items.map((it: any) => Number(it.qty) *
Number(it.unitPriceCents));
      const splitPlan = this.split.buildSplit(ownershipRes,
lineTotals);

      const gross = splitPlan.totals.gross;

      // 4) rodar RulesEngine (inclui CAR/IE + custódia + split
obrigatório + identidade)
      const ctx = {
        actor,
        tenantId,
        operation: 'PDV_SALE',
        propertyId: dto.propertyId,
        payload: {
          items: dto.items.map((it: any, idx: number) => ({
            ...it,
            identityRef: it.identityRef,
            // resumo p/ regra antiga (fornecedor/identidade)
            supplierTenantId:
ownershipRes[idx]?.shares?.[0]?.ownerTenantId ?? null,
          })),
          ownershipSummary: { owners: Array.from(ownersSet), hasBulk
},
          splitPlan: { enabled: true, ...splitPlan }, // aqui já está
"habilitado"
```

```
        }
      };

      const ruleRes = await this.rules.run(ctx);
      if (!ruleRes.ok) {
        await this.audit.block(actor, {
          operation: 'PDV_SALE',
          propertyId: dto.propertyId,
          violations: ruleRes.violations
        });
        throw new Error(`BLOCKED_${ruleRes.violations[0].code}`);
      }

      // 5) criar sale e reservation (escrow) — estado PENDING_PAYMENT
      const saleId = makeId('sale');
      const saleRef = this.db.collection('pdv_sales').doc(saleId);

      tx.create(saleRef, {
        saleId,
        tenantId,
        propertyId: dto.propertyId,
        status: 'PENDING_PAYMENT',
        items: dto.items,
        ownership: ownershipRes,
        splitPlan,
        amountCents: gross,
        note: dto.note ?? null,
        createdAt: now,
        createdBy: { uid: actor.uid, role: actor.role },
      });

      // 6) criar cobrança escrow no Asaas (fora do tx idealmente)
      // Para manter consistência no exemplo: criamos uma
"chargeRequest" e um worker faz Asaas.
      // Se você quer "imediato", faça o AsaasService.createCharge
antes do tx e grave o retorno.
      const chargeReqId = makeId('chgreq');
      const chargeReqRef =
this.db.collection('payment_charge_requests').doc(chargeReqId);

      tx.create(chargeReqRef, {
        chargeReqId,
        tenantId,
        saleId,
        provider: 'ASAAS',
        amountCents: gross,
        currency: 'BRL',
        status: 'REQUESTED',
        createdAt: now,
      });

      // 7) auditoria append-only
      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        propertyId: dto.propertyId,
        type: 'PDV_SALE_CREATED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { saleId, amountCents: gross, owners:
Array.from(ownersSet), hasBulk },
```

```
        ts: now
      });

      const out = {
        saleId,
        status: 'PENDING_PAYMENT',
        amountCents: gross,
        splitTargets: splitPlan.targets,
        charge: { status: 'REQUESTED', chargeReqId }
      };

      tx.create(idemRef, {
        tenantId,
        uid: actor.uid,
        scope: 'pdv.sale.create',
        clientRequestId: dto.clientRequestId,
        createdAt: now,
        result: out
      });

      return out;
    });
  }

  // Se você preferir "charge imediato":
  // 1) criaCharge fora do tx com idempotência no AsaasService
  // 2) grava chargeId+invoiceUrl no sale
}
```

Como fica o "pagamento confirmado"?

Você recebe webhook Asaas:

- marca `pdv_sales.status = HELD` (ou `PAID`)
- dispara "split executor" para efetivar split
- só então `status = DONE`
- registra `SETTLEMENT_TRANCHE_RELEASED` / `SPLIT_EXECUTED` em auditoria

(Se quiser eu mando esse webhook handler também.)

B) POST /api/nf/request

Esse endpoint **não emite a NF**: ele cria uma **solicitação** que pode ficar:

- `PENDING` (apta) → segue para emissão
- `BLOCKED` (travada) → exibe motivo + ação corretiva

B1) DTO

```
// src/fiscal/dto/nf-request.dto.ts
import { IsNotEmpty, IsOptional, IsString } from 'class-validator';

export class NfRequestDto {
  @IsString() @IsNotEmpty()
```

```
  clientRequestId: string;

  @IsString() @IsNotEmpty()
  propertyId: string;

  @IsNotEmpty()
  payload: any; // itens, destinatário, etc.

  @IsOptional() @IsString()
  note?: string;
}
```

B2) Controller

```
// src/fiscal/fiscal.controller.ts
import { Body, Controller, Post, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { FiscalService } from './fiscal.service';
import { NfRequestDto } from './dto/nf-request.dto';

@Controller('/api/nf')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class FiscalController {
  constructor(private readonly svc: FiscalService) {}

  @Post('/request')
  @Roles('ADMIN','MANAGER','OPERATOR','PRODUCER')
  @Permissions('nf.request')
  async request(@Actor() actor: any, @Body() dto: NfRequestDto) {
    return this.svc.requestNf(actor, dto);
  }
}
```

B3) Service (RulesEngine + cria `fiscal_issues` com state)

```
// src/fiscal/fiscal.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { RulesEngine } from '../rules/rules.engine';
import { AuditWriter } from '../audit/audit-writer.service';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class FiscalService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly rules: RulesEngine,
    private readonly audit: AuditWriter,
  ) {}

  async requestNf(actor: any, dto: any) {
```

```
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'nf.request');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const issueId = makeId('nfi');
      const issueRef =
this.db.collection('fiscal_issues').doc(issueId);

      // roda regras
      const ctx = {
        actor,
        tenantId,
        operation: 'NF_EMIT',
        propertyId: dto.propertyId,
        payload: dto.payload
      };

      const ruleRes = await this.rules.run(ctx);

      if (!ruleRes.ok) {
        // registra solicitação já bloqueada
        tx.create(issueRef, {
          tenantId,
          issueId,
          propertyId: dto.propertyId,
          type: 'NF_EMISSION_REQUEST',
          payload: dto.payload,
          note: dto.note ?? null,
          state: 'BLOCKED',
          blockReason: ruleRes.violations,
          createdAt: now,
          createdBy: { uid: actor.uid, role: actor.role }
        });

        // auditoria de bloqueio
        const auditId = makeId('aud');
        tx.create(this.db.collection('audit_events').doc(auditId), {
          auditId,
          tenantId,
          propertyId: dto.propertyId,
          type: 'RULE_BLOCKED',
          actor: { uid: actor.uid, role: actor.role },
          meta: { operation: 'NF_EMIT', issueId, violations:
ruleRes.violations },
          ts: now
        });

        const out = {
          issueId,
          state: 'BLOCKED',
          violations: ruleRes.violations
        };

        tx.create(idemRef, {
```

```
        tenantId,
        uid: actor.uid,
        scope: 'nf.request',
        clientRequestId: dto.clientRequestId,
        createdAt: now,
        result: out
      });

      return out;
    }

    // apta
    tx.create(issueRef, {
      tenantId,
      issueId,
      propertyId: dto.propertyId,
      type: 'NF_EMISSION_REQUEST',
      payload: dto.payload,
      note: dto.note ?? null,
      state: 'PENDING',
      blockReason: null,
      createdAt: now,
      createdBy: { uid: actor.uid, role: actor.role }
    });

    const auditId = makeId('aud');
    tx.create(this.db.collection('audit_events').doc(auditId), {
      auditId,
      tenantId,
      propertyId: dto.propertyId,
      type: 'NF_REQUEST_CREATED',
      actor: { uid: actor.uid, role: actor.role },
      meta: { issueId },
      ts: now
    });

    const out = { issueId, state: 'PENDING' };

    tx.create(idemRef, {
      tenantId,
      uid: actor.uid,
      scope: 'nf.request',
      clientRequestId: dto.clientRequestId,
      createdAt: now,
      result: out
    });

    return out;
  });
  }
}
```

C) Módulos/Providers necessários

C1) Permissões mínimas

- `pdv.sale.create`
- `nf.request`

C2) Coleções criadas

- `pdv_sales`
- `payment_charge_requests`
- `fiscal_issues`
- `audit_events`
- `idempotency`

D) Como o app usa isso (fluxo simples)

PDV

1. `POST /api/pdv/sales` → recebe `saleId` + `chargeReqId`
2. App mostra "Aguardando pagamento"
3. App consulta `GET /api/pdv/sales/:id/status` (se quiser eu mando)
4. Quando webhook Asaas confirmar: `PAID/HELD`
5. App libera "Emitir NF" (se aplicável) e "Entregar"

NF

1. `POST /api/nf/request`

- se `BLOCKED`: mostrar motivos e ação corretiva
- se `PENDING`: seguir para emissão / integração fiscal

Quer que eu entregue agora também:

- `GET /api/pdv/sales/:id/status` (leve pra polling)
- handler de **webhook Asaas** que muda `pdv_sales` para `PAID/HELD` e dispara split
- executor de split (escrow → fornecedores) com auditoria `SPLIT_EXECUTED`

Perfeito — abaixo estão **os 3 blocos completos**:

1. `GET /api/pdv/sales/:id/status` (leve, ideal para polling)
2. Webhook **Asaas**: marca venda como `PAID/HELD`, gera evento e dispara split
3. **Split Executor**: calcula e registra `SPLIT_EXECUTED` + (opcional) chama Asaas split/transfer

Observação prática: no Asaas, "split" pode variar conforme o produto (split de cobrança / transferências). Eu deixei o executor preparado para 2 modos:

- **MODE_A:** registrar split internamente + liberar por marcos (sempre funciona)
- **MODE_B:** integrar split/transfer via AsaasService (você ajusta conforme seu plano/produto)

1) GET /api/pdv/sales/:id/status

Controller

```
// src/pdv/pdv-status.controller.ts
import { Controller, Get, Param, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { PdvStatusService } from './pdv-status.service';

@Controller('/api/pdv/sales')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class PdvStatusController {
  constructor(private readonly svc: PdvStatusService) {}

  @Get('/:id/status')
  @Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','AUDITOR')
  @Permissions('pdv.sale.read')
  async status(@Actor() actor: any, @Param('id') saleId: string) {
    return this.svc.getStatus(actor, saleId);
  }
}
```

Service

```
// src/pdv/pdv-status.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

@Injectable()
export class PdvStatusService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async getStatus(actor: any, saleId: string) {
    const ref = this.db.collection('pdv_sales').doc(saleId);
    const snap = await ref.get();
    if (!snap.exists) throw new Error('SALE_NOT_FOUND');

    const s = snap.data()!;
    if (s.tenantId !== actor.tenantId) throw new
Error('SALE_FORBIDDEN');

    return {
      saleId,
      status: s.status,
      amountCents: s.amountCents,
      updatedAt: s.updatedAt ?? s.createdAt,
      payment: {
        provider: s.payment?.provider ?? null,
        chargeId: s.payment?.chargeId ?? null,
        invoiceUrl: s.payment?.invoiceUrl ?? null,
        status: s.payment?.status ?? null,
        paidAt: s.payment?.paidAt ?? null,
      },
```

```
      settlement: {
        status: s.settlement?.status ?? null, // PENDING|EXECUTED
        executedAt: s.settlement?.executedAt ?? null,
      }
    };
  }
}
```

---

2) Webhook Asaas → atualiza `pdv_sales` e dispara split

2.1 Controller webhook (sem auth, com validação de token)

```
// src/integrations/asaas/asaas-webhook.controller.ts
import { Body, Controller, Headers, Post } from '@nestjs/common';
import { AsaasWebhookService } from './asaas-webhook.service';

@Controller('/api/webhooks/asaas')
export class AsaasWebhookController {
  constructor(private readonly svc: AsaasWebhookService) {}

  @Post()
  async handle(@Headers('asaas-access-token') token: string, @Body()
payload: any) {
    // você pode trocar por assinatura HMAC também
    const expected = process.env.ASAAS_WEBHOOK_TOKEN;
    if (expected && token !== expected) throw new
Error('WEBHOOK_FORBIDDEN');

    return this.svc.process(payload);
  }
}
```

2.2 Service webhook

```
// src/integrations/asaas/asaas-webhook.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../../common/firestore.provider';
import { makeId } from '../../common/utils/id.util';
import { SplitExecutorService } from '../../settlement/split-
executor.service';

@Injectable()
export class AsaasWebhookService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly splitExec: SplitExecutorService
  ) {}

  async process(payload: any) {
    // Asaas costuma enviar payload com event + payment
    const event = payload?.event;
    const payment = payload?.payment;
    if (!event || !payment?.id) return { ok: true };

    // exemplo: PAYMENT_CONFIRMED / PAYMENT_RECEIVED
    const paid =
      event === 'PAYMENT_CONFIRMED' ||
      event === 'PAYMENT_RECEIVED' ||
      payment?.status === 'RECEIVED' ||
      payment?.status === 'CONFIRMED';
```

```
    if (!paid) return { ok: true };

    // localizar chargeReq ou sale pelo chargeId
    // Recomendação: quando criar charge no Asaas, salvar chargeId em
payment_charge_requests e em pdv_sales.payment.chargeId
    const q = await this.db.collection('payment_charge_requests')
      .where('provider', '==', 'ASAAS')
      .where('chargeId', '==', payment.id)
      .limit(1).get();

    if (q.empty) return { ok: true };

    const chargeReq = q.docs[0].data();
    const saleId = chargeReq.saleId;
    const tenantId = chargeReq.tenantId;

    // atualizar status e disparar split em sequência
    await this.db.runTransaction(async (tx) => {
      const saleRef = this.db.collection('pdv_sales').doc(saleId);
      const saleSnap = await tx.get(saleRef);
      if (!saleSnap.exists) throw new Error('SALE_NOT_FOUND');
      const s = saleSnap.data()!;

      // idempotência do webhook: se já está PAID/HELD, não refaz
      if (s.status === 'HELD' || s.status === 'PAID' || s.status ===
'DONE') return;

      tx.set(saleRef, {
        status: 'HELD',
        payment: {
          provider: 'ASAAS',
          chargeId: payment.id,
          invoiceUrl: payment.invoiceUrl ?? s.payment?.invoiceUrl ??
null,
          status: payment.status ?? 'RECEIVED',
          paidAt: Date.now()
        },
        updatedAt: Date.now(),
        settlement: { status: 'PENDING' }
      }, { merge: true });

      // auditoria append-only
      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        type: 'PDV_PAYMENT_CONFIRMED',
        actor: { uid: 'SYSTEM', role: 'WEBHOOK' },
        meta: { saleId, chargeId: payment.id, status: payment.status
},
        ts: Date.now()
      });
    });

    // Split executor fora da transação (pode chamar APIs externas)
    await this.splitExec.executeForSale(tenantId, saleId);

    return { ok: true };
  }
}
```

3) Split Executor (registra e opcionalmente faz transferências)

3.1 Service

```
// src/settlement/split-executor.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';
import { AsaasService } from '../integrations/asaas/asaas.service';

@Injectable()
export class SplitExecutorService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly asaas: AsaasService,
  ) {}

  async executeForSale(tenantId: string, saleId: string) {
    const saleRef = this.db.collection('pdv_sales').doc(saleId);
    const snap = await saleRef.get();
    if (!snap.exists) throw new Error('SALE_NOT_FOUND');

    const s = snap.data()!;
    if (s.tenantId !== tenantId) throw new
Error('SALE_TENANT_MISMATCH');

    // idempotência: se já executou, sai
    if (s.settlement?.status === 'EXECUTED') return { ok: true,
already: true };

    // splitPlan.targets: [{targetTenantId, amount}]
    const targets = s.splitPlan?.targets ?? [];
    if (!targets.length) {
      // se não há split, marca como executado mesmo assim
      await saleRef.set({ settlement: { status: 'EXECUTED',
executedAt: Date.now() }, status: 'DONE', updatedAt: Date.now() }, {
merge: true });
      return { ok: true, noTargets: true };
    }

    // MODE_A: registrar internamente e liberar por marcos
    // MODE_B: efetuar transferências reais via Asaas (ajuste conforme
seu produto)
    const mode = process.env.SPLIT_MODE || 'MODE_A';

    if (mode === 'MODE_B') {
      // Exemplo: criar transferências
      // Você precisa mapear targetTenantId -> wallet/account no Asaas
      for (const t of targets) {
        const payoutAccount = await
this.getAsaasAccountForTenant(t.targetTenantId);
        await this.asaas.createTransfer({
          amountCents: t.amount,
          toAsaasAccountId: payoutAccount,
          description: `Split PDV Sale ${saleId}`
        });
      }
    }

    // registra execução (append-only) + marca sale DONE
    await this.db.runTransaction(async (tx) => {
```

```
      const fresh = await tx.get(saleRef);
      if (!fresh.exists) throw new Error('SALE_NOT_FOUND');
      const cur = fresh.data()!;

      if (cur.settlement?.status === 'EXECUTED') return;

      tx.set(saleRef, {
        settlement: { status: 'EXECUTED', executedAt: Date.now(), mode
},
        status: 'DONE',
        updatedAt: Date.now()
      }, { merge: true });

      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        type: 'SPLIT_EXECUTED',
        actor: { uid: 'SYSTEM', role: 'SETTLEMENT' },
        meta: { saleId, mode, targets },
        ts: Date.now()
      });
    });

    return { ok: true, mode, targetsCount: targets.length };
  }

  // Exemplo de mapeamento tenant -> conta Asaas (você pode manter em
tenants/{id}.asaasAccountId)
  private async getAsaasAccountForTenant(targetTenantId: string):
Promise<string> {
    const snap = await
this.db.collection('tenants').doc(targetTenantId).get();
    const t = snap.exists ? snap.data() : null;
    const id = t?.asaasAccountId;
    if (!id) throw new Error('ASAAS_ACCOUNT_NOT_LINKED');
    return id;
  }
}
```

3.2 AsaasService stub (transfer)

```
// src/integrations/asaas/asaas.service.ts
import { Injectable } from '@nestjs/common';

@Injectable()
export class AsaasService {
  // ... seu createCharge já existe / será implementado

  async createTransfer(input: { amountCents: number; toAsaasAccountId:
string; description: string }) {
    // implementar chamada real ao Asaas conforme documentação e
credenciais
    // aqui é stub
    return { ok: true, transferId: `tr_${Date.now()}` };
  }
}
```

---

4) Permissões mínimas

- `pdv.sale.read`
- `pdv.sale.create`
- `audit.read` (para auditor)
- webhook não usa permissão, usa token

---

5) Como o app usa (fluxo exato)

1. `POST /api/pdv/sales` → recebe `saleId` + `chargeReqId`
2. App mostra "Aguardando pagamento"
3. App faz polling: `GET /api/pdv/sales/:id/status`
4. Quando status virar `HELD` → mostra "Pago. Processando split..."
5. Depois vira `DONE` → libera emissão NF / entrega / baixa estoque

---

6) Ajuste recomendável (charge imediato vs assíncrono)

Hoje deixei a cobrança como "request" para worker/webhook preencher `chargeId`.
Se você quer **link de pagamento na hora**:

- faça `asaas.createCharge(...)` antes do `tx`
- grave `chargeId + invoiceUrl` direto na sale e no charge_request

Se quiser, eu mando a versão "imediata" também.

Quer que eu feche agora:

- `POST /api/pdv/sales/:id/cancel` com multa + regras
- `POST /api/pdv/sales/:id/refund` (quando permitido)
- e o módulo "baixa estoque + evidência por foto/voz" acoplado ao PDV?

Entendido — vamos ajustar para ficar **perfeito no agro**:

- **Estoque interno da fazenda**: movimentação operacional (entrada/saída/consumo/aplicação), com evidência por foto/voz, offline, auditoria e regras ambientais/contratuais.
- **Estoque de loja**: é **venda** (PDV) ou "código" (catálogo/identidade) com split/escrow e rastreio de origem (NF/serial/lote).

Vou te entregar agora:

1. Separação de módulos (FarmInventory vs StoreRetail)

2. Endpoints de **baixa de estoque da fazenda** + evidência (foto/voz/planilha manual)
3. Regras: bloqueios por CAR/IE + limites ambientais + contrato + custódia
4. Como "foto de anotação manual" vira lançamento sem digitação (pipeline OCR/IA opcional, mas com fallback humano)

---

1) Módulos separados

1.1 FarmInventory (fazenda)

Coleções:

- `farm_items` (insumos internos: sal mineral, vacina, ração, fertilizante…)
- `farm_stock_balance` (saldo por propriedade/local)
- `farm_stock_moves` (append-only)
- `farm_applications` (aplicação em cultura/lote/animais)
- `evidence` (foto/voz/documento)
- `projects/contracts` (amarra travas e marcos)

Operações típicas:

- entrada (compra/recebimento)
- transferência entre depósitos
- consumo (baixa)
- aplicação (com evidência e alvo: talhão/lote/animal)

✅ Aqui normalmente **não existe split por fornecedor** em cada saída; o split e fiscal entram quando há venda/contrato/consignação, não na operação interna.

1.2 StoreRetail (loja)

Coleções:

- `catalog_items`, `inventory_units`, `pdv_sales`
- split/escrow obrigatório em multi-fornecedor
- rastreio por NF/serial/lote

✅ Aqui é venda, então split/escrow faz sentido sempre que houver consignação/múltiplos fornecedores.

---

2) Endpoints: estoque interno da fazenda (baixa + evidência)

2.1 `POST /api/farm/stock/movements` (movimentação interna)

DTO

```
// src/farm/dto/farm-movement.dto.ts
import { IsNotEmpty, IsOptional, IsString } from 'class-validator';

export class FarmStockMovementDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsString() @IsNotEmpty()
  propertyId: string;

  @IsString() @IsNotEmpty()
  itemId: string;

  @IsString() @IsNotEmpty()
  movementType: 'IN'|'OUT'|'TRANSFER';

  @IsNotEmpty()
  qty: number;

  @IsOptional() @IsString()
  fromLocationId?: string;

  @IsOptional() @IsString()
  toLocationId?: string;

  // opcional: aplicação vinculada
  @IsOptional()
  application?: {
    targetType: 'PASTURE'|'FIELD'|'ANIMAL_LOT'|'ANIMAL';
    targetId: string;
  };

  @IsOptional() @IsString()
  note?: string;

  // evidência (pode ser enviada depois via offline queue)
  @IsOptional()
  evidence?: {
    required: boolean;
    evidenceId?: string;     // se já comitada
    storagePath?: string;    // se já subiu e vai comitar depois
    riskType?: 'A'|'B';
  };
}
```

Controller

```
// src/farm/farm-stock.controller.ts
import { Body, Controller, Post, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { FarmStockService } from './farm-stock.service';
import { FarmStockMovementDto } from './dto/farm-movement.dto';
```

```
@Controller('/api/farm/stock')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class FarmStockController {
  constructor(private readonly svc: FarmStockService) {}

  @Post('/movements')
  @Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN')
  @Permissions('farm.stock.move')
  async move(@Actor() actor: any, @Body() dto: FarmStockMovementDto) {
    return this.svc.createMovement(actor, dto);
  }
}
```

Service (append-only + saldo + regras)

```
// src/farm/farm-stock.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';
import { RulesEngine } from '../rules/rules.engine';

@Injectable()
export class FarmStockService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly rules: RulesEngine,
  ) {}

  async createMovement(actor: any, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'farm.stock.move');
    const idemRef = this.db.collection('idempotency').doc(key);

    // regras: CAR/IE + limites ambientais + contrato (se a aplicação
exigir)
    const ctx = {
      actor,
      tenantId,
      operation: 'FARM_STOCK_MOVE',
      propertyId: dto.propertyId,
      payload: dto
    };
    const ruleRes = await this.rules.run(ctx);
    if (!ruleRes.ok) throw new
Error(`BLOCKED_${ruleRes.violations[0].code}`);

    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const moveId = makeId('fsm');
      const moveRef =
this.db.collection('farm_stock_moves').doc(moveId);

      // saldo atual
```

```
      const balKey =
`${dto.propertyId}_${dto.itemId}_${dto.fromLocationId || 'default'}`;
      const balRef =
this.db.collection('farm_stock_balance').doc(`${tenantId}_${balKey}`);
      const balSnap = await tx.get(balRef);
      const current = balSnap.exists ? (balSnap.data()!.qty || 0) : 0;

      const qty = Number(dto.qty);
      let next = current;

      if (dto.movementType === 'IN') next = current + qty;
      if (dto.movementType === 'OUT') next = current - qty;
      if (dto.movementType === 'TRANSFER') next = current - qty;

      if (next < 0) throw new Error('INSUFFICIENT_STOCK');

      tx.set(balRef, {
        tenantId,
        propertyId: dto.propertyId,
        itemId: dto.itemId,
        locationId: dto.fromLocationId || 'default',
        qty: next,
        updatedAt: now
      }, { merge: true });

      // se transfer, soma no destino
      if (dto.movementType === 'TRANSFER') {
        const toKey =
`${dto.propertyId}_${dto.itemId}_${dto.toLocationId || 'default'}`;
        const toRef =
this.db.collection('farm_stock_balance').doc(`${tenantId}_${toKey}`);
        const toSnap = await tx.get(toRef);
        const toCur = toSnap.exists ? (toSnap.data()!.qty || 0) : 0;
        tx.set(toRef, {
          tenantId,
          propertyId: dto.propertyId,
          itemId: dto.itemId,
          locationId: dto.toLocationId || 'default',
          qty: toCur + qty,
          updatedAt: now
        }, { merge: true });
      }

      // append-only do movimento (imutável)
      tx.create(moveRef, {
        moveId,
        tenantId,
        propertyId: dto.propertyId,
        itemId: dto.itemId,
        movementType: dto.movementType,
        qty,
        fromLocationId: dto.fromLocationId ?? null,
        toLocationId: dto.toLocationId ?? null,
        balanceBefore: current,
        balanceAfter: next,
        application: dto.application ?? null,
        note: dto.note ?? null,
        evidence: dto.evidence ?? null,
        performedBy: { uid: actor.uid, role: actor.role },
        performedAt: now
      });
```

```
      // auditoria
      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        propertyId: dto.propertyId,
        type: 'FARM_STOCK_MOVED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { moveId, itemId: dto.itemId, movementType:
dto.movementType, qty },
        ts: now
      });

      const out = { moveId, balanceAfter: next };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'farm.stock.move', clientRequestId: dto.clientRequestId, createdAt:
now, result: out });

      return out;
    });
  }
}
```

---

3) “Foto de anotação manual” → lançamento sem digitar

Aqui é o jeito mais seguro e compatível:

3.1 Fluxo no app

1. operador retira insumo e **anota na planilha de papel**
2. tira **foto** (evidência Tipo B ou DOC)
3. app cria “Rascunho de lançamento” (`farm_drafts`) com:
   - `photoEvidenceId` (ou `storagePath`)
   - `status: DRAFT_PENDING_PARSE`
4. quando houver internet:
   - backend tenta “extrair” (OCR/IA) e propõe campos (`item`, `qty`, `data`)
5. humano confirma 1 toque (“Confirmar lançamento”)
6. sistema cria `farm_stock_move` oficial (append-only)

✅ Isso evita erro e mantém lastro.

3.2 Coleção de rascunhos

```
farm_drafts/{draftId}

{
  "tenantId":"TENANT_X",
  "propertyId":"prop_123",
  "type":"STOCK_OUT_FROM_PAPER",
  "evidenceId":"evg_...",
  "status":"PENDING_PARSE|PARSED|CONFIRMED|REJECTED",
  "suggested": { "itemId":"...", "qty": 2, "movementType":"OUT" },
  "createdAt": 1730000000000
```

```
}
```

Se você quiser, eu te mando o endpoint `POST /api/farm/drafts/from-photo` + `POST /api/farm/drafts/:id/confirm`.

---

4) Regras específicas para fazenda (ambiental + contrato)

Você já tem:

- `PropertyCommercialBlockRule` (CAR/IE)
- `CustodyBlocksFiscalRule` (custódia terceiros)

Agora adiciona regras pro estoque interno:

4.1 Regra: limites ambientais por aplicação

Ex.: aplicação de defensivo em área bloqueada ou acima do permitido.

```
// src/rules/rules/environmentLimits.rule.ts
import { Rule, RuleContext, RuleResult } from '../rules.types';

export class EnvironmentLimitsRule implements Rule {
  id = 'environmentLimits';

  applies(ctx: RuleContext) {
    return ctx.operation === 'FARM_STOCK_MOVE' &&
!!ctx.payload?.application;
  }

  async evaluate(ctx: RuleContext): Promise<RuleResult> {
    // aqui você consulta limites do CAR/zoneamento armazenado no
sistema
    // Exemplo: se o talhão está com restrição
    const app = ctx.payload.application;
    const blocked = false; // substituir por consulta real
    if (blocked) {
      return { ok:false, code:'ENV_LIMIT_BLOCK', message:'Aplicação
bloqueada por limite ambiental/CAR.', details:{ targetId: app.targetId
} };
    }
    return { ok:true };
  }
}
```

4.2 Regra: exige evidência quando operação crítica

Por exemplo, retirada alta de insumo caro, vacina controlada, agrotóxico.

```
// src/rules/rules/evidenceRequired.rule.ts
import { Rule, RuleContext, RuleResult } from '../rules.types';

export class EvidenceRequiredRule implements Rule {
  id = 'evidenceRequired';
```

```
  applies(ctx: RuleContext) {
    return ctx.operation === 'FARM_STOCK_MOVE';
  }

  async evaluate(ctx: RuleContext): Promise<RuleResult> {
    const qty = Number(ctx.payload?.qty || 0);
    const itemId = ctx.payload?.itemId;

    const critical = qty >= 10 || String(itemId).includes('AGROTOX')
|| String(itemId).includes('VACINA');
    if (!critical) return { ok:true };

    const ev = ctx.payload?.evidence;
    if (!ev?.required || (!ev.evidenceId && !ev.storagePath)) {
      return { ok:false, code:'EVIDENCE_REQUIRED', message:'Movimento
crítico exige evidência (foto/voz/GPS).', details:{ itemId, qty } };
    }

    return { ok:true };
  }
}
```

---

5) Conclusão: como fica “loja” vs “fazenda”

✅ **Fazenda (interno)**

- baixa / aplicação / consumo
- evidência (foto/voz/papel)
- regras ambientais + CAR/IE + contratos
- sem split na saída interna (salvo se for consignação/terceiro em custódia)

✅ **Loja (retail)**

- PDV e marketplace
- split/escrow obrigatório para multi-fornecedor/consignação
- rastreio por identidade (NF/serial/lote)

Próximo passo que eu já deixo pronto se você confirmar “sim” (você já confirmou):

1. Endpoint **`POST /api/farm/drafts/from-photo`** (cria rascunho com evidência)
2. Endpoint **`POST /api/farm/drafts/:id/confirm`** (gera movimento oficial)
3. Tela mobile “Baixa por foto” (botões grandes + voz)

Se você disser “sim”, eu mando esses 3 completos no mesmo padrão dos anteriores.

Entendido — vamos ajustar para ficar **perfeito no agro**:

- **Estoque interno da fazenda**: movimentação operacional (entrada/saída/consumo/aplicação), com evidência por foto/voz, offline, auditoria e regras ambientais/contratuais.
- **Estoque de loja**: é **venda** (PDV) ou "código" (catálogo/identidade) com split/escrow e rastreio de origem (NF/serial/lote).

Vou te entregar agora:

1. Separação de módulos (FarmInventory vs StoreRetail)
2. Endpoints de **baixa de estoque da fazenda** + evidência (foto/voz/planilha manual)
3. Regras: bloqueios por CAR/IE + limites ambientais + contrato + custódia
4. Como "foto de anotação manual" vira lançamento sem digitação (pipeline OCR/IA opcional, mas com fallback humano)

---

1) Módulos separados

1.1 FarmInventory (fazenda)

Coleções:

- `farm_items` (insumos internos: sal mineral, vacina, ração, fertilizante…)
- `farm_stock_balance` (saldo por propriedade/local)
- `farm_stock_moves` (append-only)
- `farm_applications` (aplicação em cultura/lote/animais)
- `evidence` (foto/voz/documento)
- `projects/contracts` (amarra travas e marcos)

Operações típicas:

- entrada (compra/recebimento)
- transferência entre depósitos
- consumo (baixa)
- aplicação (com evidência e alvo: talhão/lote/animal)

✅ Aqui normalmente **não existe split por fornecedor** em cada saída; o split e fiscal entram quando há venda/contrato/consignação, não na operação interna.

1.2 StoreRetail (loja)

Coleções:

- `catalog_items`, `inventory_units`, `pdv_sales`
- split/escrow obrigatório em multi-fornecedor
- rastreio por NF/serial/lote

✅ Aqui é venda, então split/escrow faz sentido sempre que houver consignação/múltiplos fornecedores.

---

2) Endpoints: estoque interno da fazenda (baixa + evidência)

2.1 `POST /api/farm/stock/movements` (movimentação interna)

DTO

```
// src/farm/dto/farm-movement.dto.ts
import { IsNotEmpty, IsOptional, IsString } from 'class-validator';

export class FarmStockMovementDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsString() @IsNotEmpty()
  propertyId: string;

  @IsString() @IsNotEmpty()
  itemId: string;

  @IsString() @IsNotEmpty()
  movementType: 'IN'|'OUT'|'TRANSFER';

  @IsNotEmpty()
  qty: number;

  @IsOptional() @IsString()
  fromLocationId?: string;

  @IsOptional() @IsString()
  toLocationId?: string;

  // opcional: aplicação vinculada
  @IsOptional()
  application?: {
    targetType: 'PASTURE'|'FIELD'|'ANIMAL_LOT'|'ANIMAL';
    targetId: string;
  };

  @IsOptional() @IsString()
  note?: string;

  // evidência (pode ser enviada depois via offline queue)
  @IsOptional()
  evidence?: {
    required: boolean;
    evidenceId?: string;     // se já comitada
    storagePath?: string;    // se já subiu e vai comitar depois
    riskType?: 'A'|'B';
  };
}
```

Controller

```
// src/farm/farm-stock.controller.ts
import { Body, Controller, Post, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { FarmStockService } from './farm-stock.service';
import { FarmStockMovementDto } from './dto/farm-movement.dto';

@Controller('/api/farm/stock')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class FarmStockController {
  constructor(private readonly svc: FarmStockService) {}

  @Post('/movements')
  @Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN')
  @Permissions('farm.stock.move')
  async move(@Actor() actor: any, @Body() dto: FarmStockMovementDto) {
    return this.svc.createMovement(actor, dto);
  }
}
```

Service (append-only + saldo + regras)

```
// src/farm/farm-stock.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';
import { RulesEngine } from '../rules/rules.engine';

@Injectable()
export class FarmStockService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly rules: RulesEngine,
  ) {}

  async createMovement(actor: any, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'farm.stock.move');
    const idemRef = this.db.collection('idempotency').doc(key);

    // regras: CAR/IE + limites ambientais + contrato (se a aplicação
exigir)
    const ctx = {
      actor,
      tenantId,
      operation: 'FARM_STOCK_MOVE',
      propertyId: dto.propertyId,
      payload: dto
    };
```

```
    const ruleRes = await this.rules.run(ctx);
    if (!ruleRes.ok) throw new
Error(`BLOCKED_${ruleRes.violations[0].code}`);

    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const moveId = makeId('fsm');
      const moveRef =
this.db.collection('farm_stock_moves').doc(moveId);

      // saldo atual
      const balKey =
`${dto.propertyId}_${dto.itemId}_${dto.fromLocationId || 'default'}`;
      const balRef =
this.db.collection('farm_stock_balance').doc(`${tenantId}_${balKey}`);
      const balSnap = await tx.get(balRef);
      const current = balSnap.exists ? (balSnap.data()!.qty || 0) : 0;

      const qty = Number(dto.qty);
      let next = current;

      if (dto.movementType === 'IN') next = current + qty;
      if (dto.movementType === 'OUT') next = current - qty;
      if (dto.movementType === 'TRANSFER') next = current - qty;

      if (next < 0) throw new Error('INSUFFICIENT_STOCK');

      tx.set(balRef, {
        tenantId,
        propertyId: dto.propertyId,
        itemId: dto.itemId,
        locationId: dto.fromLocationId || 'default',
        qty: next,
        updatedAt: now
      }, { merge: true });

      // se transfer, soma no destino
      if (dto.movementType === 'TRANSFER') {
        const toKey =
`${dto.propertyId}_${dto.itemId}_${dto.toLocationId || 'default'}`;
        const toRef =
this.db.collection('farm_stock_balance').doc(`${tenantId}_${toKey}`);
        const toSnap = await tx.get(toRef);
        const toCur = toSnap.exists ? (toSnap.data()!.qty || 0) : 0;
        tx.set(toRef, {
          tenantId,
          propertyId: dto.propertyId,
          itemId: dto.itemId,
          locationId: dto.toLocationId || 'default',
          qty: toCur + qty,
          updatedAt: now
        }, { merge: true });
      }

      // append-only do movimento (imutável)
      tx.create(moveRef, {
        moveId,
        tenantId,
        propertyId: dto.propertyId,
```

```
        itemId: dto.itemId,
        movementType: dto.movementType,
        qty,
        fromLocationId: dto.fromLocationId ?? null,
        toLocationId: dto.toLocationId ?? null,
        balanceBefore: current,
        balanceAfter: next,
        application: dto.application ?? null,
        note: dto.note ?? null,
        evidence: dto.evidence ?? null,
        performedBy: { uid: actor.uid, role: actor.role },
        performedAt: now
      });

      // auditoria
      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        propertyId: dto.propertyId,
        type: 'FARM_STOCK_MOVED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { moveId, itemId: dto.itemId, movementType:
dto.movementType, qty },
        ts: now
      });

      const out = { moveId, balanceAfter: next };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'farm.stock.move', clientRequestId: dto.clientRequestId, createdAt:
now, result: out });

      return out;
    });
  }
}
```

---

3) “Foto de anotação manual” → lançamento sem digitar

Aqui é o jeito mais seguro e compatível:

3.1 Fluxo no app

1. operador retira insumo e **anota na planilha de papel**
2. tira **foto** (evidência Tipo B ou DOC)
3. app cria “Rascunho de lançamento” (`farm_drafts`) com:
   - `photoEvidenceId` (ou `storagePath`)
   - `status: DRAFT_PENDING_PARSE`
4. quando houver internet:
   - backend tenta “extrair” (OCR/IA) e propõe campos (`item`, `qty`, `data`)
5. humano confirma 1 toque (“Confirmar lançamento”)
6. sistema cria `farm_stock_move` oficial (append-only)

✅ Isso evita erro e mantém lastro.

3.2 Coleção de rascunhos

```
farm_drafts/{draftId}

{
  "tenantId":"TENANT_X",
  "propertyId":"prop_123",
  "type":"STOCK_OUT_FROM_PAPER",
  "evidenceId":"evg_...",
  "status":"PENDING_PARSE|PARSED|CONFIRMED|REJECTED",
  "suggested": { "itemId":"...", "qty": 2, "movementType":"OUT" },
  "createdAt": 1730000000000
}
```

Se você quiser, eu te mando o endpoint `POST /api/farm/drafts/from-photo` + `POST /api/farm/drafts/:id/confirm`.

---

4) Regras específicas para fazenda (ambiental + contrato)

Você já tem:

- `PropertyCommercialBlockRule` (CAR/IE)
- `CustodyBlocksFiscalRule` (custódia terceiros)

Agora adiciona regras pro estoque interno:

4.1 Regra: limites ambientais por aplicação

Ex.: aplicação de defensivo em área bloqueada ou acima do permitido.

```
// src/rules/rules/environmentLimits.rule.ts
import { Rule, RuleContext, RuleResult } from '../rules.types';

export class EnvironmentLimitsRule implements Rule {
  id = 'environmentLimits';

  applies(ctx: RuleContext) {
    return ctx.operation === 'FARM_STOCK_MOVE' &&
!!ctx.payload?.application;
  }

  async evaluate(ctx: RuleContext): Promise<RuleResult> {
    // aqui você consulta limites do CAR/zoneamento armazenado no
sistema
    // Exemplo: se o talhão está com restrição
    const app = ctx.payload.application;
    const blocked = false; // substituir por consulta real
    if (blocked) {
      return { ok:false, code:'ENV_LIMIT_BLOCK', message:'Aplicação
bloqueada por limite ambiental/CAR.', details:{ targetId: app.targetId
} };
    }
    return { ok:true };
```

```
  }
}
```

4.2 Regra: exige evidência quando operação crítica

Por exemplo, retirada alta de insumo caro, vacina controlada, agrotóxico.

```
// src/rules/rules/evidenceRequired.rule.ts
import { Rule, RuleContext, RuleResult } from '../rules.types';

export class EvidenceRequiredRule implements Rule {
  id = 'evidenceRequired';

  applies(ctx: RuleContext) {
    return ctx.operation === 'FARM_STOCK_MOVE';
  }

  async evaluate(ctx: RuleContext): Promise<RuleResult> {
    const qty = Number(ctx.payload?.qty || 0);
    const itemId = ctx.payload?.itemId;

    const critical = qty >= 10 || String(itemId).includes('AGROTOX')
|| String(itemId).includes('VACINA');
    if (!critical) return { ok:true };

    const ev = ctx.payload?.evidence;
    if (!ev?.required || (!ev.evidenceId && !ev.storagePath)) {
      return { ok:false, code:'EVIDENCE_REQUIRED', message:'Movimento
crítico exige evidência (foto/voz/GPS).', details:{ itemId, qty } };
    }

    return { ok:true };
  }
}
```

---

5) Conclusão: como fica “loja” vs “fazenda”

✅ **Fazenda (interno)**

- baixa / aplicação / consumo
- evidência (foto/voz/papel)
- regras ambientais + CAR/IE + contratos
- sem split na saída interna (salvo se for consignação/terceiro em custódia)

✅ **Loja (retail)**

- PDV e marketplace
- split/escrow obrigatório para multi-fornecedor/consignação
- rastreio por identidade (NF/serial/lote)

Próximo passo que eu já deixo pronto se você confirmar “sim” (você já confirmou):

1. Endpoint **`POST /api/farm/drafts/from-photo`** (cria rascunho com evidência)
2. Endpoint **`POST /api/farm/drafts/:id/confirm`** (gera movimento oficial)
3. Tela mobile “Baixa por foto” (botões grandes + voz)

Se você disser “sim”, eu mando esses 3 completos no mesmo padrão dos anteriores.

Fechado — aqui vão **os 2 endpoints** que faltavam + **UI de lista grande com busca por voz** (sem complicar).

---

1) GET /api/farm/drafts/:id (read-only)

Controller

```
// src/farm/farm-drafts-read.controller.ts
import { Controller, Get, Param, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { FarmDraftsReadService } from './farm-drafts-read.service';

@Controller('/api/farm/drafts')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class FarmDraftsReadController {
  constructor(private readonly svc: FarmDraftsReadService) {}

  @Get('/:id')

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDITOR')
  @Permissions('farm.draft.read')
  async getOne(@Actor() actor: any, @Param('id') draftId: string) {
    return this.svc.getOne(actor, draftId);
  }
}
```

Service

```
// src/farm/farm-drafts-read.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

@Injectable()
export class FarmDraftsReadService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async getOne(actor: any, draftId: string) {
    const ref = this.db.collection('farm_drafts').doc(draftId);
    const snap = await ref.get();
    if (!snap.exists) throw new Error('DRAFT_NOT_FOUND');
```

```
    const d = snap.data()!;
    if (d.tenantId !== actor.tenantId) throw new
Error('DRAFT_FORBIDDEN');

    return d;
  }
}
```

---

2) GET /api/farm/items?propertyId=... (lista de insumos)

2.1 Modelo simples

```
farm_items/{itemId}

{
  "tenantId": "TENANT_X",
  "itemId": "itm_sal_mineral_25kg",
  "name": "Sal mineral 25kg",
  "category": "MINERAL",
  "unit": "SACO",
  "barcode": "789...",
  "active": true,
  "createdAt": 1730000000000
}
```

2.2 Controller

```
// src/farm/farm-items.controller.ts
import { Controller, Get, Query, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { FarmItemsService } from './farm-items.service';

@Controller('/api/farm/items')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class FarmItemsController {
  constructor(private readonly svc: FarmItemsService) {}

  @Get()

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDITOR')
  @Permissions('farm.items.read')
  async list(@Actor() actor: any, @Query('q') q?: string) {
    return this.svc.list(actor, q);
  }
}
```

2.3 Service (com busca simples por prefixo)

Firestore “contains” não é ótimo. Estratégia prática:

- manter `nameLower`

- pesquisar por prefixo usando >= e <

```
// src/farm/farm-items.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

function rangeEnd(prefix: string) {
  // truque de prefixo: prefix + \uf8ff
  return prefix + '\uf8ff';
}

@Injectable()
export class FarmItemsService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async list(actor: any, q?: string) {
    const tenantId = actor.tenantId;
    const base = this.db.collection('farm_items')
      .where('tenantId', '==', tenantId)
      .where('active', '==', true);

    if (!q || !q.trim()) {
      const snap = await base.orderBy('nameLower').limit(200).get();
      return { items: snap.docs.map(d => d.data()) };
    }

    const qq = q.trim().toLowerCase();

    const snap = await base
      .orderBy('nameLower')
      .startAt(qq)
      .endAt(rangeEnd(qq))
      .limit(200)
      .get();

    return { items: snap.docs.map(d => d.data()) };
  }
}
```

Ao criar/editar item, salve `nameLower = name.toLowerCase()`.

---

3) Tela mobile: selecionar item com botões grandes + busca por voz

3.1 Tela “Escolher insumo”

- campo grande de busca
- botão 🎙 “Falar”
- lista grande
- ao tocar, retorna `itemId` para a tela de confirmação

```
// src/screens/FarmItemPickerScreen.tsx
import React, { useMemo, useState } from 'react';
import { FlatList, Text, TextInput, View } from 'react-native';
```

```
import { useQuery } from '@tanstack/react-query';
import { api } from '../services/api';
import { BigButton } from '../ui/BigButton';

export function FarmItemPickerScreen({ navigation, route }: any) {
  const onPick = route.params?.onPick; // função callback (ou use
navigation params)
  const [qText, setQText] = useState('');

  const q = useQuery({
    queryKey: ['farmItems', qText],
    queryFn: async () => (await api.get('/api/farm/items', { params: {
q: qText } })).data,
  });

  const items = useMemo(() => q.data?.items || [], [q.data]);

  async function voiceSearch() {
    // aqui você usa Speech-to-text (expo-speech ou lib)
    // simulando:
    setQText('sal mineral');
  }

  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 24, fontWeight: '800' }}>Escolher
insumo</Text>
      <Text style={{ fontSize: 16, opacity: 0.8, marginTop: 8 }}>
        Digite ou fale o nome do insumo.
      </Text>

      <View style={{ marginTop: 12 }}>
        <TextInput
          value={qText}
          onChangeText={setQText}
          placeholder="Ex.: sal mineral, vacina, ração…"
          style={{ borderWidth: 1, borderRadius: 12, padding: 12,
fontSize: 16 }}
        />
        <View style={{ marginTop: 8 }}>
          <BigButton icon="🎙" title="Falar para buscar" subtitle="Ex.:
“sal mineral”" onPress={voiceSearch} />
        </View>
      </View>

      <FlatList
        style={{ marginTop: 12 }}
        data={items}
        keyExtractor={(it) => it.itemId}
        refreshing={q.isFetching}
        onRefresh={() => q.refetch()}
        renderItem={({ item }) => (
          <BigButton
            icon="📦"
            title={item.name}
            subtitle={`${item.category ?? ''} • ${item.unit ?? ''}`}
            onPress={() => {
              if (typeof onPick === 'function') onPick(item);
              navigation.goBack();
            }}
          />
```

```
        )}
      />
    </View>
  );
}
```

3.2 Usar no “Confirmar Draft”

No `FarmDraftConfirmScreen`, em vez de digitar `itemId`, você chama:

```
<BigButton
  icon="🔎"
  title="Escolher insumo"
  subtitle={itemId ? `Selecionado: ${itemId}` : 'Toque para buscar'}
  onPress={() => navigation.navigate('FarmItemPicker', {
    onPick: (it: any) => setItemId(it.itemId)
  })}
/>
```

---

4) Permissões (mínimo)

- `farm.draft.read`
- `farm.items.read`

---

5) Checklist de “funciona perfeito” (sem furos)

✅ Draft criado mesmo offline (com evidência depois)
✅ Confirmação gera movimento oficial append-only
✅ Saldo sempre consistente
✅ Auditoria registra tudo
✅ Busca fácil (texto/voz) para pessoas com dificuldade de leitura

---

Se você quiser, o próximo módulo que fecha o agro “de verdade” é:
✅ **Aplicação de insumo** (vincular talhão/lote/animal) + limites ambientais (CAR) + laudo/laboratório como evidência DOC.

Beleza — vamos fechar o módulo **Aplicação de Insumo** (fazenda) com:

- aplicação em **talhão/pasto** (culturas/silvicultura)
- aplicação em **lote/animal** (vacina, vermífugo, ração, sal)
- **limites ambientais (CAR/zoneamento)** como regra de bloqueio
- **laudo/laboratório** como evidência DOC
- **offline** + evidência por foto/voz/planilha manual
- **auditoria append-only**

Vou entregar:

1. modelos Firestore
2. endpoints completos
3. RulesEngine específico (limite ambiental + exigência de laudo)
4. UI mobile simples (botões grandes)

---

1) Firestore: modelos

1.1 Entidades de “alvo” (targets)

Talhão / área (culturas/silvicultura)

```
fields/{fieldId}

{
  "tenantId": "TENANT_X",
  "fieldId": "fld_001",
  "propertyId": "prop_123",
  "name": "Talhão 1",
  "type": "CROP|SILVICULTURE|PASTURE",
  "crop": { "name": "Milho" },
  "geoFence": { "type": "POLYGON", "coords": [] }
}
```

Lote animal

```
animal_lots/{lotId}

{
  "tenantId": "TENANT_X",
  "lotId": "lot_001",
  "propertyId": "prop_123",
  "species": "BOVINE|OVINE|FISH|POULTRY",
  "name": "Lote recria 2026"
}
```

1.2 Aplicação (append-only)

```
farm_applications/{applicationId}

{
  "tenantId":"TENANT_X",
  "applicationId":"app_001",
  "propertyId":"prop_123",

  "itemId":"itm_sal_mineral_25kg",
  "qty": 2,
  "unit":"SACO",

  "target": { "type":"FIELD|PASTURE|ANIMAL_LOT|ANIMAL", "id":"fld_001"
},
  "applicationType":"FERTILIZER|PESTICIDE|FEED|VACCINE|MINERAL|OTHER",

  "relatedMoveId":"fsm_...",
  "evidence": {
    "riskType":"B",
```

```
    "evidenceIds": ["evg_..."],
    "docs": ["evg_doc_..."]
  },

  "createdAt": 1730000000000,
  "createdBy": { "uid":"...", "role":"OPERATOR" }
}
```

1.3 Limites ambientais (CAR / zona)

```
environment_limits/{limitId}

{
  "tenantId":"TENANT_X",
  "limitId":"env_001",
  "propertyId":"prop_123",
  "targetType":"FIELD|PASTURE",
  "targetId":"fld_001",
  "status":"ACTIVE",
  "rules": {
    "blockedApplicationTypes":["PESTICIDE"],
    "maxKgPerHaPerMonth": { "FERTILIZER": 200 },
    "requiresLabDocFor":["PESTICIDE","FERTILIZER"]
  }
}
```

---

2) Endpoints completos

2.1 `POST /api/farm/applications` (baixa estoque + registra aplicação)

DTO

```
// src/farm/dto/farm-application.dto.ts
import { IsArray, IsNotEmpty, IsOptional, IsString } from 'class-
validator';

export class FarmApplicationDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsString() @IsNotEmpty()
  propertyId: string;

  @IsString() @IsNotEmpty()
  itemId: string;

  @IsNotEmpty()
  qty: number;

  @IsString() @IsNotEmpty()
  fromLocationId: string;

  @IsString() @IsNotEmpty()
  applicationType:
'FERTILIZER'|'PESTICIDE'|'FEED'|'VACCINE'|'MINERAL'|'OTHER';

  @IsNotEmpty()
```

```
  target: { type: 'FIELD'|'PASTURE'|'ANIMAL_LOT'|'ANIMAL'; id: string
};

  @IsOptional()
  note?: string;

  // evidências anexadas (foto/voz/doc)
  @IsOptional()
  evidenceIds?: string[];

  @IsOptional()
  docEvidenceIds?: string[]; // laudos/laboratório PDF
}
```

Controller

```
// src/farm/farm-applications.controller.ts
import { Body, Controller, Post, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { FarmApplicationsService } from './farm-applications.service';
import { FarmApplicationDto } from './dto/farm-application.dto';

@Controller('/api/farm/applications')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class FarmApplicationsController {
  constructor(private readonly svc: FarmApplicationsService) {}

  @Post()
  @Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN')
  @Permissions('farm.application.create')
  async create(@Actor() actor: any, @Body() dto: FarmApplicationDto) {
    return this.svc.create(actor, dto);
  }
}
```

Service (transação: baixa estoque + cria aplicação + auditoria)

```
// src/farm/farm-applications.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';
import { RulesEngine } from '../rules/rules.engine';

@Injectable()
export class FarmApplicationsService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly rules: RulesEngine,
  ) {}

  async create(actor: any, dto: any) {
```

```
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'farm.application.create');
    const idemRef = this.db.collection('idempotency').doc(key);

    // regras ambientais + CAR/IE + evidência/ laudo
    const ctx = {
      actor,
      tenantId,
      operation: 'FARM_APPLICATION',
      propertyId: dto.propertyId,
      payload: dto
    };
    const ruleRes = await this.rules.run(ctx);
    if (!ruleRes.ok) throw new
Error(`BLOCKED_${ruleRes.violations[0].code}`);

    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      // 1) baixa estoque (OUT)
      const balKey =
`${dto.propertyId}_${dto.itemId}_${dto.fromLocationId || 'default'}`;
      const balRef =
this.db.collection('farm_stock_balance').doc(`${tenantId}_${balKey}`);
      const balSnap = await tx.get(balRef);
      const current = balSnap.exists ? (balSnap.data()!.qty || 0) : 0;

      const qty = Number(dto.qty);
      const next = current - qty;
      if (next < 0) throw new Error('INSUFFICIENT_STOCK');

      tx.set(balRef, {
        tenantId,
        propertyId: dto.propertyId,
        itemId: dto.itemId,
        locationId: dto.fromLocationId || 'default',
        qty: next,
        updatedAt: now
      }, { merge: true });

      const moveId = makeId('fsm');
      tx.create(this.db.collection('farm_stock_moves').doc(moveId), {
        moveId,
        tenantId,
        propertyId: dto.propertyId,
        itemId: dto.itemId,
        movementType: 'OUT',
        qty,
        fromLocationId: dto.fromLocationId,
        toLocationId: null,
        balanceBefore: current,
        balanceAfter: next,
        application: dto.target ? { targetType: dto.target.type,
targetId: dto.target.id } : null,
        note: dto.note ?? null,
        evidence: {
          riskType: 'B',
```

```
          evidenceIds: dto.evidenceIds ?? [],
          docEvidenceIds: dto.docEvidenceIds ?? []
        },
        performedBy: { uid: actor.uid, role: actor.role },
        performedAt: now
      });

      // 2) cria aplicação (append-only)
      const applicationId = makeId('app');

tx.create(this.db.collection('farm_applications').doc(applicationId),
{
        tenantId,
        applicationId,
        propertyId: dto.propertyId,
        itemId: dto.itemId,
        qty,
        target: dto.target,
        applicationType: dto.applicationType,
        relatedMoveId: moveId,
        evidence: {
          riskType: 'B',
          evidenceIds: dto.evidenceIds ?? [],
          docs: dto.docEvidenceIds ?? []
        },
        note: dto.note ?? null,
        createdAt: now,
        createdBy: { uid: actor.uid, role: actor.role }
      });

      // 3) auditoria
      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        propertyId: dto.propertyId,
        type: 'FARM_APPLICATION_CREATED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { applicationId, moveId, itemId: dto.itemId, qty,
target: dto.target, applicationType: dto.applicationType },
        ts: now
      });

      const out = { applicationId, moveId, balanceAfter: next };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'farm.application.create', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });

      return out;
    });
  }
}
```

---

3) Regras (RulesEngine) específicas da aplicação

3.1 Regra: bloqueio ambiental por target (CAR/zoneamento)

```
// src/rules/rules/environmentApplication.rule.ts
import { Inject } from '@nestjs/common';
import { FIRESTORE } from '../../common/firestore.provider';
```

```
import { Rule, RuleContext, RuleResult } from '../rules.types';

export class EnvironmentApplicationRule implements Rule {
  id = 'environmentApplication';

  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  applies(ctx: RuleContext) {
    return ctx.operation === 'FARM_APPLICATION' &&
!!ctx.payload?.target;
  }

  async evaluate(ctx: RuleContext): Promise<RuleResult> {
    const { propertyId, target, applicationType } = ctx.payload;

    // só aplica a FIELD/PASTURE
    if (!['FIELD','PASTURE'].includes(target.type)) return { ok: true
};

    const snap = await this.db.collection('environment_limits')
      .where('tenantId', '==', ctx.tenantId)
      .where('propertyId', '==', propertyId)
      .where('targetType', '==', target.type)
      .where('targetId', '==', target.id)
      .where('status', '==', 'ACTIVE')
      .limit(20)
      .get();

    if (snap.empty) return { ok: true };

    const lim = snap.docs[0].data();
    const blocked = (lim.rules?.blockedApplicationTypes ||
[]).includes(applicationType);
    if (blocked) {
      return { ok:false, code:'ENV_APP_BLOCKED', message:'Aplicação
bloqueada por limite ambiental (CAR).', details:{ limitId: lim.limitId
} };
    }
    return { ok:true };
  }
}
```

3.2 Regra: exige laudo para certos tipos

```
// src/rules/rules/labDocRequired.rule.ts
import { Inject } from '@nestjs/common';
import { FIRESTORE } from '../../common/firestore.provider';
import { Rule, RuleContext, RuleResult } from '../rules.types';

export class LabDocRequiredRule implements Rule {
  id = 'labDocRequired';

  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  applies(ctx: RuleContext) {
    return ctx.operation === 'FARM_APPLICATION';
  }

  async evaluate(ctx: RuleContext): Promise<RuleResult> {
    const { propertyId, target, applicationType, docEvidenceIds } =
ctx.payload;
```

```
    if (!target || ![‘FIELD’,’PASTURE’].includes(target.type)) return
{ ok:true };

    const snap = await this.db.collection('environment_limits')
      .where('tenantId', '==', ctx.tenantId)
      .where('propertyId', '==', propertyId)
      .where('targetType', '==', target.type)
      .where('targetId', '==', target.id)
      .where('status', '==', 'ACTIVE')
      .limit(20)
      .get();

    if (snap.empty) return { ok:true };

    const lim = snap.docs[0].data();
    const needs = (lim.rules?.requiresLabDocFor ||
[]).includes(applicationType);
    if (!needs) return { ok:true };

    if (!Array.isArray(docEvidenceIds) || docEvidenceIds.length === 0)
{
      return { ok:false, code:'LAB_DOC_REQUIRED', message:'Aplicação
exige laudo/documento (PDF) como evidência.', details:{ limitId:
lim.limitId, applicationType } };
    }

    return { ok:true };
  }
}
```

Você adiciona essas regras no `RulesModule` junto com as outras.

---

4) UI mobile “Aplicar insumo” (grande e simples)

Fluxo:

1. escolher propriedade
2. escolher insumo (usa `FarmItemPicker`)
3. escolher alvo (Talhão/Pasto/Lote/Animal)
4. informar quantidade (+ / -)
5. anexar evidência (foto/voz/laudo)
6. enviar

Tela

```
// src/screens/FarmApplyInputScreen.tsx
import React, { useState } from 'react';
import { Text, TextInput, View } from 'react-native';
import { BigButton } from '../ui/BigButton';
import { api } from '../services/api';
import { v4 as uuid } from 'uuid';

export function FarmApplyInputScreen({ navigation, route }: any) {
  const { propertyId } = route.params;
```

```
  const [itemId, setItemId] = useState('');
  const [qty, setQty] = useState('1');
  const [fromLocationId, setFromLocationId] = useState('default');
  const [applicationType, setApplicationType] =
useState<'FERTILIZER'|'PESTICIDE'|'FEED'|'VACCINE'|'MINERAL'|'OTHER'>(
'MINERAL');
  const [target, setTarget] = useState<any>(null);

  const [evidenceIds, setEvidenceIds] = useState<string[]>([]);
  const [docEvidenceIds, setDocEvidenceIds] = useState<string[]>([]);

  async function attachPhotoMock() {
    // ideal: usar Evidence module (upload-url + commit), aqui só
simula:
    setEvidenceIds((prev) => [...prev, `evg_${Date.now()}`]);
  }

  async function attachLabDocMock() {
    setDocEvidenceIds((prev) => [...prev, `evg_doc_${Date.now()}`]);
  }

  async function submit() {
    await api.post('/api/farm/applications', {
      clientRequestId: uuid(),
      propertyId,
      itemId,
      qty: Number(qty),
      fromLocationId,
      applicationType,
      target,
      evidenceIds,
      docEvidenceIds
    });
    navigation.goBack();
  }

  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 24, fontWeight: '800' }}>Aplicar
Insumo</Text>
      <Text style={{ fontSize: 16, opacity: 0.8, marginTop: 8 }}>
        Registra a aplicação e baixa o estoque automaticamente.
      </Text>

      <BigButton
        icon="📦"
        title="Escolher insumo"
        subtitle={itemId ? `Selecionado: ${itemId}` : 'Toque para
buscar'}
        onPress={() => navigation.navigate('FarmItemPicker', { onPick:
(it: any) => setItemId(it.itemId) })}
      />

      <BigButton
        icon="🎯"
        title="Escolher alvo"
        subtitle={target ? `${target.type} • ${target.id}` : 'Talhão /
Pasto / Lote / Animal'}
        onPress={() => {
          // você pode criar um picker igual do item (FieldPicker /
LotPicker)
```

```
          setTarget({ type: 'FIELD', id: 'fld_001' }); // mock
        }}
      />

      <Text style={{ marginTop: 12, fontSize: 16, fontWeight: '700'
}}>Tipo</Text>
      <TextInput value={applicationType}
onChangeText={(t:any)=>setApplicationType(t)} style={{ borderWidth:1,
borderRadius:12, padding:12, marginTop:8 }} />

      <Text style={{ marginTop: 12, fontSize: 16, fontWeight: '700'
}}>Quantidade</Text>
      <TextInput value={qty} onChangeText={setQty}
keyboardType="numeric" style={{ borderWidth:1, borderRadius:12,
padding:12, marginTop:8 }} />

      <Text style={{ marginTop: 12, fontSize: 16, fontWeight: '700'
}}>Local de estoque</Text>
      <TextInput value={fromLocationId}
onChangeText={setFromLocationId} style={{ borderWidth:1,
borderRadius:12, padding:12, marginTop:8 }} />

      <View style={{ marginTop: 12 }}>
        <BigButton icon="📷" title="Anexar foto (evidência)"
subtitle={`Fotos: ${evidenceIds.length}`} onPress={attachPhotoMock} />
        <BigButton icon="📄" title="Anexar laudo (PDF)"
subtitle={`Laudos: ${docEvidenceIds.length}`}
onPress={attachLabDocMock} />
      </View>

      <View style={{ marginTop: 12 }}>
        <BigButton icon="✅" title="Registrar aplicação"
subtitle="Baixa estoque + auditoria" onPress={submit}
disabled={!itemId || !target || Number(qty)<=0} />
      </View>
    </View>
  );
}
```

---

5) Permissões

- `farm.application.create`
- `farm.items.read`

Se você quiser, eu fecho o “alvo” com **pickers reais**:

- `GET /api/farm/fields?propertyId=...`
- `GET /api/farm/animal-lots?propertyId=...`
- `GET /api/farm/animals/search?q=...` (por brinco/RFID)

e as telas de seleção grandes, com busca por voz igual ao insumo.

Perfeito — abaixo vão **os 3 endpoints + 3 telas mobile (pickers)** no mesmo padrão "grande e simples", com busca por texto e botão 🎙 (simulado).

---

1) GET /api/farm/fields?propertyId=... (talhões/pastos)

1.1 Controller

```
// src/farm/farm-fields.controller.ts
import { Controller, Get, Query, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { FarmFieldsService } from './farm-fields.service';

@Controller('/api/farm/fields')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class FarmFieldsController {
  constructor(private readonly svc: FarmFieldsService) {}

  @Get()

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDITOR')
  @Permissions('farm.fields.read')
  async list(
    @Actor() actor: any,
    @Query('propertyId') propertyId: string,
    @Query('q') q?: string
  ) {
    return this.svc.list(actor, propertyId, q);
  }
}
```

1.2 Service (prefix search por `nameLower`)

```
// src/farm/farm-fields.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

function rangeEnd(prefix: string) {
  return prefix + '\uf8ff';
}

@Injectable()
export class FarmFieldsService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async list(actor: any, propertyId: string, q?: string) {
    if (!propertyId) throw new Error('PROPERTY_ID_REQUIRED');

    const tenantId = actor.tenantId;
    const base = this.db.collection('fields')
      .where('tenantId', '==', tenantId)
      .where('propertyId', '==', propertyId);
```

```
    if (!q || !q.trim()) {
      const snap = await base.orderBy('nameLower').limit(200).get();
      return { items: snap.docs.map(d => d.data()) };
    }

    const qq = q.trim().toLowerCase();
    const snap = await base
      .orderBy('nameLower')
      .startAt(qq)
      .endAt(rangeEnd(qq))
      .limit(200)
      .get();

    return { items: snap.docs.map(d => d.data()) };
  }
}
```

---

2) GET /api/farm/animal-lots?propertyId=...

2.1 Controller

```
// src/farm/farm-animal-lots.controller.ts
import { Controller, Get, Query, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { FarmAnimalLotsService } from './farm-animal-lots.service';

@Controller('/api/farm/animal-lots')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class FarmAnimalLotsController {
  constructor(private readonly svc: FarmAnimalLotsService) {}

  @Get()

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDITOR')
  @Permissions('farm.animalLots.read')
  async list(
    @Actor() actor: any,
    @Query('propertyId') propertyId: string,
    @Query('q') q?: string
  ) {
    return this.svc.list(actor, propertyId, q);
  }
}
```

2.2 Service

```
// src/farm/farm-animal-lots.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

function rangeEnd(prefix: string) {
  return prefix + '\uf8ff';
}

@Injectable()
```

```
export class FarmAnimalLotsService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async list(actor: any, propertyId: string, q?: string) {
    if (!propertyId) throw new Error('PROPERTY_ID_REQUIRED');

    const tenantId = actor.tenantId;
    const base = this.db.collection('animal_lots')
      .where('tenantId', '==', tenantId)
      .where('propertyId', '==', propertyId);

    if (!q || !q.trim()) {
      const snap = await base.orderBy('nameLower').limit(200).get();
      return { items: snap.docs.map(d => d.data()) };
    }

    const qq = q.trim().toLowerCase();
    const snap = await
base.orderBy('nameLower').startAt(qq).endAt(rangeEnd(qq)).limit(200).g
et();
    return { items: snap.docs.map(d => d.data()) };
  }
}
```

---

3) GET /api/farm/animals/search?propertyId=...&q=...

Busca por:

- brinco / RFID / EID
- também por `animalId` (se quiser)
- opcionalmente por mãe (motherIdentity)

3.1 Controller

```
// src/farm/farm-animals-search.controller.ts
import { Controller, Get, Query, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { FarmAnimalsSearchService } from './farm-animals-
search.service';

@Controller('/api/farm/animals')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class FarmAnimalsSearchController {
  constructor(private readonly svc: FarmAnimalsSearchService) {}

  @Get('/search')

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDITOR',
'LOGISTICS')
  @Permissions('farm.animals.read')
```

```
  async search(
    @Actor() actor: any,
    @Query('propertyId') propertyId: string,
    @Query('q') q: string
  ) {
    return this.svc.search(actor, propertyId, q);
  }
}
```

3.2 Service

```
// src/farm/farm-animals-search.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

@Injectable()
export class FarmAnimalsSearchService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async search(actor: any, propertyId: string, q: string) {
    if (!propertyId) throw new Error('PROPERTY_ID_REQUIRED');
    if (!q || !q.trim()) return { items: [] };

    const tenantId = actor.tenantId;
    const qq = q.trim().toUpperCase();

    // busca por identity.value exato (mais confiável)
    const snap = await this.db.collection('animals')
      .where('tenantId', '==', tenantId)
      .where('propertyId', '==', propertyId)
      .where('identity.value', '==', qq)
      .limit(50)
      .get();

    if (!snap.empty) return { items: snap.docs.map(d => d.data()) };

    // fallback: busca por animalId exato
    const snap2 = await this.db.collection('animals')
      .where('tenantId', '==', tenantId)
      .where('propertyId', '==', propertyId)
      .where('animalId', '==', qq)
      .limit(50)
      .get();

    return { items: snap2.docs.map(d => d.data()) };
  }
}
```

---

4) Permissões

- farm.fields.read
- farm.animalLots.read
- farm.animals.read

---

5) Telas mobile (Pickers)

5.1 FieldPicker (Talhão/Pasto)

```
// src/screens/FarmFieldPickerScreen.tsx
import React, { useMemo, useState } from 'react';
import { FlatList, Text, TextInput, View } from 'react-native';
import { useQuery } from '@tanstack/react-query';
import { api } from '../services/api';
import { BigButton } from '../ui/BigButton';

export function FarmFieldPickerScreen({ navigation, route }: any) {
  const { propertyId, onPick } = route.params;
  const [qText, setQText] = useState('');

  const q = useQuery({
    queryKey: ['farmFields', propertyId, qText],
    queryFn: async () => (await api.get('/api/farm/fields', { params:
{ propertyId, q: qText } })).data,
  });

  const items = useMemo(() => q.data?.items || [], [q.data]);

  async function voiceSearch() {
    setQText('talhão'); // simulado
  }

  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 24, fontWeight: '800' }}>Escolher
talhão/pasto</Text>

      <TextInput
        value={qText}
        onChangeText={setQText}
        placeholder="Ex.: talhão 1, pasto norte…"
        style={{ borderWidth: 1, borderRadius: 12, padding: 12,
fontSize: 16, marginTop: 12 }}
      />

      <BigButton icon="🎙" title="Falar para buscar" subtitle="Ex.:
“talhão 1”" onPress={voiceSearch} />

      <FlatList
        style={{ marginTop: 12 }}
        data={items}
        keyExtractor={(it) => it.fieldId}
        refreshing={q.isFetching}
        onRefresh={() => q.refetch()}
        renderItem={({ item }) => (
          <BigButton
            icon="🌱"
            title={item.name}
            subtitle={`${item.type ?? ''} • ${item.crop?.name ?? ''}`}
            onPress={() => { if (typeof onPick === 'function')
onPick({ type: item.type === 'PASTURE' ? 'PASTURE' : 'FIELD', id:
item.fieldId }); navigation.goBack(); }}
          />
        )}
      />
    </View>
```

```
  );
}
```

5.2 LotPicker (Lote animal)

```
// src/screens/FarmLotPickerScreen.tsx
import React, { useMemo, useState } from 'react';
import { FlatList, Text, TextInput, View } from 'react-native';
import { useQuery } from '@tanstack/react-query';
import { api } from '../services/api';
import { BigButton } from '../ui/BigButton';

export function FarmLotPickerScreen({ navigation, route }: any) {
  const { propertyId, onPick } = route.params;
  const [qText, setQText] = useState('');

  const q = useQuery({
    queryKey: ['farmLots', propertyId, qText],
    queryFn: async () => (await api.get('/api/farm/animal-lots', {
params: { propertyId, q: qText } })).data,
  });

  const items = useMemo(() => q.data?.items || [], [q.data]);

  async function voiceSearch() {
    setQText('recria'); // simulado
  }

  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 24, fontWeight: '800' }}>Escolher lote
animal</Text>

      <TextInput
        value={qText}
        onChangeText={setQText}
        placeholder="Ex.: recria, engorda, peixe tanque…"
        style={{ borderWidth: 1, borderRadius: 12, padding: 12,
fontSize: 16, marginTop: 12 }}
      />

      <BigButton icon="🎙" title="Falar para buscar" subtitle="Ex.:
“recria”" onPress={voiceSearch} />

      <FlatList
        style={{ marginTop: 12 }}
        data={items}
        keyExtractor={(it) => it.lotId}
        refreshing={q.isFetching}
        onRefresh={() => q.refetch()}
        renderItem={({ item }) => (
          <BigButton
            icon="🐄"
            title={item.name}
            subtitle={`${item.species ?? ''}`}
            onPress={() => { if (typeof onPick === 'function')
onPick({ type: 'ANIMAL_LOT', id: item.lotId }); navigation.goBack();
}}
          />
        )}
      />
    </View>
```

```
  );
}
```

5.3 AnimalPicker (buscar por brinco/RFID)

```
// src/screens/FarmAnimalPickerScreen.tsx
import React, { useMemo, useState } from 'react';
import { FlatList, Text, TextInput, View } from 'react-native';
import { useQuery } from '@tanstack/react-query';
import { api } from '../services/api';
import { BigButton } from '../ui/BigButton';

export function FarmAnimalPickerScreen({ navigation, route }: any) {
  const { propertyId, onPick } = route.params;
  const [qText, setQText] = useState('');

  const q = useQuery({
    queryKey: ['farmAnimalsSearch', propertyId, qText],
    enabled: (qText.trim().length >= 2),
    queryFn: async () => (await api.get('/api/farm/animals/search', {
params: { propertyId, q: qText } })).data,
  });

  const items = useMemo(() => q.data?.items || [], [q.data]);

  async function voiceSearch() {
    setQText('BR123'); // simulado
  }

  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 24, fontWeight: '800' }}>Escolher
animal</Text>
      <Text style={{ fontSize: 16, opacity: 0.8, marginTop: 8 }}>
        Digite ou fale o brinco/RFID.
      </Text>

      <TextInput
        value={qText}
        onChangeText={setQText}
        placeholder="Ex.: BR123"
        style={{ borderWidth: 1, borderRadius: 12, padding: 12,
fontSize: 16, marginTop: 12 }}
      />

      <BigButton icon="🎙" title="Falar para buscar" subtitle="Ex.:
“BR123”" onPress={voiceSearch} />

      <FlatList
        style={{ marginTop: 12 }}
        data={items}
        keyExtractor={(it) => it.animalId}
        renderItem={({ item }) => (
          <BigButton
            icon="🐂"
            title={`${item.identity?.value ?? item.animalId}`}
            subtitle={`${item.species ?? ''} • ${item.status ?? ''}`}
            onPress={() => { if (typeof onPick === 'function')
onPick({ type: 'ANIMAL', id: item.animalId, identity: item.identity
}); navigation.goBack(); }}
          />
        )}
```

```
      />
    </View>
  );
}
```

---

6) Conectar na tela “Aplicar insumo”

Na `FarmApplyInputScreen`, substitua o mock do target por navegação:

- Botão “Escolher alvo” abre menu:
  - Talhão/Pasto → `FarmFieldPicker`
  - Lote animal → `FarmLotPicker`
  - Animal → `FarmAnimalPicker`

Exemplo simples:

```
<BigButton
  icon="🎯"
  title="Escolher alvo"
  subtitle={target ? `${target.type} • ${target.id}` : 'Talhão / Pasto
/ Lote / Animal'}
  onPress={() => navigation.navigate('FarmFieldPicker', { propertyId,
onPick: setTarget })}
/>
```

(Se quiser, eu te mando uma tela “TargetPicker” que abre as 3 opções com botões grandes.)

Se você quiser fechar o pacote final do agro, o próximo bloco é:
✅ **Laboratório / Laudos**: endpoint para upload/commit de PDF + vincular ao item/aplicação + verificação por auditor (read-only) + regra “sem laudo não aplica”.

Perfeito — vamos fechar o módulo **Laboratório / Laudos (PDF)** com:

- **upload + commit** (evidência DOC)
- **vincular** laudo a aplicação/animal/lote/talhão/contrato
- **validação por AUDITOR/TECHNICIAN** (read-only + parecer)
- **RulesEngine**: “sem laudo não aplica” (quando regra exigir)
- **append-only** (histórico imutável)

Vou te entregar agora:

1. Modelos Firestore
2. Endpoints completos: upload-url, commit, link, verify, list
3. Regras no RulesEngine (documento obrigatório + documento verificado)
4. Fluxo mobile simples

---

1) Firestore: modelos

1.1 Documento/Laboratório (evidência tipo DOC)

```
lab_docs/{docId}

{
  "tenantId": "TENANT_X",
  "docId": "doc_001",
  "propertyId": "prop_123",
  "docType": "LAB_REPORT|INVOICE|CERTIFICATE|ANALYSIS",
  "title": "Laudo solo - Talhão 1 - Jan/2026",
  "file": {
    "storagePath": "tenants/TENANT_X/lab_docs/doc_001.pdf",
    "mime": "application/pdf",
    "size": 123456
  },
  "hash": { "sha256": "..." },

  "targets": [
    { "type": "FIELD", "id": "fld_001" }
  ],

  "status": "UPLOADED|LINKED|VERIFIED|REJECTED",
  "createdAt": 1730000000000,
  "createdBy": { "uid": "...", "role": "TECHNICIAN" },

  "verification": {
    "status": "PENDING|VERIFIED|REJECTED",
    "verifiedAt": null,
    "verifiedBy": null,
    "opinionId": null,
    "reason": null
  }
}
```

1.2 Vinculação (append-only) para rastrear “quem ligou no quê”

```
doc_links/{linkId}

{
  "tenantId": "TENANT_X",
  "linkId": "lnk_001",
  "docId": "doc_001",
  "target": { "type": "FARM_APPLICATION", "id": "app_001" },
  "createdAt": 1730000000000,
  "createdBy": { "uid": "...", "role": "OPERATOR" }
}
```

---

2) Endpoints

2.1 `POST /api/lab/docs/upload-url` (gera URL p/ subir PDF)

DTO

```
// src/lab/dto/lab-upload-url.dto.ts
import { IsNotEmpty, IsString } from 'class-validator';
```

```
export class LabUploadUrlDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsString() @IsNotEmpty()
  propertyId: string;

  @IsString() @IsNotEmpty()
  filename: string; // ex: laudo_solo.pdf

  @IsString() @IsNotEmpty()
  mime: string; // application/pdf
}
```

Resposta

```
{
  "docId": "doc_001",
  "storagePath": "tenants/TENANT_X/lab_docs/doc_001.pdf",
  "uploadUrl": "https://signed-url..."
}
```

Implementação: se você já usa Firebase Storage, você pode:

- usar **upload direto do app** (sem signed URL) com SDK do Firebase
- ou **signed URL** via Cloud Function/Server.
  Eu vou te deixar a versão "agnóstica": backend cria doc e retorna `storagePath`.
  O app sobe pelo Firebase SDK e depois chama `commit`.

2.2 `POST /api/lab/docs/commit` (confirma upload e grava hash/metadata)

DTO

```
// src/lab/dto/lab-commit.dto.ts
import { IsArray, IsNotEmpty, IsOptional, IsString } from 'class-
validator';

export class LabCommitDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsString() @IsNotEmpty()
  docId: string;

  @IsString() @IsNotEmpty()
  title: string;

  @IsString() @IsNotEmpty()
  docType: 'LAB_REPORT'|'INVOICE'|'CERTIFICATE'|'ANALYSIS';

  @IsOptional() @IsArray()
  targets?: { type:
'FIELD'|'PASTURE'|'ANIMAL_LOT'|'ANIMAL'|'PROPERTY'; id: string }[];
```

```
  @IsString() @IsNotEmpty()
  sha256: string; // hash do PDF calculado no app

  @IsNotEmpty()
  size: number;
}
```

2.3 `POST /api/lab/docs/:docId/link` (vincular a aplicação/contrato/etc)

DTO

```
// src/lab/dto/lab-link.dto.ts
import { IsNotEmpty, IsString } from 'class-validator';

export class LabLinkDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsNotEmpty()
  target: { type: 'FARM_APPLICATION'|'CONTRACT'|'PROJECT'|'NF_ISSUE';
id: string };
}
```

2.4 `POST /api/lab/docs/:docId/verify` (auditor valida laudo)

DTO

```
// src/lab/dto/lab-verify.dto.ts
import { IsIn, IsNotEmpty, IsOptional, IsString } from 'class-
validator';

export class LabVerifyDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsIn(['VERIFIED','REJECTED'])
  decision: 'VERIFIED'|'REJECTED';

  @IsOptional() @IsString()
  reason?: string;

  @IsOptional() @IsString()
  opinionSummary?: string; // gera opinion imutável (hash/cadeia)
}
```

2.5 `GET /api/lab/docs?propertyId=...&q=...` (listar/pesquisar)

---

3) Implementação NestJS (Controller + Service)

3.1 Controller

```
// src/lab/lab.controller.ts
import { Body, Controller, Get, Param, Post, Query, UseGuards } from
'@nestjs/common';
```

```
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { LabService } from './lab.service';
import { LabUploadUrlDto } from './dto/lab-upload-url.dto';
import { LabCommitDto } from './dto/lab-commit.dto';
import { LabLinkDto } from './dto/lab-link.dto';
import { LabVerifyDto } from './dto/lab-verify.dto';

@Controller('/api/lab/docs')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class LabController {
  constructor(private readonly svc: LabService) {}

  @Post('/upload-url')
  @Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN')
  @Permissions('lab.docs.create')
  uploadUrl(@Actor() actor: any, @Body() dto: LabUploadUrlDto) {
    return this.svc.createUploadTarget(actor, dto);
  }

  @Post('/commit')
  @Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN')
  @Permissions('lab.docs.commit')
  commit(@Actor() actor: any, @Body() dto: LabCommitDto) {
    return this.svc.commit(actor, dto);
  }

  @Post('/:docId/link')
  @Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN')
  @Permissions('lab.docs.link')
  link(@Actor() actor: any, @Param('docId') docId: string, @Body()
dto: LabLinkDto) {
    return this.svc.link(actor, docId, dto);
  }

  @Post('/:docId/verify')
  @Roles('AUDITOR','ADMIN','MANAGER','TECHNICIAN')
  @Permissions('lab.docs.verify')
  verify(@Actor() actor: any, @Param('docId') docId: string, @Body()
dto: LabVerifyDto) {
    return this.svc.verify(actor, docId, dto);
  }

  @Get()

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDITOR')
  @Permissions('lab.docs.read')
  list(@Actor() actor: any, @Query('propertyId') propertyId: string,
@Query('q') q?: string) {
    return this.svc.list(actor, propertyId, q);
  }
}
```

3.2 Service (idempotência + auditoria + parecer opcional)

```
// src/lab/lab.service.ts
import { Inject, Injectable } from '@nestjs/common';
```

```
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';
import { AuditorOpinionService } from '../audit/auditor-
opinion.service';

function rangeEnd(prefix: string) { return prefix + '\uf8ff'; }

@Injectable()
export class LabService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly opinions: AuditorOpinionService, // opcional p/
parecer hash/cadeia
  ) {}

  async createUploadTarget(actor: any, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'lab.docs.uploadUrl');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const docId = makeId('doc');
      const ext = dto.filename?.toLowerCase().endsWith('.pdf') ?
'.pdf' : '.pdf';
      const storagePath =
`tenants/${tenantId}/lab_docs/${docId}${ext}`;

      tx.create(this.db.collection('lab_docs').doc(docId), {
        tenantId,
        docId,
        propertyId: dto.propertyId,
        docType: 'LAB_REPORT',
        title: dto.filename,
        file: { storagePath, mime: dto.mime, size: 0 },
        hash: { sha256: null },
        targets: [],
        status: 'UPLOADED',
        createdAt: now,
        createdBy: { uid: actor.uid, role: actor.role },
        nameLower: String(dto.filename || '').toLowerCase(),
        verification: { status: 'PENDING', verifiedAt: null,
verifiedBy: null, opinionId: null, reason: null }
      });

      // auditoria
      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        propertyId: dto.propertyId,
        type: 'LAB_DOC_UPLOAD_TARGET_CREATED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { docId, storagePath },
```

```
        ts: now
      });

      // se você usar signed URL, gere aqui (placeholder)
      const uploadUrl = null;

      const out = { docId, storagePath, uploadUrl };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'lab.docs.uploadUrl', clientRequestId: dto.clientRequestId, createdAt:
now, result: out });
      return out;
    });
  }

  async commit(actor: any, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'lab.docs.commit');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const ref = this.db.collection('lab_docs').doc(dto.docId);
      const snap = await tx.get(ref);
      if (!snap.exists) throw new Error('DOC_NOT_FOUND');

      const d = snap.data()!;
      if (d.tenantId !== tenantId) throw new Error('DOC_FORBIDDEN');

      tx.set(ref, {
        title: dto.title,
        docType: dto.docType,
        targets: dto.targets ?? [],
        file: { ...d.file, size: dto.size },
        hash: { sha256: dto.sha256 },
        status: 'LINKED',
        updatedAt: now,
        nameLower: String(dto.title || '').toLowerCase(),
        verification: { ...d.verification, status: 'PENDING' }
      }, { merge: true });

      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        propertyId: d.propertyId,
        type: 'LAB_DOC_COMMITTED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { docId: dto.docId, sha256: dto.sha256, targets:
dto.targets ?? [] },
        ts: now
      });

      const out = { docId: dto.docId, status: 'LINKED' };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'lab.docs.commit', clientRequestId: dto.clientRequestId, createdAt:
now, result: out });
```

```
      return out;
    });
  }

  async link(actor: any, docId: string, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'lab.docs.link');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const docRef = this.db.collection('lab_docs').doc(docId);
      const docSnap = await tx.get(docRef);
      if (!docSnap.exists) throw new Error('DOC_NOT_FOUND');
      const d = docSnap.data()!;
      if (d.tenantId !== tenantId) throw new Error('DOC_FORBIDDEN');

      const linkId = makeId('lnk');
      tx.create(this.db.collection('doc_links').doc(linkId), {
        tenantId,
        linkId,
        docId,
        target: dto.target,
        createdAt: now,
        createdBy: { uid: actor.uid, role: actor.role }
      });

      tx.set(docRef, { status: 'LINKED', updatedAt: now }, { merge:
true });

      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        propertyId: d.propertyId,
        type: 'LAB_DOC_LINKED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { docId, linkId, target: dto.target },
        ts: now
      });

      const out = { linkId, docId, target: dto.target };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'lab.docs.link', clientRequestId: dto.clientRequestId, createdAt: now,
result: out });
      return out;
    });
  }

  async verify(actor: any, docId: string, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'lab.docs.verify');
    const idemRef = this.db.collection('idempotency').doc(key);
```

```
    // regra simples: só AUDITOR/TECHNICIAN/MANAGER etc (já no guard)
    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const docRef = this.db.collection('lab_docs').doc(docId);
      const snap = await tx.get(docRef);
      if (!snap.exists) throw new Error('DOC_NOT_FOUND');
      const d = snap.data()!;
      if (d.tenantId !== tenantId) throw new Error('DOC_FORBIDDEN');

      // opcional: criar parecer imutável com hash/cadeia
      let opinionId = null;
      if (dto.opinionSummary && dto.opinionSummary.trim()) {
        // fora de tx seria melhor; aqui a gente registra referência e
você pode criar o opinion antes
        // Para manter consistente: grave só texto e depois crie
opinion em endpoint separado se preferir.
        opinionId = `opn_${Date.now()}`;
      }

      const vStatus = dto.decision === 'VERIFIED' ? 'VERIFIED' :
'REJECTED';

      tx.set(docRef, {
        status: vStatus,
        verification: {
          status: vStatus,
          verifiedAt: now,
          verifiedBy: { uid: actor.uid, role: actor.role },
          opinionId,
          reason: dto.reason ?? null
        },
        updatedAt: now
      }, { merge: true });

      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        propertyId: d.propertyId,
        type: 'LAB_DOC_VERIFIED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { docId, decision: vStatus, reason: dto.reason ?? null
},
        ts: now
      });

      const out = { docId, verification: { status: vStatus } };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'lab.docs.verify', clientRequestId: dto.clientRequestId, createdAt:
now, result: out });
      return out;
    });
  }

  async list(actor: any, propertyId: string, q?: string) {
    if (!propertyId) throw new Error('PROPERTY_ID_REQUIRED');
    const tenantId = actor.tenantId;
```

```
    const base = this.db.collection('lab_docs')
      .where('tenantId', '==', tenantId)
      .where('propertyId', '==', propertyId);

    if (!q || !q.trim()) {
      const snap = await base.orderBy('nameLower').limit(100).get();
      return { items: snap.docs.map(d => d.data()) };
    }

    const qq = q.trim().toLowerCase();
    const snap = await
base.orderBy('nameLower').startAt(qq).endAt(rangeEnd(qq)).limit(100).g
et();
    return { items: snap.docs.map(d => d.data()) };
  }
}
```

---

4) RulesEngine: exigir laudo (e opcionalmente exigir verificação)

Você já tem `LabDocRequiredRule`. Agora deixo uma versão mais "jurídica":

4.1 Regra: quando limite exige laudo, docEvidenceIds precisa ter docs VERIFICADOS

```
// src/rules/rules/labDocVerifiedRequired.rule.ts
import { Inject } from '@nestjs/common';
import { FIRESTORE } from '../../common/firestore.provider';
import { Rule, RuleContext, RuleResult } from '../rules.types';

export class LabDocVerifiedRequiredRule implements Rule {
  id = 'labDocVerifiedRequired';

  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  applies(ctx: RuleContext) {
    return ctx.operation === 'FARM_APPLICATION';
  }

  async evaluate(ctx: RuleContext): Promise<RuleResult> {
    const { docEvidenceIds, applicationType } = ctx.payload;

    // Se não tem docs, outra regra já bloqueia (LAB_DOC_REQUIRED).
Aqui valida "verificado".
    if (!Array.isArray(docEvidenceIds) || docEvidenceIds.length === 0)
return { ok: true };

    // Exemplo: só exigir verificação para PESTICIDE e FERTILIZER
    const needsVerified =
['PESTICIDE','FERTILIZER'].includes(applicationType);
    if (!needsVerified) return { ok: true };

    // checa docs
    for (const docId of docEvidenceIds) {
      const snap = await
this.db.collection('lab_docs').doc(docId).get();
      if (!snap.exists) {
```

```
        return { ok:false, code:'LAB_DOC_NOT_FOUND',
message:'Documento de laudo não encontrado.', details:{ docId } };
      }
      const d = snap.data()!;
      if (d.tenantId !== ctx.tenantId) {
        return { ok:false, code:'LAB_DOC_FORBIDDEN',
message:'Documento não pertence ao tenant.', details:{ docId } };
      }
      if (d.verification?.status !== 'VERIFIED') {
        return { ok:false, code:'LAB_DOC_NOT_VERIFIED', message:'Laudo
precisa estar validado (AUDITOR/TECHNICIAN).', details:{ docId,
status: d.verification?.status } };
      }
    }

    return { ok: true };
  }
}
```

Coloque essa regra depois da `LabDocRequiredRule` no `RulesModule`.

---

5) Fluxo mobile (bem simples)

Upload/Commit (no app)

1. usuário escolhe PDF (laudo)
2. calcula SHA256 (ou deixa backend calcular depois)
3. chama `POST /api/lab/docs/upload-url` → recebe `docId` + `storagePath`
4. faz upload do PDF para `storagePath` (Firebase Storage SDK)
5. chama `POST /api/lab/docs/commit` com `sha256`, `title`, `targets`
6. (opcional) chama `/link` para vincular à aplicação/contrato

Auditoria

- AUDITOR abre lista `GET /api/lab/docs?propertyId=...`
- toca em um doc → `verify` (VERIFIED/REJECTED) com motivo

✅ A aplicação de insumo só passa se:

- doc anexado (quando exigido)
- doc verificado (quando regra pedir)

Se você quiser, o próximo bloco que fecha **cadeia completa agro** é:
✅ **Recebimento de leite (mensal) com preço variável + contrato por demanda + escrow por etapas + proteção contra calote** (trava por CAR/IE do comprador e travas em ativos cadastrados).

Perfeito — vamos fechar o módulo **Leite (preço variável) + contrato por demanda + proteção anti-calote**, exatamente como você descreveu:

- produtor entrega **30 dias**, recebe **30 dias depois**
- indústria retém valor → risco de calote
- volume pode oscilar (variável)
- precisa **segurança jurídica**: garantia do comprador + penhora/travas em ativos + bloqueios por CAR/IE
- pagamento em **escrow por etapas** (marcos verificáveis)
- se não cumprir → trava e bloqueia comercialização no sistema (e pode gerar parecer/auditoria)

Abaixo vai o desenho técnico completo (Firestore + endpoints + regras + fluxo).

---

1) Conceito: Contrato "variável" por janela + preço indexado + ajuste

O contrato não "adianta tudo". Ele cria um **mecanismo de liquidação**:

- **Janela**: ex. 01–30 do mês (entregas)
- **Prazo de pagamento**: D+30 (após fechamento)
- **Preço**: pode ser:
  - **Indexado** (ex.: preço referência regional + fator de qualidade)
  - **Faixa com teto/piso** (min/max)
  - **Fórmula** (ex.: média ponderada + bonificações)
- **Volume**: variável; escore e garantia devem cobrir **pior caso** (limite máximo contratado)

A indústria não "adianta volume desconhecido"; ela deposita **garantia mínima** e o restante vai sendo ajustado conforme o leite entra e é validado.

---

2) Firestore: modelos

2.1 Contrato leite

```
milk_contracts/{contractId}

{
  "tenantId":"TENANT_PRODUCER",
  "contractId":"mlk_ctr_001",
  "producerTenantId":"TENANT_PRODUCER",
  "buyerTenantId":"TENANT_INDUSTRY",
  "propertyId":"prop_123",

  "cycle": {
    "windowType":"MONTHLY",
    "paymentDelayDays":30,
    "minVolumeLiters": 10000,
    "maxVolumeLiters": 30000
  },
```

```
  "priceModel": {
    "type":"INDEXED",
    "index":"REGIONAL_REFERENCE",
    "baseCentsPerLiter": 240,
    "qualityAdjustments": {
      "fatBonusCents": 10,
      "proteinBonusCents": 10,
      "somaticPenaltyCents": -15
    },
    "floorCentsPerLiter": 210,
    "capCentsPerLiter": 320
  },

  "escrow": {
    "required": true,
    "guaranteeMinCents": 5000000,
    "coverageMode":"MAX_VOLUME_CAP",
    "penaltyCancelPercent": 5
  },

  "buyerGuarantees": {
    "lockedAssets": [
      { "type":"REGISTERED_ASSET", "assetId":"ast_001" }
    ],
    "blockBy": { "car": true, "ie": true }
  },

  "state":"ACTIVE",
  "createdAt":1730000000000
}
```

2.2 Lote de recebimentos (mensal)

```
milk_cycles/{cycleId}

{
  "tenantId":"TENANT_PRODUCER",
  "cycleId":"mlk_cyc_2026_01",
  "contractId":"mlk_ctr_001",
  "period": { "from":"2026-01-01", "to":"2026-01-30" },
  "status":"OPEN|CLOSED|INVOICED|PAID",

  "totals": {
    "deliveredLiters": 0,
    "estimatedAmountCents": 0,
    "finalAmountCents": 0
  },

  "escrow": {
    "status":"NOT_CREATED|HELD|PARTIAL_RELEASED|RELEASED|DEFAULTED",
    "escrowId": null
  },

  "createdAt":1730000000000
}
```

2.3 Entrega diária (evento com evidência)

```
milk_deliveries/{deliveryId}

{
```

```
  "tenantId":"TENANT_PRODUCER",
  "deliveryId":"mlk_del_001",
  "contractId":"mlk_ctr_001",
  "cycleId":"mlk_cyc_2026_01",
  "date":"2026-01-05",

  "liters": 950,
  "quality": { "fat": 3.6, "protein": 3.2, "somatic": 250000 },

  "evidence": {
    "riskType":"B",
    "evidenceIds":["evg_photo_scale","evg_gps"],
    "docIds":["doc_lab_..."]
  },

  "status":"RECEIVED|DISPUTED|CONFIRMED",
  "createdAt":1730000000000
}
```

2.4 Garantias/travas do comprador (anti-calote)

```
buyer_guarantee_locks/{lockId}

{
  "tenantId":"TENANT_INDUSTRY",
  "lockId":"lck_001",
  "buyerTenantId":"TENANT_INDUSTRY",
  "contractId":"mlk_ctr_001",

  "lockType":"ASSET_LOCK|COMMERCIAL_BLOCK",
  "ref": { "type":"CAR", "value":"CAR-XYZ" },

  "state":"ACTIVE",
  "reason":"Escrow obrigatório e risco de calote.",
  "createdAt":1730000000000
}
```

Isso implementa o que você quer: **bloquear comercialização dentro do sistema** se não pagar.

---

3) Endpoints

3.1 Criar contrato

```
POST /api/milk/contracts
```

3.2 Abrir ciclo mensal (auto ou manual)

```
POST /api/milk/contracts/:id/cycles/open
```

3.3 Registrar entrega (com evidência)

```
POST /api/milk/deliveries
```

3.4 Fechar ciclo (calcula valor final e gera cobrança escrow)

`POST /api/milk/cycles/:id/close`

3.5 Confirmar pagamento escrow (webhook) → libera split

`POST /api/webhooks/asaas` (reaproveita)

3.6 Ações anti-calote

- `POST /api/milk/cycles/:id/default` (gera bloqueios e travas)
- `POST /api/milk/buyer/:tenantId/unblock` (somente auditor/admin após regularização)

---

4) Regra de cálculo (volume variável)

Estimativa diária

```
estimatedAmount = liters * clamp(indexPriceAdjusted, floor, cap)
```

Ajuste mensal

No fechamento do ciclo:

- soma liters
- calcula preço final (pode usar índice do período + ajustes)
- `finalAmount = sum(deliveryFinalAmounts)`
- compara com escrow:
  - se escrow < finalAmount → exige complemento
  - se escrow > finalAmount → devolve excedente (ou mantém como garantia futura)

✅ Assim a indústria não precisa adiantar "o desconhecido", mas **não consegue dar calote** porque:

- o sistema só fecha e libera NF/transferências dela se escrow/garantia estiver ok
- e se não pagar, ela fica travada (CAR/IE/ativos) dentro do ecossistema

---

5) Regras (RulesEngine) — o núcleo jurídico/técnico

5.1 `MilkEscrowRequiredRule`

- cada ciclo precisa de escrow "HELD" para permitir:
  - emissão de NF do comprador dentro do Ciclo+
  - movimentação/comercialização relevante

    - retirada de produto/ativos relacionados

5.2 `BuyerCommercialBlockRule`

- se ciclo em `DEFAULTED`, cria `buyer_guarantee_locks`:
    - bloqueia comercialização no sistema
    - bloqueia transferência de ativos cadastrados
    - bloqueia novos contratos com aquele comprador

5.3 `EvidenceRequiredForMilkDeliveryRule`

- entrega exige:
    - foto do medidor/volume
    - GPS/perímetro (geofence)
    - (opcional) doc laboratório mensal/por lote

---

6) Implementação (NestJS) — endpoints principais

A seguir coloco **3 endpoints essenciais** (entrega, fechar ciclo, default). Os demais seguem mesmo padrão.

6.1 `POST /api/milk/deliveries`

```
// src/milk/milk-deliveries.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';
import { RulesEngine } from '../rules/rules.engine';

@Injectable()
export class MilkDeliveriesService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly rules: RulesEngine,
  ) {}

  async create(actor: any, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const ctx = { actor, tenantId, operation: 'MILK_DELIVERY',
propertyId: dto.propertyId, payload: dto };
    const ruleRes = await this.rules.run(ctx);
    if (!ruleRes.ok) throw new
Error(`BLOCKED_${ruleRes.violations[0].code}`);

    const deliveryId = makeId('mlk_del');
    const ref = this.db.collection('milk_deliveries').doc(deliveryId);

    await ref.set({
      tenantId,
      deliveryId,
```

```
      contractId: dto.contractId,
      cycleId: dto.cycleId,
      propertyId: dto.propertyId,
      date: dto.date,
      liters: dto.liters,
      quality: dto.quality ?? null,
      evidence: dto.evidence ?? null,
      status: 'RECEIVED',
      createdAt: now,
      createdBy: { uid: actor.uid, role: actor.role }
    });

    return { deliveryId, status: 'RECEIVED' };
  }
}
```

6.2 `POST /api/milk/cycles/:id/close` (calcula + exige escrow)

```
// src/milk/milk-cycles.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';
import { AsaasService } from '../integrations/asaas/asaas.service';

function clamp(n: number, a: number, b: number) { return Math.max(a,
Math.min(b, n)); }

@Injectable()
export class MilkCyclesService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly asaas: AsaasService,
  ) {}

  async closeCycle(actor: any, cycleId: string) {
    const now = Date.now();

    const cycleRef = this.db.collection('milk_cycles').doc(cycleId);
    const cycleSnap = await cycleRef.get();
    if (!cycleSnap.exists) throw new Error('CYCLE_NOT_FOUND');

    const cyc = cycleSnap.data()!;
    if (cyc.tenantId !== actor.tenantId) throw new
Error('CYCLE_FORBIDDEN');

    // carregar contrato
    const ctrSnap = await
this.db.collection('milk_contracts').doc(cyc.contractId).get();
    if (!ctrSnap.exists) throw new Error('CONTRACT_NOT_FOUND');
    const ctr = ctrSnap.data()!;

    // carregar entregas do ciclo
    const delSnap = await this.db.collection('milk_deliveries')
      .where('tenantId', '==', actor.tenantId)
      .where('cycleId', '==', cycleId)
      .get();

    let litersTotal = 0;
    let finalAmount = 0;

    for (const doc of delSnap.docs) {
      const d = doc.data();
```

```
      litersTotal += Number(d.liters || 0);

      // preço simples: base + ajustes (exemplo)
      const base = Number(ctr.priceModel.baseCentsPerLiter || 0);
      const floor = Number(ctr.priceModel.floorCentsPerLiter || 0);
      const cap = Number(ctr.priceModel.capCentsPerLiter || 999999);

      let adj = base;
      // aqui você aplicaria índice regional e bonificações reais
      adj = clamp(adj, floor, cap);

      finalAmount += Math.round(adj * Number(d.liters || 0));
    }

    // define escrow mínimo a exigir: max(garantia mínima, valor
final)
    const minGuarantee = Number(ctr.escrow.guaranteeMinCents || 0);
    const required = Math.max(minGuarantee, finalAmount);

    // cria cobrança escrow (request) - pode ser imediato via Asaas
    const escrowId = makeId('esc');
    await this.db.collection('milk_escrows').doc(escrowId).set({
      tenantId: actor.tenantId,
      escrowId,
      contractId: cyc.contractId,
      cycleId,
      buyerTenantId: ctr.buyerTenantId,
      amountCents: required,
      status: 'REQUESTED',
      createdAt: now
    });

    await cycleRef.set({
      status: 'CLOSED',
      totals: { deliveredLiters: litersTotal, finalAmountCents:
finalAmount, estimatedAmountCents: finalAmount },
      escrow: { status: 'NOT_CREATED', escrowId },
      closedAt: now
    }, { merge: true });

    return {
      cycleId,
      status: 'CLOSED',
      deliveredLiters: litersTotal,
      finalAmountCents: finalAmount,
      escrow: { escrowId, requiredCents: required, status: 'REQUESTED'
}
    };
  }
}
```

6.3 `POST /api/milk/cycles/:id/default` (calote → trava comprador)

```
// src/milk/milk-default.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class MilkDefaultService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}
```

```
  async markDefault(actor: any, cycleId: string, reason?: string) {
    const now = Date.now();

    const cycleRef = this.db.collection('milk_cycles').doc(cycleId);
    const cycleSnap = await cycleRef.get();
    if (!cycleSnap.exists) throw new Error('CYCLE_NOT_FOUND');

    const cyc = cycleSnap.data()!;
    if (cyc.tenantId !== actor.tenantId) throw new
Error('CYCLE_FORBIDDEN');

    const ctrSnap = await
this.db.collection('milk_contracts').doc(cyc.contractId).get();
    const ctr = ctrSnap.data()!;
    const buyerTenantId = ctr.buyerTenantId;

    // cria bloqueio comercial do comprador (no sistema)
    const lockId = makeId('lck');
    await
this.db.collection('buyer_guarantee_locks').doc(lockId).set({
      tenantId: buyerTenantId,
      lockId,
      buyerTenantId,
      contractId: cyc.contractId,
      lockType: 'COMMERCIAL_BLOCK',
      ref: { type: 'TENANT', value: buyerTenantId },
      state: 'ACTIVE',
      reason: reason ?? 'Inadimplência em ciclo de leite (escrow não
pago).',
      createdAt: now
    });

    // marca ciclo como DEFAULTED
    await cycleRef.set({
      status: 'CLOSED',
      escrow: { ...cyc.escrow, status: 'DEFAULTED' },
      defaultedAt: now,
      defaultReason: reason ?? null
    }, { merge: true });

    return { cycleId, status: 'DEFAULTED', buyerTenantId, lockId };
  }
}
```

---

7) Travar “emissão de notas / comercialização” do comprador no Ciclo+

Você já tem o conceito de bloqueio por **CAR/IE**. Para a indústria:

- em todas operações críticas (NF, venda, transferências, contratos novos):
  - `BuyerCommercialBlockRule` consulta `buyer_guarantee_locks` do tenant comprador
  - se `ACTIVE` → bloqueia com motivo e link para regularização

Isso resolve “calote recorrente” dentro do ecossistema.

---

8) UI simples (produtor)

- “Registrar entrega” (um botão por dia)
- informar litros (ou ler do medidor)
- foto + GPS
- anexar laudo (mensal)
- “Fechar mês”
- “Aguardar escrow/pagamento”

UI indústria:

- “Pagar escrow do mês”
- “Ver entregas e laudos”
- “Aceitar/contestar”

Se você quiser, eu fecho o último ponto para ficar **à prova de fraude**:
✅ **Disputa**: indústria contesta entrega → entra em estado `DISPUTED` e exige vistoria de TECHNICIAN/AUDITOR (read-only) antes de liquidar.

Sim — dá pra fazer isso **bem encaixado no Ciclo+** e com “lastro” forte:

- **Grupo regional de produtores** (pool) para garantir volume mínimo pra indústria.
- **Indústria paga em escrow** por **coleta** (cada coleta gera split automático).
- **Produtor lança a entrega** (autonomia) com evidência no tanque (foto/vídeo + GPS + leitura).
- A cada entrega/coleta nasce um **“Ticket de Recebimento”** (direito de crédito) que fica **auditável, imutável e rastreável** até o dinheiro cair (na próxima coleta ou no D+30).

Abaixo vai o desenho técnico completo (modelos + regras + endpoints + estados).

---

1) Conceito operacional

Dois fluxos possíveis

1. **Coleta-a-coleta (recomendado)**
   - A cada coleta: cria cobrança escrow, trava valor e já prepara split por produtor.
   - Quando a coleta é validada (evidência + auditoria), o sistema libera o pagamento **da coleta anterior** automaticamente (ou por marcos).
2. **Mensal (fechamento)**
   - Mantém o seu modelo D+30, mas com tickets diários e garantia mínima.

Você pediu “ticket até receber dinheiro na próxima coleta”: isso é exatamente o **Fluxo 1** com “janela rolante”.

---

2) Modelos Firestore

2.1 Grupo regional (pool)

```
milk_pools/{poolId}

{
  "tenantId":"TENANT_PLATFORM",
  "poolId":"pool_mg_regiao_01",
  "name":"Grupo Leite Região X",
  "region": { "lat": -18.9, "lng": -48.2, "radiusKm": 120 },
  "members":[
    { "producerTenantId":"TENANT_P1", "propertyId":"prop_p1",
"quotaPercent": 20 },
    { "producerTenantId":"TENANT_P2", "propertyId":"prop_p2",
"quotaPercent": 30 }
  ],
  "buyerTenantId":"TENANT_INDUSTRY",
  "state":"ACTIVE",
  "createdAt":1730000000000
}
```

2.2 Contrato do pool com a indústria

```
milk_pool_contracts/{poolContractId}

{
  "tenantId":"TENANT_PLATFORM",
  "poolContractId":"pc_001",
  "poolId":"pool_mg_regiao_01",
  "buyerTenantId":"TENANT_INDUSTRY",
  "priceModel": {
    "type":"INDEXED",
    "baseCentsPerLiter": 240,
    "floorCentsPerLiter": 210,
    "capCentsPerLiter": 320
  },
  "collectionPolicy":{
    "requiresTankAudit": true,
    "requiresGpsGeofence": true,
    "maxHoursToSubmitByProducer": 24
  },
  "escrowPolicy":{
    "mode":"PER_COLLECTION_ROLLING",
    "holdBackPercent": 5,
    "penaltyCancelPercent": 5
  },
  "state":"ACTIVE"
}
```

2.3 Coleta (evento do caminhão/rota)

```
milk_collections/{collectionId}

{
  "tenantId":"TENANT_PLATFORM",
  "collectionId":"col_2026_02_05_route_01",
  "poolContractId":"pc_001",
  "buyerTenantId":"TENANT_INDUSTRY",
  "route": { "vehicleId":"trk_01", "driverUserId":"u_123" },
  "status":"OPEN|LOCKED|PAID",
  "createdAt":1730000000000
}
```

2.4 Entrega lançada pelo produtor (na coleta)

```
milk_collection_deliveries/{deliveryId}

{
  "tenantId":"TENANT_PLATFORM",
  "deliveryId":"dly_001",
  "collectionId":"col_2026_02_05_route_01",
  "poolId":"pool_mg_regiao_01",
  "producerTenantId":"TENANT_P1",
  "propertyId":"prop_p1",
  "liters": 820,
  "quality": { "fat": 3.6, "protein": 3.2, "somatic": 250000 },

  "tankAudit": {
    "meterReading": "foto/leitura",
    "temperature": 4.2,
    "sampleTaken": true
  },

  "evidence": {
    "riskType":"B",
    "evidenceIds":["evg_photo_tank","evg_gps","evg_video"],
    "docIds":["doc_lab_optional"]
  },

  "status":"SUBMITTED|VERIFIED|DISPUTED|REJECTED",
  "createdAt":1730000000000
}
```

2.5 Ticket (direito de receber)

```
milk_tickets/{ticketId}

{
  "tenantId":"TENANT_PLATFORM",
  "ticketId":"tkt_001",
  "deliveryId":"dly_001",
  "collectionId":"col_2026_02_05_route_01",
  "producerTenantId":"TENANT_P1",
  "amountCents": 196800,
  "status":"ISSUED|PLEDGED|SETTLED|CANCELLED",
  "settlement": {
    "escrowId":"esc_001",
    "expectedReleaseAt":1730000000000
  },
  "createdAt":1730000000000
```

```
}
```

2.6 Escrow por coleta + split

```
milk_escrows/{escrowId}

{
  "tenantId":"TENANT_PLATFORM",
  "escrowId":"esc_001",
  "collectionId":"col_2026_02_05_route_01",
  "buyerTenantId":"TENANT_INDUSTRY",
  "status":"REQUESTED|HELD|RELEASED|DEFAULTED",
  "amountCents": 12000000,
  "splitPlan": {
    "targets":[
      { "producerTenantId":"TENANT_P1", "amountCents": 196800,
"ticketId":"tkt_001" },
      { "producerTenantId":"TENANT_P2", "amountCents": 250000,
"ticketId":"tkt_002" }
    ]
  }
}
```

---

3) Estados e regras principais

3.1 Máquina de estados

- Delivery: `SUBMITTED → VERIFIED → (gera TICKET) → SETTLED`
- Collection: `OPEN → LOCKED (após validações) → PAID (escrow liberado)`
- Ticket: `ISSUED → SETTLED`

3.2 Regras (RulesEngine)

1. **Evidência obrigatória**: foto do tanque + GPS + leitura/temperatura (Tipo B)
2. **Geofence**: só aceita entrega se estiver dentro do perímetro da propriedade
3. **Janela**: produtor pode lançar até X horas após coleta (anti fraude)
4. **Escrow obrigatório por coleta**: indústria só "fecha" coleta se escrow estiver HELD
5. **Disputa**: se indústria contestar, ticket não liquida até vistoria (AUDITOR/TECHNICIAN)

---

4) Endpoints (mínimos para funcionar)

Produtor

1. `POST /api/milk/pools/:poolId/collections/:collectionId/deliveries`

- cria `milk_collection_deliveries` com evidência

- status `SUBMITTED`

2. `GET /api/milk/tickets?producerTenantId=...&status=ISSUED`

- lista tickets a receber

Indústria / Logística

3. `POST /api/milk/collections/:collectionId/lock`

- valida entregas verificadas
- calcula valores e splitPlan
- cria escrow "REQUESTED"

4. `POST /api/milk/collections/:collectionId/escrow/create`

- cria cobrança Asaas e coloca `HELD` quando pago

5. Webhook Asaas → `HELD` → executa split → marca tickets `SETTLED` → collection `PAID`

Auditor/Disputa

6. `POST /api/milk/deliveries/:deliveryId/dispute`
7. `POST /api/milk/deliveries/:deliveryId/verify` (TECHNICIAN/AUDITOR)

---

5) Como "pagar na próxima coleta" sem risco de volume variável

A sacada é o **rolling escrow**:

- Coleta N: indústria deposita escrow da **Coleta N-1** (ou "saldo devido") + garantia mínima.
- Quando as entregas N são verificadas, o sistema:
  - emite tickets N (ISSUED)
  - libera pagamento dos tickets N-1 (SETTLED)

Isso cria um ciclo contínuo:

- produtor sempre tem tickets auditáveis
- indústria sempre está "coberta" por garantia
- se parar de pagar → trava imediato (bloqueio comercial e de ativos)

---

6) Anti-calote "forte" dentro do sistema

Quando escrow não é pago no prazo:

- `buyer_guarantee_locks` é criado para o comprador:
  - bloqueia emissão NF dentro do sistema
  - bloqueia novos contratos
  - bloqueia movimentações de ativos cadastrados (no ecossistema Ciclo+)
  - tudo com auditoria append-only (lastro)

---

7) UI simples pro produtor (sem leitura difícil)

- Botão grande: **"Lançar entrega"**
- Passos:
  1. Foto do tanque/medidor
  2. Capturar GPS
  3. Informar litros (ou voz: "oitocentos e vinte litros")
  4. Confirmar
- Resultado: "Ticket emitido: R$ X (pagar na próxima coleta)"

---

8) Quer que eu já te entregue o código NestJS desses endpoints também?

Se você disser "sim" (de novo), eu mando **agora**:

- `POST deliveries` (produtor)
- `POST collection lock` (indústria)
- executor escrow/split/tickets (webhook)
- `verify/dispute` (auditor)
- e as regras (geofence + janela + escrow obrigatório)

Tudo no mesmo padrão que você já está usando (multi-tenant + idempotência + audit_events).

Perfeito — isso encaixa como um papel **CONFERENTE** (pode ser `TECHNICIAN` ou `OPERATOR` credenciado) e vira um **evento diário obrigatório**: *Tank Check* → gera *Ticket Diário*.

A ideia fica assim:

- **Todo dia** existe um "Conferente do tanque" (presencial ou credenciado) que:
  - mede volume (litros)
  - registra temperatura
  - tira foto/vídeo do medidor + GPS (geofence)
  - (opcional) coleta amostra
- O sistema cria um **TICKET DIÁRIO** (direito de crédito) imediatamente
- Na **coleta do caminhão**, os tickets diários do período entram no fechamento e pagamento (escrow + split)

Ou seja: **o produtor não depende do caminhão** para ter ticket — ele tem ticket diário com lastro.

---

1) Ajuste de papéis

Adicionar role/permissão:

- `INSPECTOR` (Conferente) **ou** usar `TECHNICIAN` com permissão específica

Permissões novas:

- `milk.tankCheck.create`
- `milk.tankCheck.verify` (se quiser dupla validação)
- `milk.ticket.issue`

---

2) Modelos Firestore

2.1 Tank Check diário (evento auditável)

```
milk_tank_checks/{checkId}

{
  "tenantId":"TENANT_PLATFORM",
  "checkId":"chk_2026_02_05_prop_p1",
  "poolId":"pool_mg_regiao_01",
  "producerTenantId":"TENANT_P1",
  "propertyId":"prop_p1",
  "date":"2026-02-05",

  "reading": {
    "liters": 820,
    "temperature": 4.2,
    "meterType":"RULER|DIGITAL|FLOW",
    "meterValue":"82cm"
  },

  "evidence": {
    "riskType":"B",
    "evidenceIds":["evg_photo_meter","evg_gps","evg_video_optional"]
  },

  "status":"RECORDED|DISPUTED|CONFIRMED",
  "recordedBy": { "uid":"u_conf_01", "role":"INSPECTOR" },
  "recordedAt":1730000000000
}
```

2.2 Ticket diário (direito do produtor)

```
milk_daily_tickets/{ticketId}

{
  "tenantId":"TENANT_PLATFORM",
  "ticketId":"tktd_2026_02_05_p1",
  "checkId":"chk_2026_02_05_prop_p1",
  "poolId":"pool_mg_regiao_01",
```

```
  "producerTenantId":"TENANT_P1",
  "propertyId":"prop_p1",
  "date":"2026-02-05",
  "liters": 820,

  "pricing": {
    "centsPerLiter": 240,
    "amountCents": 196800,
    "model":"INDEXED/FLOOR/CAP",
    "isEstimated": true
  },

  "status":"ISSUED|SETTLED|CANCELLED",
  "settlement": { "collectionId": null, "escrowId": null },
  "createdAt":1730000000000
}
```

isEstimated=true porque o valor final pode ajustar no fechamento (qualidade/índice). Mas o ticket já dá **direito de receber** e serve como prova.

---

3) Regras obrigatórias (RulesEngine)

3.1 `TankCheckEvidenceRule`

Bloqueia se:

- sem foto do medidor
- sem GPS (ou fora do geofence)
- sem temperatura (se exigido)

3.2 `OneCheckPerDayRule`

Um check por propriedade/dia:

- impede duplicidade/fraude

3.3 `InspectorRequiredRule`

Somente usuário com permissão `milk.tankCheck.create` pode registrar.

3.4 `TicketIssuanceRule`

Ao registrar check:

- emite ticket diário automaticamente
- registra auditoria append-only

---

4) Endpoints (essenciais)

4.1 Registrar conferência diária

```
POST /api/milk/tank-checks
```

Body:

```
{
  "clientRequestId":"uuid",
  "poolId":"pool_mg_regiao_01",
  "producerTenantId":"TENANT_P1",
  "propertyId":"prop_p1",
  "date":"2026-02-05",
  "reading": { "liters":820, "temperature":4.2, "meterType":"RULER",
"meterValue":"82cm" },
  "evidence": { "evidenceIds":["evg_photo_meter","evg_gps"] }
}
```

Resposta:

```
{ "checkId":"chk_...", "ticketId":"tktd_...", "amountCents":196800,
"status":"ISSUED" }
```

4.2 Listar tickets do produtor

```
GET /api/milk/daily-tickets?producerTenantId=...&status=ISSUED
```

4.3 Disputar/confirmar (auditor/indústria)

- `POST /api/milk/tank-checks/:id/dispute`
- `POST /api/milk/tank-checks/:id/confirm`

---

5) Código NestJS (registrar tank check + emitir ticket)

Service (transação: cria check + cria ticket + auditoria)

```
// src/milk/milk-tank-checks.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';
import { idemKey } from '../common/idempotency/idempotency.util';
import { RulesEngine } from '../rules/rules.engine';

function clamp(n: number, a: number, b: number) { return Math.max(a,
Math.min(b, n)); }

@Injectable()
export class MilkTankChecksService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly rules: RulesEngine,
  ) {}

  async create(actor: any, dto: any) {
    const now = Date.now();
```

```
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'milk.tankCheck.create');
    const idemRef = this.db.collection('idempotency').doc(key);

    const ctx = { actor, tenantId, operation: 'MILK_TANK_CHECK',
propertyId: dto.propertyId, payload: dto };
    const ruleRes = await this.rules.run(ctx);
    if (!ruleRes.ok) throw new
Error(`BLOCKED_${ruleRes.violations[0].code}`);

    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      // 1) garantir 1 check por dia/propriedade
      // chave determinística para evitar duplicidade:
      const checkId = `chk_${dto.date}_${dto.propertyId}`;
      const checkRef =
this.db.collection('milk_tank_checks').doc(checkId);
      const checkSnap = await tx.get(checkRef);
      if (checkSnap.exists) throw new Error('CHECK_ALREADY_EXISTS');

      tx.create(checkRef, {
        tenantId,
        checkId,
        poolId: dto.poolId,
        producerTenantId: dto.producerTenantId,
        propertyId: dto.propertyId,
        date: dto.date,
        reading: dto.reading,
        evidence: { riskType: 'B', evidenceIds:
dto.evidence?.evidenceIds ?? [] },
        status: 'RECORDED',
        recordedBy: { uid: actor.uid, role: actor.role },
        recordedAt: now
      });

      // 2) precificação estimada (piso/teto) — usa contrato do pool
      const pcSnap = await
tx.get(this.db.collection('milk_pool_contracts').doc(dto.poolContractI
d));
      if (!pcSnap.exists) throw new Error('POOL_CONTRACT_NOT_FOUND');
      const pc = pcSnap.data()!;
      const base = Number(pc.priceModel.baseCentsPerLiter || 0);
      const floor = Number(pc.priceModel.floorCentsPerLiter || 0);
      const cap = Number(pc.priceModel.capCentsPerLiter || 999999);
      const centsPerLiter = clamp(base, floor, cap);

      const liters = Number(dto.reading?.liters || 0);
      const amountCents = Math.round(centsPerLiter * liters);

      // 3) emitir ticket diário
      const ticketId = `tktd_${dto.date}_${dto.propertyId}`;
      const ticketRef =
this.db.collection('milk_daily_tickets').doc(ticketId);

      tx.create(ticketRef, {
        tenantId,
        ticketId,
```

```
        checkId,
        poolId: dto.poolId,
        producerTenantId: dto.producerTenantId,
        propertyId: dto.propertyId,
        date: dto.date,
        liters,
        pricing: {
          centsPerLiter,
          amountCents,
          model: pc.priceModel.type ?? 'INDEXED',
          isEstimated: true
        },
        status: 'ISSUED',
        settlement: { collectionId: null, escrowId: null },
        createdAt: now
      });

      // 4) auditoria
      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        propertyId: dto.propertyId,
        type: 'MILK_TANK_CHECK_RECORDED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { checkId, ticketId, liters, amountCents },
        ts: now
      });

      const out = { checkId, ticketId, amountCents, status: 'ISSUED'
};
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'milk.tankCheck.create', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });
      return out;
    });
  }
}
```

Controller

```
// src/milk/milk-tank-checks.controller.ts
import { Body, Controller, Post, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { MilkTankChecksService } from './milk-tank-checks.service';

@Controller('/api/milk/tank-checks')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class MilkTankChecksController {
  constructor(private readonly svc: MilkTankChecksService) {}

  @Post()
  @Roles('ADMIN','MANAGER','TECHNICIAN','OPERATOR','AUDITOR') //
conferente credenciado entra aqui
  @Permissions('milk.tankCheck.create')
  async create(@Actor() actor: any, @Body() dto: any) {
```

```
    return this.svc.create(actor, dto);
  }
}
```

---

6) Como isso “vira pagamento na próxima coleta”

Na coleta do caminhão (collection):

- o motorista/LOGISTICS confirma volumes coletados
- o sistema “agrega” tickets diários desde a última coleta
- cria `milk_escrows` com split por produtor **com base nos tickets**
- ao pagar o escrow (Asaas webhook):
  - marca tickets como `SETTLED`
  - registra auditoria + split

✅ O produtor passa a ter:

- **ticket diário** (direito registrado)
- **pagamento garantido** no próximo ciclo de coleta (se a indústria quiser continuar operando, ela precisa manter escrow em dia)

---

7) O que falta para fechar 100% (eu já posso mandar também)

Se você disser “sim”, eu mando o próximo bloco completo:

1. `POST /api/milk/collections/:id/settle-from-tickets`
   - puxa tickets ISSUED, gera escrow e splitPlan
2. Webhook Asaas → `executeSplitForTickets()`
   - liquida tickets e grava histórico
3. UI produtor:
   - “Tickets a receber”
   - “Pago / pendente”
   - botão grande “Registrar tanque (conferente)”

Quer que eu gere esses 3 agora?

Aí vai o bloco completo com os **3** que fecham 100%:

1. `POST /api/milk/collections/:id/settle-from-tickets` (indústria/logística fecha coleta usando tickets diários)
2. Webhook Asaas → `executeSplitForTickets()` (liquida tickets e grava auditoria)
3. UI do produtor: “Tickets a receber / pagos” + “Registrar tanque”

---

1) POST /api/milk/collections/:id/settle-from-tickets

Objetivo

- Pega todos os `milk_daily_tickets` **ISSUED** dos produtores do pool (até uma data)
- Calcula total
- Cria `milk_escrows` com `splitPlan` por produtor
- Trava a collection (`LOCKED`)
- Deixa pronto para pagar (escrow REQUESTED → HELD)

DTO

```
// src/milk/dto/milk-settle-from-tickets.dto.ts
import { IsNotEmpty, IsOptional, IsString } from 'class-validator';

export class MilkSettleFromTicketsDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsString() @IsNotEmpty()
  poolId: string;

  @IsString() @IsNotEmpty()
  poolContractId: string;

  // liquida tickets até esta data (inclusive) — ex: "2026-02-05"
  @IsString() @IsNotEmpty()
  upToDate: string;

  @IsOptional()
  holdBackPercent?: number; // ex: 5 para segurar qualidade/disputa
}
```

Service

```
// src/milk/milk-collections-settlement.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';
import { idemKey } from '../common/idempotency/idempotency.util';

@Injectable()
export class MilkCollectionsSettlementService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async settleFromTickets(actor: any, collectionId: string, dto: any)
{
    const now = Date.now();
    const tenantId = actor.tenantId;

    // aqui: normalmente tenantId seria o tenant "platform" ou
"industry", depende do seu modelo.
    // O importante é garantir que só buyer/logistics autorizado
execute.

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'milk.collection.settleFromTickets');
    const idemRef = this.db.collection('idempotency').doc(key);
```

```
    return this.db.runTransaction(async (tx) => {
      const idemSnap = await tx.get(idemRef);
      if (idemSnap.exists) return idemSnap.data()!.result;

      const colRef =
this.db.collection('milk_collections').doc(collectionId);
      const colSnap = await tx.get(colRef);
      if (!colSnap.exists) throw new Error('COLLECTION_NOT_FOUND');

      const col = colSnap.data()!;
      if (col.status !== 'OPEN') throw new
Error('COLLECTION_NOT_OPEN');

      // carregar pool contract p/ holdback default
      const pcRef =
this.db.collection('milk_pool_contracts').doc(dto.poolContractId);
      const pcSnap = await tx.get(pcRef);
      if (!pcSnap.exists) throw new Error('POOL_CONTRACT_NOT_FOUND');
      const pc = pcSnap.data()!;

      const holdBackPercent = (dto.holdBackPercent ??
pc.escrowPolicy?.holdBackPercent ?? 0);
      const buyerTenantId = pc.buyerTenantId;

      // buscar tickets ISSUED até upToDate para os membros do pool
      // OBS: Firestore tem limitações de query. Estratégia boa:
      // - manter collection "milk_daily_tickets" no tenant platform
      // - indexar por poolId + status + date
      const ticketSnap = await
this.db.collection('milk_daily_tickets')
        .where('poolId', '==', dto.poolId)
        .where('status', '==', 'ISSUED')
        .where('date', '<=', dto.upToDate)
        .limit(5000)
        .get();

      if (ticketSnap.empty) throw new Error('NO_TICKETS_TO_SETTLE');

      // agrupar por produtor
      const byProducer = new Map<string, { amount: number, tickets:
string[] }>();
      let gross = 0;

      for (const doc of ticketSnap.docs) {
        const t = doc.data();
        const producerTenantId = t.producerTenantId;
        const amount = Number(t.pricing?.amountCents || 0);

        gross += amount;

        const prev = byProducer.get(producerTenantId) ?? { amount: 0,
tickets: [] };
        prev.amount += amount;
        prev.tickets.push(t.ticketId);
        byProducer.set(producerTenantId, prev);
      }

      const holdBack = Math.round(gross * (holdBackPercent / 100));
      const escrowAmount = gross - holdBack;

      const escrowId = makeId('esc_mlk');
```

```
      const escrowRef =
this.db.collection('milk_escrows').doc(escrowId);

      // montar splitPlan (proporcional, aplicando holdback por
produtor)
      const targets: any[] = [];
      for (const [producerTenantId, agg] of byProducer.entries()) {
        const producerHold = Math.round(agg.amount * (holdBackPercent
/ 100));
        const producerPay = agg.amount - producerHold;

        targets.push({
          producerTenantId,
          amountCents: producerPay,
          ticketIds: agg.tickets
        });
      }

      tx.create(escrowRef, {
        tenantId,
        escrowId,
        collectionId,
        poolId: dto.poolId,
        poolContractId: dto.poolContractId,
        buyerTenantId,
        status: 'REQUESTED',
        grossAmountCents: gross,
        holdBackPercent,
        holdBackCents: holdBack,
        amountCents: escrowAmount,
        splitPlan: { targets },
        createdAt: now,
        createdBy: { uid: actor.uid, role: actor.role }
      });

      // travar collection
      tx.set(colRef, {
        status: 'LOCKED',
        lockedAt: now,
        escrowId
      }, { merge: true });

      // marcar tickets como PLEDGED (travados no escrow)
      for (const doc of ticketSnap.docs) {
        const tRef =
this.db.collection('milk_daily_tickets').doc(doc.id);
        tx.set(tRef, {
          status: 'PLEDGED',
          settlement: { collectionId, escrowId }
        }, { merge: true });
      }

      // auditoria
      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        type: 'MILK_COLLECTION_LOCKED_FROM_TICKETS',
        actor: { uid: actor.uid, role: actor.role },
        meta: { collectionId, escrowId, gross, holdBackPercent,
escrowAmount, upToDate: dto.upToDate },
```

```
        ts: now
      });

      const out = {
        collectionId,
        status: 'LOCKED',
        escrow: { escrowId, grossAmountCents: gross, holdBackPercent,
amountCents: escrowAmount },
        producers: targets.length,
        tickets: ticketSnap.size
      };

      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'milk.collection.settleFromTickets', clientRequestId:
dto.clientRequestId, createdAt: now, result: out });
      return out;
    });
  }
}
```

Controller

```
// src/milk/milk-collections.controller.ts
import { Body, Controller, Param, Post, UseGuards } from
'@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { MilkCollectionsSettlementService } from './milk-collections-
settlement.service';
import { MilkSettleFromTicketsDto } from './dto/milk-settle-from-
tickets.dto';

@Controller('/api/milk/collections')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class MilkCollectionsController {
  constructor(private readonly settleSvc:
MilkCollectionsSettlementService) {}

  @Post('/:id/settle-from-tickets')
  @Roles('ADMIN','MANAGER','LOGISTICS','SUPPLIER','AUDITOR')
  @Permissions('milk.collection.settle')
  async settleFromTickets(@Actor() actor: any, @Param('id') id:
string, @Body() dto: MilkSettleFromTicketsDto) {
    return this.settleSvc.settleFromTickets(actor, id, dto);
  }
}
```

---

2) Webhook Asaas → executeSplitForTickets()

A premissa:

- Quando o Asaas confirmar pagamento (escrow “HELD”), você:
  - marca escrow HELD

- marca tickets `SETTLED`
- registra auditoria
- marca collection `PAID`

2.1 Função de liquidação (service)

```
// src/milk/milk-escrow-settlement.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class MilkEscrowSettlementService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async onEscrowPaid(escrowId: string, paidAt: number) {
    const escrowRef =
this.db.collection('milk_escrows').doc(escrowId);

    return this.db.runTransaction(async (tx) => {
      const escSnap = await tx.get(escrowRef);
      if (!escSnap.exists) throw new Error('ESCROW_NOT_FOUND');

      const esc = escSnap.data()!;
      if (esc.status === 'RELEASED') return { ok: true, already: true
};

      // 1) marcar escrow como HELD -> RELEASED (se você liberar
imediato)
      tx.set(escrowRef, { status: 'RELEASED', paidAt, releasedAt:
paidAt }, { merge: true });

      // 2) liquidar tickets
      const targets = esc.splitPlan?.targets || [];
      let settledTickets = 0;

      for (const t of targets) {
        const ticketIds: string[] = t.ticketIds || [];
        for (const ticketId of ticketIds) {
          const tRef =
this.db.collection('milk_daily_tickets').doc(ticketId);
          tx.set(tRef, {
            status: 'SETTLED',
            settledAt: paidAt
          }, { merge: true });
          settledTickets++;
        }
      }

      // 3) marcar collection como PAID
      if (esc.collectionId) {

tx.set(this.db.collection('milk_collections').doc(esc.collectionId), {
          status: 'PAID',
          paidAt,
          escrowId
        }, { merge: true });
      }

      // 4) auditoria
```

```
      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId: esc.tenantId,
        type: 'MILK_ESCROW_RELEASED_AND_TICKETS_SETTLED',
        meta: { escrowId, collectionId: esc.collectionId,
settledTickets, amountCents: esc.amountCents },
        ts: paidAt
      });

      return { ok: true, escrowId, settledTickets };
    });
  }
}
```

2.2 Handler do webhook

No seu webhook do Asaas, quando identificar pagamento do escrow:

- extrai `escrowId` do metadata (ou referência)
- chama `onEscrowPaid(escrowId, Date.now())`

Se você já tem webhook Asaas, só "pluga" esse service.

---

3) UI Produtor: tickets a receber / pagos + registrar tanque

3.1 Endpoints para UI

Listar tickets por produtor

```
GET /api/milk/daily-
tickets?producerTenantId=...&status=ISSUED|PLEDGED|SETTLED
```

Service simples (query por producerTenantId + status):

```
// src/milk/milk-tickets-read.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

@Injectable()
export class MilkTicketsReadService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async list(actor: any, producerTenantId: string, status?: string) {
    // segurança: produtor só vê os próprios
    if (actor.tenantId !== producerTenantId && actor.role !== 'ADMIN'
&& actor.role !== 'AUDITOR') {
      throw new Error('FORBIDDEN');
    }

    let q = this.db.collection('milk_daily_tickets')
      .where('producerTenantId', '==', producerTenantId)
      .limit(200);
```

```
    if (status) q = q.where('status', '==', status);

    const snap = await q.get();
    return { items: snap.docs.map(d => d.data()) };
  }
}
```

3.2 Tela “Tickets”

```
// src/screens/MilkTicketsScreen.tsx
import React, { useMemo, useState } from 'react';
import { FlatList, Text, View } from 'react-native';
import { useQuery } from '@tanstack/react-query';
import { api } from '../services/api';
import { BigButton } from '../ui/BigButton';

export function MilkTicketsScreen({ navigation, route }: any) {
  const { producerTenantId, propertyId } = route.params;
  const [status, setStatus] =
useState<'ISSUED'|'PLEDGED'|'SETTLED'>('ISSUED');

  const q = useQuery({
    queryKey: ['milkTickets', producerTenantId, status],
    queryFn: async () => (await api.get('/api/milk/daily-tickets', {
params: { producerTenantId, status } })).data,
  });

  const items = useMemo(() => q.data?.items || [], [q.data]);

  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 24, fontWeight: '800' }}>Tickets do
Leite</Text>
      <Text style={{ fontSize: 16, opacity: 0.8, marginTop: 8 }}>
        Direitos de receber gerados pelo conferente.
      </Text>

      <View style={{ marginTop: 12 }}>
        <BigButton icon="🟢" title="A receber (emitidos)"
subtitle="Tickets gerados hoje/ontem" onPress={() =>
setStatus('ISSUED')} />
        <BigButton icon="🟡" title="Em pagamento (travados)"
subtitle="Já entraram no escrow" onPress={() => setStatus('PLEDGED')}
/>
        <BigButton icon="✅" title="Pagos" subtitle="Liquidados pelo
escrow" onPress={() => setStatus('SETTLED')} />
      </View>

      <View style={{ marginTop: 12 }}>
        <BigButton
          icon="📍"
          title="Registrar tanque (conferente)"
          subtitle="Foto + GPS + litros"
          onPress={() => navigation.navigate('MilkTankCheck', {
producerTenantId, propertyId })}
        />
      </View>

      <FlatList
        style={{ marginTop: 12 }}
        data={items}
        keyExtractor={(it) => it.ticketId}
```

```
        refreshing={q.isFetching}
        onRefresh={() => q.refetch()}
        renderItem={({ item }) => (
          <BigButton
            icon={item.status === 'SETTLED' ? '✅' : item.status ===
'PLEDGED' ? '🟡' : '🟢'}
            title={`${item.date} • ${item.liters} L`}
            subtitle={`R$ ${(item.pricing?.amountCents ?? 0)/100} •
${item.status}`}
            onPress={() => {}}
          />
        )}
      />
    </View>
  );
}
```

3.3 Tela “Tank Check” (conferente)

Você já tem a tela de registrar tanque; aqui só liga a navegação.

---

4) Regras cruciais para “conferente diário”

Para ficar à prova de fraude:

- **1 ticket/dia por propriedade** (chave determinística `tktd_DATE_propertyId`)
- **Geofence obrigatório**
- **Foto do medidor obrigatório**
- **Somente conferente credenciado** (permissão)
- **Auditor pode contestar** e cancelar ticket (com histórico)

Próximo passo (se você quiser continuar no mesmo ritmo)

Eu fecho a parte do **conferente com rota** (LOGISTICS):

- o caminhão abre a “Rota do dia”
- cada parada valida/assina as leituras do conferente
- isso reduz disputa e melhora a aceitação do pagamento

Se quiser, mando esse módulo “Rota de Coleta” completo também.

Perfeito — abaixo está o módulo **Rota de Coleta (LOGISTICS)** completo, encaixado com **conferente diário + tickets** e com **lastro forte** (GPS/tempo/foto/assinatura).

A meta é: **o caminhão abre a rota**, em cada propriedade ele **valida/assina** a leitura do conferente (tank check/ticket), e isso vira “aceite” que **reduz disputa** e acelera **escrow + split**.

---

1) Modelo operacional da Rota

Papéis

- **LOGISTICS** (motorista/transportador): executa rota, coleta, valida no local.
- **INSPECTOR/TECHNICIAN** (conferente): mede diariamente e gera ticket.
- **AUDITOR**: resolve disputa e valida exceções.

Fluxo

1. LOGISTICS cria **Rota do dia** (Route) e abre status `OPEN`.
2. Para cada parada (Stop/Property):
    - sistema mostra o **Tank Check do dia** (se existir) + tickets pendentes.
    - motorista faz **Check-in** (GPS + horário) e valida:
        - volume coletado (pode diferir do tanque — fica registrado)
        - foto do lacre / bomba / medidor do caminhão (opcional)
        - "assinatura/aceite" (PIN, assinatura touch, ou atestado do dispositivo)
3. Ao final: `COMPLETE`
4. Rota completa → gera **Collection** e pode chamar `settle-from-tickets` automaticamente.

---

2) Firestore: modelos

2.1 Rota

```
milk_routes/{routeId}
```

```
{
  "tenantId":"TENANT_PLATFORM",
  "routeId":"rt_2026_02_05_trk01",
  "poolId":"pool_mg_regiao_01",
  "poolContractId":"pc_001",
  "buyerTenantId":"TENANT_INDUSTRY",

  "vehicleId":"trk_01",
  "driverUserId":"u_driver_01",

  "status":"OPEN|IN_PROGRESS|COMPLETED|CANCELLED",
  "createdAt":1730000000000,
  "startedAt":null,
  "completedAt":null
}
```

2.2 Paradas (stops) da rota

```
milk_route_stops/{stopId}
```

```
{
  "tenantId":"TENANT_PLATFORM",
  "stopId":"st_rt_2026_02_05_trk01_001",
  "routeId":"rt_2026_02_05_trk01",
```

```
  "producerTenantId":"TENANT_P1",
  "propertyId":"prop_p1",
  "order": 1,

  "status":"PENDING|ARRIVED|VERIFIED|DISPUTED|SKIPPED",
  "arrivedAt": null,
  "verifiedAt": null,

  "tankCheck": {
    "checkId":"chk_2026-02-05_prop_p1",
    "ticketId":"tktd_2026-02-05_prop_p1",
    "tankLiters": 820
  },

  "collected": {
    "litersCollected": 800,
    "temperature": 4.3,
    "sealOk": true
  },

  "evidence": {
    "riskType":"B",

"evidenceIds":["evg_gps_arrival","evg_photo_truck_meter","evg_signatur
e"]
  },

  "acceptance": {
    "acceptedByDriver": true,
    "acceptedByProducer": false,
    "producerPinHash": null
  }
}
```

2.3 Aceite / assinatura (opcional, imutável)

```
milk_acceptances/{accId}

{
  "tenantId":"TENANT_PLATFORM",
  "accId":"acc_001",
  "routeId":"rt_...",
  "stopId":"st_...",
  "propertyId":"prop_p1",
  "signMethod":"PIN|SIGNATURE|DEVICE_ATTEST",
  "signatureProof":"hash/metadata",
  "createdAt":1730000000000
}
```

2.4 Collection gerada da rota

Você pode **reusar** `milk_collections/{collectionId}`:

- `collectionId = routeId` (recomendado)
  Assim rota e coleta são 1:1.

---

3) Endpoints (mínimos e suficientes)

3.1 Abrir rota

```
POST /api/milk/routes/open
```

- cria route + stops

Body (exemplo)

```
{
  "clientRequestId":"uuid",
  "poolId":"pool_mg_regiao_01",
  "poolContractId":"pc_001",
  "vehicleId":"trk_01",
  "stops":[
    { "producerTenantId":"TENANT_P1", "propertyId":"prop_p1",
"order":1 },
    { "producerTenantId":"TENANT_P2", "propertyId":"prop_p2",
"order":2 }
  ]
}
```

3.2 Iniciar rota

```
POST /api/milk/routes/:routeId/start
```

3.3 Chegada na parada (GPS obrigatório)

```
POST /api/milk/routes/:routeId/stops/:stopId/arrive
```

Body:

```
{
  "clientRequestId":"uuid",
  "gps": { "lat": -18.9, "lng": -48.2 },
  "evidenceIds":["evg_gps_arrival","evg_photo_property_gate"]
}
```

3.4 Validar parada (assina e liga ao ticket)

```
POST /api/milk/routes/:routeId/stops/:stopId/verify
```

Body:

```
{
  "clientRequestId":"uuid",
  "tankCheckId":"chk_2026-02-05_prop_p1",
  "ticketId":"tktd_2026-02-05_prop_p1",
  "litersCollected":800,
  "temperature":4.3,
  "sealOk": true,
  "evidenceIds":["evg_photo_truck_meter","evg_signature"]
}
```

3.5 Finalizar rota

```
POST /api/milk/routes/:routeId/complete
```

- marca route COMPLETED
- cria/atualiza `milk_collections` (1:1)
- (opcional) chama `settle-from-tickets` automaticamente (até a data)

---

4) Regras (RulesEngine) da rota

4.1 `RouteStopGeofenceRule`

- `arrive` só aceita se GPS dentro do geofence da propriedade (ou raio X m).

4.2 `TankCheckMustExistRule`

- para verificar a parada, o `tankCheckId` precisa existir e ser do **mesmo dia** (ou janela permitida).

4.3 `TicketMustMatchCheckRule`

- ticket informado deve corresponder ao check (`ticket.checkId == tankCheckId`) e estar `ISSUED`.

4.4 `MaxDeviationRule` (opcional)

- se `abs(tankLiters - litersCollected) > limite`, vira `DISPUTED` automaticamente e exige auditor.

4.5 `StopStateMachineRule`

- `PENDING → ARRIVED → VERIFIED`
- não deixa "pular" estados.

---

5) Implementação NestJS (núcleo dos 3 endpoints)

Abaixo vai o essencial com transações e auditoria.

5.1 Open route

```
// src/milk/milk-routes.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class MilkRoutesService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async open(actor: any, dto: any) {
```

```
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'milk.routes.open');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
      if (idem.exists) return idem.data()!.result;

      const routeId = `rt_${new
Date().toISOString().slice(0,10)}_${dto.vehicleId}`;
      const routeRef = this.db.collection('milk_routes').doc(routeId);

      tx.create(routeRef, {
        tenantId,
        routeId,
        poolId: dto.poolId,
        poolContractId: dto.poolContractId,
        vehicleId: dto.vehicleId,
        driverUserId: actor.uid,
        status: 'OPEN',
        createdAt: now
      });

      for (const s of dto.stops || []) {
        const stopId = makeId('st');
        tx.create(this.db.collection('milk_route_stops').doc(stopId),
{
          tenantId,
          stopId,
          routeId,
          producerTenantId: s.producerTenantId,
          propertyId: s.propertyId,
          order: s.order ?? 0,
          status: 'PENDING',
          arrivedAt: null,
          verifiedAt: null,
          tankCheck: null,
          collected: null,
          evidence: { riskType: 'B', evidenceIds: [] },
          acceptance: { acceptedByDriver: false, acceptedByProducer:
false, producerPinHash: null }
        });
      }

      const out = { routeId, status: 'OPEN' };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'milk.routes.open', clientRequestId: dto.clientRequestId, createdAt:
now, result: out });
      return out;
    });
  }

  async arrive(actor: any, routeId: string, stopId: string, dto: any)
{
    const now = Date.now();
    const stopRef =
this.db.collection('milk_route_stops').doc(stopId);
```

```
    await this.db.runTransaction(async (tx) => {
      const snap = await tx.get(stopRef);
      if (!snap.exists) throw new Error('STOP_NOT_FOUND');
      const st = snap.data()!;
      if (st.routeId !== routeId) throw new
Error('STOP_ROUTE_MISMATCH');
      if (st.status !== 'PENDING') throw new
Error('STOP_NOT_PENDING');

      tx.set(stopRef, {
        status: 'ARRIVED',
        arrivedAt: now,
        evidence: { ...st.evidence, evidenceIds: Array.from(new
Set([...(st.evidence?.evidenceIds || []), ...(dto.evidenceIds ||
[])])) },
        gpsArrival: dto.gps ?? null
      }, { merge: true });

      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId: st.tenantId,
        type: 'MILK_ROUTE_STOP_ARRIVED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { routeId, stopId, propertyId: st.propertyId },
        ts: now
      });
    });

    return { routeId, stopId, status: 'ARRIVED' };
  }

  async verify(actor: any, routeId: string, stopId: string, dto: any)
{
    const now = Date.now();
    const stopRef =
this.db.collection('milk_route_stops').doc(stopId);

    return this.db.runTransaction(async (tx) => {
      const snap = await tx.get(stopRef);
      if (!snap.exists) throw new Error('STOP_NOT_FOUND');
      const st = snap.data()!;
      if (st.routeId !== routeId) throw new
Error('STOP_ROUTE_MISMATCH');
      if (st.status !== 'ARRIVED') throw new
Error('STOP_NOT_ARRIVED');

      // carrega tankCheck + ticket
      const chkRef =
this.db.collection('milk_tank_checks').doc(dto.tankCheckId);
      const chkSnap = await tx.get(chkRef);
      if (!chkSnap.exists) throw new Error('TANK_CHECK_NOT_FOUND');
      const chk = chkSnap.data()!;
      if (chk.propertyId !== st.propertyId) throw new
Error('CHECK_PROPERTY_MISMATCH');

      const tRef =
this.db.collection('milk_daily_tickets').doc(dto.ticketId);
      const tSnap = await tx.get(tRef);
      if (!tSnap.exists) throw new Error('TICKET_NOT_FOUND');
      const t = tSnap.data()!;
```

```
      if (t.checkId !== dto.tankCheckId) throw new
Error('TICKET_CHECK_MISMATCH');
      if (t.status !== 'ISSUED') throw new Error('TICKET_NOT_ISSUED');

      const tankLiters = Number(chk.reading?.liters || 0);
      const litersCollected = Number(dto.litersCollected || 0);

      // desvio grande -> DISPUTED automático (opcional)
      const deviation = Math.abs(tankLiters - litersCollected);
      const deviationLimit = 200; // ajuste conforme regra
      const status = deviation > deviationLimit ? 'DISPUTED' :
'VERIFIED';

      tx.set(stopRef, {
        status,
        verifiedAt: now,
        tankCheck: { checkId: dto.tankCheckId, ticketId: dto.ticketId,
tankLiters },
        collected: { litersCollected, temperature: dto.temperature ??
null, sealOk: !!dto.sealOk },
        evidence: { ...st.evidence, evidenceIds: Array.from(new
Set([...(st.evidence?.evidenceIds || []), ...(dto.evidenceIds ||
[])])) },
        acceptance: { ...st.acceptance, acceptedByDriver: true }
      }, { merge: true });

      // se verified: travar ticket para a rota (opcional: vira
PLEDGED quando settle-from-tickets rodar)
      // aqui só registra vínculo:
      tx.set(tRef, {
        routeId,
        routeStopId: stopId
      }, { merge: true });

      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId: st.tenantId,
        type: 'MILK_ROUTE_STOP_VERIFIED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { routeId, stopId, ticketId: dto.ticketId, tankLiters,
litersCollected, deviation, status },
        ts: now
      });

      return { routeId, stopId, status, deviation };
    });
  }

  async complete(actor: any, routeId: string, dto: any) {
    const now = Date.now();
    const routeRef = this.db.collection('milk_routes').doc(routeId);

    return this.db.runTransaction(async (tx) => {
      const rs = await tx.get(routeRef);
      if (!rs.exists) throw new Error('ROUTE_NOT_FOUND');
      const r = rs.data()!;
      if (!['OPEN','IN_PROGRESS'].includes(r.status)) throw new
Error('ROUTE_NOT_ACTIVE');
```

```
      tx.set(routeRef, { status: 'COMPLETED', completedAt: now }, {
merge: true });

      // cria/atualiza collection 1:1 (routeId == collectionId)
      const colRef =
this.db.collection('milk_collections').doc(routeId);
      tx.set(colRef, {
        tenantId: r.tenantId,
        collectionId: routeId,
        poolContractId: r.poolContractId,
        poolId: r.poolId,
        buyerTenantId: r.buyerTenantId ?? null,
        route: { vehicleId: r.vehicleId, driverUserId: r.driverUserId
},
        status: 'OPEN',
        createdAt: r.createdAt,
        completedRouteAt: now
      }, { merge: true });

      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId: r.tenantId,
        type: 'MILK_ROUTE_COMPLETED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { routeId, collectionId: routeId },
        ts: now
      });

      return { routeId, status: 'COMPLETED', collectionId: routeId };
    });
  }
}
```

5.2 Controller

```
// src/milk/milk-routes.controller.ts
import { Body, Controller, Param, Post, UseGuards } from
'@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { MilkRoutesService } from './milk-routes.service';

@Controller('/api/milk/routes')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class MilkRoutesController {
  constructor(private readonly svc: MilkRoutesService) {}

  @Post('/open')
  @Roles('ADMIN','MANAGER','LOGISTICS')
  @Permissions('milk.routes.open')
  open(@Actor() actor: any, @Body() dto: any) {
    return this.svc.open(actor, dto);
  }

  @Post('/:routeId/stops/:stopId/arrive')
  @Roles('ADMIN','MANAGER','LOGISTICS')
```

```
  @Permissions('milk.routes.arrive')
  arrive(@Actor() actor: any, @Param('routeId') routeId: string,
@Param('stopId') stopId: string, @Body() dto: any) {
    return this.svc.arrive(actor, routeId, stopId, dto);
  }

  @Post('/:routeId/stops/:stopId/verify')
  @Roles('ADMIN','MANAGER','LOGISTICS')
  @Permissions('milk.routes.verifyStop')
  verify(@Actor() actor: any, @Param('routeId') routeId: string,
@Param('stopId') stopId: string, @Body() dto: any) {
    return this.svc.verify(actor, routeId, stopId, dto);
  }

  @Post('/:routeId/complete')
  @Roles('ADMIN','MANAGER','LOGISTICS')
  @Permissions('milk.routes.complete')
  complete(@Actor() actor: any, @Param('routeId') routeId: string,
@Body() dto: any) {
    return this.svc.complete(actor, routeId, dto);
  }
}
```

---

6) UI Mobile LOGISTICS (rota super simples)

Tela “Rota do Dia”

- lista de paradas (botões grandes)
- cada parada mostra: propriedade + status + ticket do dia

Ações:

- **Cheguei** (GPS)
- **Validar** (litros coletados + foto medidor caminhão)
- **Próxima**

Campos grandes, com voz opcional: “coletei oitocentos litros”.

---

7) Integração automática com pagamento (o encaixe perfeito)

Depois de `complete(routeId)`:

- você chama:
  - `POST /api/milk/collections/:id/settle-from-tickets` com `upToDate = hoje`
- aí cria escrow + splitPlan
- Asaas paga → webhook liquida tickets

✅ Resultado:

- conferente gera ticket diário (direito)

- caminhão valida no local (reduz disputa)
- indústria paga via escrow
- split automático por produtor

---

8) Permissões (JSON rápido)

- `milk.routes.open`
- `milk.routes.arrive`
- `milk.routes.verifyStop`
- `milk.routes.complete`
- `milk.collection.settle` (já existe do módulo anterior)

Se você quiser, eu mando também a **tela TargetPicker** do LOGISTICS + "offline queue" (cheguei/validei sem internet e sincroniza depois) para ficar perfeito no campo.

Fechado — aqui vai o bloco **offline queue + sincronização** (cheguei/validei/complete sem internet) e a **TargetPicker do LOGISTICS** (rota/parada) com botões grandes e suporte a **voz** (simulado).

A lógica é:

- eventos críticos (ARRIVE / VERIFY / COMPLETE) viram "jobs" locais
- o app tenta enviar
- se falhar, fica na fila
- quando volta internet, ele reenvia em ordem
- idempotência no backend garante "não duplica"

---

1) Padrão offline: Job Queue local

Estrutura do job

```
export type OfflineJob = {
  id: string;                  // uuid
  createdAt: number;
  type: 'MILK_ROUTE_ARRIVE'|'MILK_ROUTE_VERIFY'|'MILK_ROUTE_COMPLETE';
  request: {
    method: 'POST';
    url: string;
    body: any;
  };
  status: 'PENDING'|'SENDING'|'DONE'|'FAILED';
  attempts: number;
  lastError?: string;
};
```

Regra de ouro: toda request leva `clientRequestId` (uuid) → backend idempotente.

---

2) Implementação Expo/React Native (AsyncStorage)

2.1 storage e helpers

```
// src/offline/offlineQueue.ts
import AsyncStorage from '@react-native-async-storage/async-storage';
import { api } from '../services/api';
import { v4 as uuid } from 'uuid';

const KEY = 'CICLO_OFFLINE_QUEUE_V1';

async function loadQueue(): Promise<any[]> {
  const raw = await AsyncStorage.getItem(KEY);
  return raw ? JSON.parse(raw) : [];
}

async function saveQueue(q: any[]) {
  await AsyncStorage.setItem(KEY, JSON.stringify(q));
}

export async function enqueueJob(job: Omit<any,
'id'|'createdAt'|'status'|'attempts'>) {
  const q = await loadQueue();
  const full = {
    id: uuid(),
    createdAt: Date.now(),
    status: 'PENDING',
    attempts: 0,
    ...job,
  };
  q.push(full);
  await saveQueue(q);
  return full;
}

export async function getQueue() {
  return await loadQueue();
}
```

2.2 processador (reenvio automático)

```
// src/offline/offlineProcessor.ts
import { api } from '../services/api';
import AsyncStorage from '@react-native-async-storage/async-storage';

const KEY = 'CICLO_OFFLINE_QUEUE_V1';

async function loadQueue(): Promise<any[]> {
  const raw = await AsyncStorage.getItem(KEY);
  return raw ? JSON.parse(raw) : [];
}
async function saveQueue(q: any[]) {
  await AsyncStorage.setItem(KEY, JSON.stringify(q));
}

function backoffMs(attempts: number) {
  // 0->1s, 1->2s, 2->4s, ... até 60s
  return Math.min(60000, 1000 * Math.pow(2, attempts));
}

export async function processOfflineQueue() {
  const q = await loadQueue();
  let changed = false;
```

```
  for (let i = 0; i < q.length; i++) {
    const job = q[i];
    if (job.status === 'DONE') continue;
    if (job.status === 'SENDING') continue;

    // backoff simples
    const wait = backoffMs(job.attempts || 0);
    const lastTryAt = job.lastTryAt || 0;
    if (Date.now() - lastTryAt < wait) continue;

    q[i] = { ...job, status: 'SENDING', lastTryAt: Date.now() };
    changed = true;
    await saveQueue(q);

    try {
      const { method, url, body } = job.request;
      if (method === 'POST') await api.post(url, body);
      else throw new Error('METHOD_NOT_SUPPORTED');

      q[i] = { ...q[i], status: 'DONE', lastError: null };
      changed = true;
      await saveQueue(q);
    } catch (e: any) {
      q[i] = {
        ...q[i],
        status: 'FAILED',
        attempts: (job.attempts || 0) + 1,
        lastError: String(e?.message || e),
      };
      changed = true;
      await saveQueue(q);
    }
  }

  return { changed, total: q.length, pending: q.filter(x => x.status
!== 'DONE').length };
}
```

2.3 “pulsador” (rodar a cada X segundos)

```
// src/offline/useOfflineSync.tsx
import { useEffect } from 'react';
import { processOfflineQueue } from './offlineProcessor';

export function useOfflineSync() {
  useEffect(() => {
    const t = setInterval(() => {
      processOfflineQueue();
    }, 6000);

    return () => clearInterval(t);
  }, []);
}
```

Chame `useOfflineSync()` no seu `App.tsx` ou no `RootNavigator`.

---

3) Como usar na Rota (ARRIVE / VERIFY / COMPLETE)

A lógica de tela:

- tenta enviar
- se falhar, enfileira
- mostra "✅ Registrado offline — será enviado quando voltar internet"

3.1 ARRIVE (offline-first)

```
import { enqueueJob } from '../offline/offlineQueue';
import { api } from '../services/api';
import { v4 as uuid } from 'uuid';

async function arriveStop(routeId: string, stopId: string, gps: any,
evidenceIds: string[]) {
  const body = {
    clientRequestId: uuid(),
    gps,
    evidenceIds
  };

  const url = `/api/milk/routes/${routeId}/stops/${stopId}/arrive`;

  try {
    await api.post(url, body);
    return { ok: true, mode: 'online' };
  } catch (e) {
    await enqueueJob({
      type: 'MILK_ROUTE_ARRIVE',
      request: { method: 'POST', url, body }
    });
    return { ok: true, mode: 'offline_queued' };
  }
}
```

3.2 VERIFY (offline-first)

```
async function verifyStop(routeId: string, stopId: string, payload:
any) {
  const body = { clientRequestId: uuid(), ...payload };
  const url = `/api/milk/routes/${routeId}/stops/${stopId}/verify`;

  try {
    await api.post(url, body);
    return { ok: true, mode: 'online' };
  } catch (e) {
    await enqueueJob({
      type: 'MILK_ROUTE_VERIFY',
      request: { method: 'POST', url, body }
    });
    return { ok: true, mode: 'offline_queued' };
  }
}
```

3.3 COMPLETE (offline-first)

```
async function completeRoute(routeId: string) {
  const body = { clientRequestId: uuid() };
  const url = `/api/milk/routes/${routeId}/complete`;

  try {
```

```
    await api.post(url, body);
    return { ok: true, mode: 'online' };
  } catch (e) {
    await enqueueJob({
      type: 'MILK_ROUTE_COMPLETE',
      request: { method: 'POST', url, body }
    });
    return { ok: true, mode: 'offline_queued' };
  }
}
```

---

4) Tela TargetPicker do LOGISTICS (rota → parada)

Você pediu "fácil para quem tem dificuldade de leitura", então:

- botões grandes
- ícones
- texto curto
- status com cor por emoji
- botão 🎙 "Falar o número da parada" (simulado)

4.1 Tela "Paradas da Rota"

```
// src/screens/LogisticsRouteStopsScreen.tsx
import React, { useMemo, useState } from 'react';
import { FlatList, Text, TextInput, View } from 'react-native';
import { useQuery } from '@tanstack/react-query';
import { api } from '../services/api';
import { BigButton } from '../ui/BigButton';

export function LogisticsRouteStopsScreen({ navigation, route }: any)
{
  const { routeId } = route.params;
  const [qText, setQText] = useState('');

  const q = useQuery({
    queryKey: ['milkRouteStops', routeId],
    queryFn: async () => (await api.get('/api/milk/routes/' + routeId
+ '/stops')).data,
  });

  const items = useMemo(() => {
    const list = q.data?.items || [];
    if (!qText.trim()) return list;
    const qq = qText.trim().toLowerCase();
    return list.filter((x: any) =>
String(x.propertyId).toLowerCase().includes(qq) ||
String(x.order).includes(qq));
  }, [q.data, qText]);

  async function voiceSearch() {
    // simula: falar "parada 2"
    setQText('2');
  }

  function statusIcon(s: string) {
    if (s === 'VERIFIED') return '✅';
```

```
    if (s === 'ARRIVED') return '📍';
    if (s === 'DISPUTED') return '⚠️';
    if (s === 'SKIPPED') return '⏭';
    return '⏳';
  }

  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 24, fontWeight: '800' }}>Rota:
Paradas</Text>
      <Text style={{ fontSize: 16, opacity: 0.8, marginTop: 6 }}>
        Toque na parada para chegar e validar.
      </Text>

      <TextInput
        value={qText}
        onChangeText={setQText}
        placeholder="Buscar: número ou propriedade"
        style={{ borderWidth: 1, borderRadius: 12, padding: 12,
fontSize: 16, marginTop: 12 }}
      />
      <BigButton icon="🎙" title="Falar a parada" subtitle="Ex.:
“parada 2”" onPress={voiceSearch} />

      <FlatList
        style={{ marginTop: 12 }}
        data={items}
        keyExtractor={(it) => it.stopId}
        refreshing={q.isFetching}
        onRefresh={() => q.refetch()}
        renderItem={({ item }) => (
          <BigButton
            icon={statusIcon(item.status)}
            title={`Parada ${item.order} • ${item.propertyId}`}
            subtitle={`${item.status} • Produtor:
${item.producerTenantId}`}
            onPress={() => navigation.navigate('LogisticsStopAction',
{ routeId, stopId: item.stopId })}
          />
        )}
      />
    </View>
  );
}
```

---

5) Endpoint faltante para a tela: GET /api/milk/routes/:routeId/stops

Controller + Service simples

```
// src/milk/milk-routes-read.controller.ts
import { Controller, Get, Param, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
```

```
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { MilkRoutesReadService } from './milk-routes-read.service';

@Controller('/api/milk/routes')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class MilkRoutesReadController {
  constructor(private readonly svc: MilkRoutesReadService) {}

  @Get('/:routeId/stops')
  @Roles('ADMIN','MANAGER','LOGISTICS','AUDITOR')
  @Permissions('milk.routes.readStops')
  async stops(@Actor() actor: any, @Param('routeId') routeId: string)
{
    return this.svc.listStops(actor, routeId);
  }
}
// src/milk/milk-routes-read.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

@Injectable()
export class MilkRoutesReadService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async listStops(actor: any, routeId: string) {
    const snap = await this.db.collection('milk_route_stops')
      .where('routeId', '==', routeId)
      .orderBy('order')
      .limit(500)
      .get();

    return { items: snap.docs.map(d => d.data()) };
  }
}
```

---

6) O que isso resolve (na prática)

✅ motorista consegue operar **sem internet**
✅ tudo fica **com lastro** (foto + GPS + hora)
✅ idempotência evita duplicar dados quando sincronizar
✅ produtor tem tickets diários e a rota valida no local
✅ reduz disputa e acelera pagamento por escrow/split

Se você quiser, eu fecho o “modo ultra simples” para pessoas com pouca leitura:

- botões ainda maiores
- modo “apenas 3 ações”: **Cheguei / Foto / Confirmar litros**
- voz para “ditar litros”
- leitura automática de números na foto do medidor (OCR) **somente quando necessário**

Fechado — abaixo está o **Modo Ultra Simples** (3 ações) para o motorista/conferente/produtor, com:

- botões **gigantes**
- fluxo "**Cheguei → Foto → Confirmar litros**"
- **voz** para ditar litros (simulado)
- **lançamento por foto de anotação em papel** (para estoque e também para leite) com OCR *opcional* (quando precisar)
- feedback visual claro ("salvo offline", "enviado", "pendente")

Observação: OCR em celular funciona melhor com biblioteca nativa (Google ML Kit / Vision). Aqui deixo a arquitetura e os endpoints. No backend, OCR deve ser "tolerante" e nunca substituir o documento fiscal — serve só para **facilitar digitação**.

---

1) UX: 3 ações

Tela 1 — "Cheguei"

- botão: 📍 **CHEGUEI**
- captura:
  - GPS
  - hora
  - (opcional) foto do portão/placa

Tela 2 — "Foto"

- botão: 📸 **TIRAR FOTO DO MEDIDOR**
- anexa evidência
- se offline, salva local e marca "fila"

Tela 3 — "Confirmar litros"

- 2 botões:
  - 🎙️ **FALAR LITROS**
  - ✅ **CONFIRMAR**
- campo numérico grande (pode ser preenchido pelo OCR ou voz)
- "enviar" (ou fila offline)

---

2) Componentes UI (botões gigantes)

2.1 BigActionButton (bem grande)

```
// src/ui/BigActionButton.tsx
import React from 'react';
import { Pressable, Text, View } from 'react-native';
```

```
export function BigActionButton({ icon, title, subtitle, onPress,
disabled }: any) {
  return (
    <Pressable
      onPress={onPress}
      disabled={!!disabled}
      style={{
        padding: 18,
        borderWidth: 2,
        borderRadius: 18,
        marginTop: 14,
        opacity: disabled ? 0.5 : 1
      }}
    >
      <View style={{ flexDirection: 'row', alignItems: 'center' }}>
        <Text style={{ fontSize: 42, marginRight: 14 }}>{icon}</Text>
        <View style={{ flex: 1 }}>
          <Text style={{ fontSize: 22, fontWeight: '900'
}}>{title}</Text>
          {!!subtitle && <Text style={{ fontSize: 16, opacity: 0.8,
marginTop: 4 }}>{subtitle}</Text>}
        </View>
      </View>
    </Pressable>
  );
}
```

---

3) Tela Ultra Simples: “Verificar Parada” (LOGISTICS)

3.1 LogisticsStopUltraScreen

```
// src/screens/LogisticsStopUltraScreen.tsx
import React, { useState } from 'react';
import { Text, TextInput, View } from 'react-native';
import { BigActionButton } from '../ui/BigActionButton';
import { arriveStop, verifyStop } from '../milk/offlineActions'; //
usa offline queue
import { v4 as uuid } from 'uuid';

export function LogisticsStopUltraScreen({ navigation, route }: any) {
  const { routeId, stopId, tankCheckId, ticketId } = route.params;

  const [step, setStep] = useState<1|2|3>(1);
  const [evidenceIds, setEvidenceIds] = useState<string[]>([]);
  const [liters, setLiters] = useState<string>('');
  const [modeMsg, setModeMsg] = useState<string>('');

  async function doArrive() {
    const gps = { lat: 0, lng: 0 }; // aqui você pega do Location
    const res = await arriveStop(routeId, stopId, gps,
['evg_gps_arrival']);
    setModeMsg(res.mode === 'offline_queued' ? '✅ Salvo offline (vai
enviar depois)' : '✅ Enviado');
    setStep(2);
  }

  async function takePhotoMock() {
    // ideal: Evidence module real (upload/commit). Aqui simula.
    setEvidenceIds((p) => [...p, `evg_photo_${Date.now()}`]);
    setModeMsg('✅ Foto anexada');
```

```
    setStep(3);
  }

  async function voiceLitersMock() {
    // simulado: em produção usar speech-to-text
    setLiters('800');
    setModeMsg('🎙 Detectado: 800 litros');
  }

  async function confirm() {
    const payload = {
      tankCheckId,
      ticketId,
      litersCollected: Number(liters),
      temperature: 4.2,
      sealOk: true,
      evidenceIds: [...evidenceIds, 'evg_signature_mock']
    };
    const res = await verifyStop(routeId, stopId, payload);
    setModeMsg(res.mode === 'offline_queued' ? '✅ Validado offline
(vai enviar depois)' : '✅ Validado e enviado');
    navigation.goBack();
  }

  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 26, fontWeight: '900' }}>Verificação
(Ultra)</Text>
      <Text style={{ fontSize: 16, opacity: 0.8, marginTop: 6 }}>
        3 passos: Cheguei → Foto → Confirmar litros
      </Text>

      {!!modeMsg && (
        <View style={{ marginTop: 12, padding: 12, borderWidth: 1,
borderRadius: 12 }}>
          <Text style={{ fontSize: 16 }}>{modeMsg}</Text>
        </View>
      )}

      {step === 1 && (
        <BigActionButton icon="📍" title="CHEGUEI" subtitle="Marca GPS
e horário" onPress={doArrive} />
      )}

      {step === 2 && (
        <BigActionButton icon="📷" title="TIRAR FOTO DO MEDIDOR"
subtitle="Anexa evidência" onPress={takePhotoMock} />
      )}

      {step === 3 && (
        <>
          <Text style={{ marginTop: 14, fontSize: 18, fontWeight:
'800' }}>Litros coletados</Text>
          <TextInput
            value={liters}
            onChangeText={setLiters}
            keyboardType="numeric"
            placeholder="Ex.: 800"
            style={{ borderWidth: 2, borderRadius: 14, padding: 16,
fontSize: 22, marginTop: 10 }}
```

```
          />

          <BigActionButton icon="🎙" title="FALAR LITROS"
subtitle="Ex.: “oitocentos”" onPress={voiceLitersMock} />
          <BigActionButton icon="✅" title="CONFIRMAR"
subtitle="Valida e envia (ou fila offline)" onPress={confirm}
disabled={!liters || Number(liters) <= 0} />
        </>
      )}
    </View>
  );
}
```

---

4) “Lançar por foto de papel” (estoque e leite)

Você pediu exatamente:

“pode ser criado uma planilha e cada retirada anotada manualmente… faz foto… lança sem digitação”.

Vamos criar um módulo genérico:

4.1 Documento “manual log photo” (evidência)

```
manual_logs/{logId}

{
  "tenantId":"TENANT_X",
  "logId":"mlog_001",
  "propertyId":"prop_123",
  "type":"STOCK_WITHDRAW|MILK_DAILY",
  "photoEvidenceId":"evg_...",
  "ocr": {
    "status":"PENDING|DONE|FAILED",
    "rawText":"...",
    "parsed": {}
  },
  "createdAt":1730000000000,
  "createdBy":{"uid":"...","role":"OPERATOR"}
}
```

4.2 Endpoint: criar “manual log” + OCR opcional

1. `POST /api/manual-logs`
2. `POST /api/manual-logs/:id/parse` (roda OCR / parse)

Parse result exemplo (leite)

```
{
  "date":"2026-02-05",
  "liters":820,
  "temperature":4.2
}
```

Parse result exemplo (estoque)

```
{
  "items":[
    { "itemId":"sal_mineral", "qty":2, "unit":"SACO" },
    { "itemId":"vacina_x", "qty":10, "unit":"DOSE" }
  ]
}
```

Importante: OCR só **sugere**; o usuário confirma antes de virar movimentação real.

---

5) Como aplicar isso no fluxo (sem complicar)

Leite (conferente)

- Tela "Registrar tanque"
  - botão "📷 Foto da anotação" (papel)
  - OCR sugere litros
  - confirma → cria `tank_check` → gera ticket

Estoque (fazenda/loja)

- Tela "Baixa de estoque"
  - foto da planilha de retirada
  - OCR sugere `itemId/qty`
  - confirma → cria `farm_stock_move` OUT e `farm_application` se tiver alvo

---

6) Regras de segurança (para não virar fraude)

- OCR não cria nada sozinho — apenas **pré-preenche**
- confirmação exige:
  - usuário com permissão
  - evidência anexada
  - geofence (quando aplicável)
- tudo vira **audit_event** append-only

---

7) O que eu recomendo como "modo padrão" no campo

- **Ultra Simples** como primeira opção
- botão "Mais opções" para telas avançadas (quando o usuário for técnico)

Se você quiser, eu mando agora também:

- o endpoint `POST /api/manual-logs` e `POST /api/manual-logs/:id/parse`
- e um parser simples por regex (para começar) que reconhece padrões tipo:
  - "05/02 820L 4,2°C"
  - "Sal mineral 2 sacos"

A seguir vai **tudo** do módulo `manual-logs`:

- `POST /api/manual-logs` (cria log com evidência/foto)
- `POST /api/manual-logs/:id/parse` (faz parse do texto OCR enviado pelo app **ou** roda OCR externo futuramente)
- Parser **simples e útil** (regex) para:
  - **Leite**: "05/02 820L 4,2C"
  - **Estoque**: "Sal mineral 2 sacos", "Vacina X 10 doses"
- `POST /api/manual-logs/:id/commit` (confirma e converte em evento real: `milk_tank_check` ou `stock_movement`)
- Auditoria append-only + idempotência

Importante: o backend **não precisa** rodar OCR agora. O app pode mandar `rawText` (digitado/voz/OCR do celular). O backend só **interpreta** e devolve "sugestão" para o usuário confirmar.

---

1) Firestore

1.1 manual_logs/{logId}

```
{
  "tenantId":"TENANT_X",
  "logId":"mlog_001",
  "propertyId":"prop_123",
  "type":"MILK_DAILY|STOCK_WITHDRAW",
  "photoEvidenceId":"evg_photo_...",
  "rawText":"05/02 820L 4,2C\nSal mineral 2 sacos\nVacina X 10 doses",
  "parse": {
    "status":"PENDING|DONE|FAILED",
    "parsed": {},
    "errors":[]
  },
  "status":"CREATED|PARSED|COMMITTED|CANCELLED",
  "createdAt":1730000000000,
  "createdBy":{"uid":"...","role":"OPERATOR"}
}
```

---

2) DTOs

2.1 Create

```
// src/manual-logs/dto/manual-log-create.dto.ts
import { IsIn, IsNotEmpty, IsOptional, IsString } from 'class-
validator';

export class ManualLogCreateDto {
```

```
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsString() @IsNotEmpty()
  propertyId: string;

  @IsIn(['MILK_DAILY','STOCK_WITHDRAW'])
  type: 'MILK_DAILY'|'STOCK_WITHDRAW';

  @IsString() @IsNotEmpty()
  photoEvidenceId: string;

  @IsOptional() @IsString()
  rawText?: string; // pode vir do OCR do celular (recomendado)
}
```

2.2 Parse

```
// src/manual-logs/dto/manual-log-parse.dto.ts
import { IsOptional, IsString, IsNotEmpty } from 'class-validator';

export class ManualLogParseDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsOptional() @IsString()
  rawText?: string; // pode sobrescrever para nova tentativa
}
```

2.3 Commit

```
// src/manual-logs/dto/manual-log-commit.dto.ts
import { IsNotEmpty, IsOptional, IsString } from 'class-validator';

export class ManualLogCommitDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  // payload confirmado pelo usuário
  @IsNotEmpty()
  confirmed: any;

  // para leite: precisa do poolContractId/poolId/producerTenantId
  @IsOptional() @IsString()
  poolId?: string;

  @IsOptional() @IsString()
  poolContractId?: string;

  @IsOptional() @IsString()
  producerTenantId?: string;
}
```

---

3) Parser (regex) — simples e funcional

3.1 Parser service

```
// src/manual-logs/manual-log-parser.service.ts
import { Injectable } from '@nestjs/common';

function norm(s: string) {
  return (s || '')
    .replace(/\t/g, ' ')
```

```
    .replace(/[ ]+/g, ' ')
    .trim();
}

function toNumberLoose(v: string) {
  if (!v) return null;
  const x = v.replace(/\./g,'').replace(',', '.'); // 1.000,5 ->
1000.5
  const n = Number(x);
  return Number.isFinite(n) ? n : null;
}

function parseDateLoose(line: string) {
  // aceita: 05/02, 05-02, 2026-02-05
  const iso = line.match(/\b(\d{4})-(\d{2})-(\d{2})\b/);
  if (iso) return `${iso[1]}-${iso[2]}-${iso[3]}`;

  const br = line.match(/\b(\d{1,2})[\/\-](\d{1,2})\b/);
  if (br) {
    const dd = br[1].padStart(2,'0');
    const mm = br[2].padStart(2,'0');
    // ano: o app pode preencher na confirmação; aqui sugere só DD/MM
    return `${dd}/${mm}`;
  }
  return null;
}

@Injectable()
export class ManualLogParserService {
  parseMilk(rawText: string) {
    const lines = rawText.split('\n').map(norm).filter(Boolean);

    let date: string | null = null;
    let liters: number | null = null;
    let temperature: number | null = null;

    const errors: string[] = [];

    for (const ln of lines) {
      if (!date) {
        const d = parseDateLoose(ln);
        if (d) date = d;
      }

      // litros: "820L" / "820 l" / "820 litros"
      if (liters == null) {
        const m =
ln.match(/\b(\d{1,6}(?:[.,]\d{1,2})?)\s*(l|lt|litro|litros)\b/i);
        if (m) liters = toNumberLoose(m[1]);
      }

      // temp: "4,2C" / "4.2 °C"
      if (temperature == null) {
        const t = ln.match(/\b(\d{1,2}(?:[.,]\d{1,2})?)\s*(°c|c)\b/i);
        if (t) temperature = toNumberLoose(t[1]);
      }
    }

    if (liters == null) errors.push('LITERS_NOT_FOUND');
    // temperatura é opcional dependendo da regra, mas sugerimos
    const parsed = { date, liters, temperature };
```

```
    return { parsed, errors };
  }

  parseStock(rawText: string) {
    const lines = rawText.split('\n').map(norm).filter(Boolean);
    const items: any[] = [];
    const errors: string[] = [];

    for (const ln of lines) {
      // Ex.: "Sal mineral 2 sacos" / "Vacina X 10 doses" / "Ração 1,5
tonelada"
      const m =
ln.match(/^(.+?)\s+(\d{1,8}(?:[.,]\d{1,3})?)\s*(saco|sacos|kg|quilo|qu
ilos|l|litro|litros|dose|doses|un|unidade|unidades|caixa|caixas|ton|to
nelada|toneladas)\b/i);
      if (!m) continue;

      const name = norm(m[1]);
      const qty = toNumberLoose(m[2]);
      const unit = m[3].toUpperCase();

      if (!name || qty == null) continue;

      items.push({ name, qty, unit });
    }

    if (items.length === 0) errors.push('NO_STOCK_ITEMS_FOUND');

    return { parsed: { items }, errors };
  }
}
```

---

4) Controller + Service

4.1 Controller

```
// src/manual-logs/manual-logs.controller.ts
import { Body, Controller, Param, Post, UseGuards } from
'@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { ManualLogsService } from './manual-logs.service';
import { ManualLogCreateDto } from './dto/manual-log-create.dto';
import { ManualLogParseDto } from './dto/manual-log-parse.dto';
import { ManualLogCommitDto } from './dto/manual-log-commit.dto';

@Controller('/api/manual-logs')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class ManualLogsController {
  constructor(private readonly svc: ManualLogsService) {}

  @Post()
```

```
@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','LOGISTICS
')
  @Permissions('manualLogs.create')
  create(@Actor() actor: any, @Body() dto: ManualLogCreateDto) {
    return this.svc.create(actor, dto);
  }

  @Post('/:id/parse')

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','LOGISTICS
')
  @Permissions('manualLogs.parse')
  parse(@Actor() actor: any, @Param('id') id: string, @Body() dto:
ManualLogParseDto) {
    return this.svc.parse(actor, id, dto);
  }

  @Post('/:id/commit')

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','LOGISTICS
')
  @Permissions('manualLogs.commit')
  commit(@Actor() actor: any, @Param('id') id: string, @Body() dto:
ManualLogCommitDto) {
    return this.svc.commit(actor, id, dto);
  }
}
```

4.2 Service (create/parse/commit)

```
// src/manual-logs/manual-logs.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';
import { ManualLogParserService } from './manual-log-parser.service';

// integrações: reaproveita seus serviços existentes:
import { MilkTankChecksService } from '../milk/milk-tank-
checks.service';
// para estoque: você terá seu StockMovementsService (exemplo)
import { StockMovementsService } from '../inventory/stock-
movements.service';

@Injectable()
export class ManualLogsService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly parser: ManualLogParserService,
    private readonly milkTankChecks: MilkTankChecksService,
    private readonly stockMoves: StockMovementsService,
  ) {}

  async create(actor: any, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'manualLogs.create');
    const idemRef = this.db.collection('idempotency').doc(key);
```

```
    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
      if (idem.exists) return idem.data()!.result;

      const logId = makeId('mlog');
      const ref = this.db.collection('manual_logs').doc(logId);

      tx.create(ref, {
        tenantId,
        logId,
        propertyId: dto.propertyId,
        type: dto.type,
        photoEvidenceId: dto.photoEvidenceId,
        rawText: dto.rawText ?? '',
        parse: { status: 'PENDING', parsed: {}, errors: [] },
        status: 'CREATED',
        createdAt: now,
        createdBy: { uid: actor.uid, role: actor.role }
      });

      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        propertyId: dto.propertyId,
        type: 'MANUAL_LOG_CREATED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { logId, logType: dto.type, photoEvidenceId:
dto.photoEvidenceId },
        ts: now
      });

      const out = { logId, status: 'CREATED' };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'manualLogs.create', clientRequestId: dto.clientRequestId, createdAt:
now, result: out });
      return out;
    });
  }

  async parse(actor: any, logId: string, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'manualLogs.parse');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
      if (idem.exists) return idem.data()!.result;

      const ref = this.db.collection('manual_logs').doc(logId);
      const snap = await tx.get(ref);
      if (!snap.exists) throw new Error('MANUAL_LOG_NOT_FOUND');

      const log = snap.data()!;
      if (log.tenantId !== tenantId) throw new Error('FORBIDDEN');

      const rawText = (dto.rawText ?? log.rawText ?? '').toString();
      let parsed: any = {};
```

```
      let errors: string[] = [];

      if (log.type === 'MILK_DAILY') {
        const r = this.parser.parseMilk(rawText);
        parsed = r.parsed;
        errors = r.errors;
      } else if (log.type === 'STOCK_WITHDRAW') {
        const r = this.parser.parseStock(rawText);
        parsed = r.parsed;
        errors = r.errors;
      } else {
        errors = ['UNSUPPORTED_TYPE'];
      }

      const status = errors.length ? 'FAILED' : 'DONE';

      tx.set(ref, {
        rawText,
        parse: { status, parsed, errors },
        status: 'PARSED',
        parsedAt: now
      }, { merge: true });

      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        propertyId: log.propertyId,
        type: 'MANUAL_LOG_PARSED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { logId, parseStatus: status, errors },
        ts: now
      });

      const out = { logId, parseStatus: status, parsed, errors };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'manualLogs.parse', clientRequestId: dto.clientRequestId, createdAt:
now, result: out });
      return out;
    });
  }

  async commit(actor: any, logId: string, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'manualLogs.commit');
    const idemRef = this.db.collection('idempotency').doc(key);

    // commit envolve chamar outros serviços -> fazemos "2 fases":
    // 1) trava/valida e marca COMMITTING
    // 2) executa ação e depois marca COMMITTED (fora da tx se
preferir)
    // aqui vamos fazer direto, mas com cuidado para idempotência.

    const logRef = this.db.collection('manual_logs').doc(logId);
    const logSnap = await logRef.get();
    if (!logSnap.exists) throw new Error('MANUAL_LOG_NOT_FOUND');
    const log = logSnap.data()!;
    if (log.tenantId !== tenantId) throw new Error('FORBIDDEN');
```

```
    // Idempotência do commit
    const idemSnap = await idemRef.get();
    if (idemSnap.exists) return idemSnap.data()!.result;

    let result: any = null;

    if (log.type === 'MILK_DAILY') {
      // confirmed deve conter: date, liters, temperature...
      const confirmed = dto.confirmed || {};
      // chama o serviço que já emite ticket diário
      result = await this.milkTankChecks.create(actor, {
        clientRequestId: `manuallog_${logId}`, // idempotente
        poolId: dto.poolId,
        poolContractId: dto.poolContractId,
        producerTenantId: dto.producerTenantId,
        poolContractIdForPricing: dto.poolContractId,
        poolContractId: dto.poolContractId,
        propertyId: log.propertyId,
        date: confirmed.date,
        reading: {
          liters: confirmed.liters,
          temperature: confirmed.temperature ?? null,
          meterType: confirmed.meterType ?? 'MANUAL',
          meterValue: confirmed.meterValue ?? null
        },
        evidence: { evidenceIds: [log.photoEvidenceId] },
        poolContractId: dto.poolContractId
      });
    }

    if (log.type === 'STOCK_WITHDRAW') {
      // confirmed.items: [{ itemId, qty, unit }]
      const confirmed = dto.confirmed || {};
      // chama serviço de movimentação de estoque (OUT)
      // OBS: você decide mapear name->itemId antes, ou confirmar
itemId na UI.
      result = await this.stockMoves.create(actor, {
        clientRequestId: `manuallog_${logId}`,
        propertyId: log.propertyId,
        moves: confirmed.items || [],
        evidenceIds: [log.photoEvidenceId],
        direction: 'OUT'
      });
    }

    // marca log como COMMITTED
    await logRef.set({
      status: 'COMMITTED',
      committedAt: now,
      committedBy: { uid: actor.uid, role: actor.role },
      commitResult: result
    }, { merge: true });

    await this.db.collection('audit_events').add({
      auditId: makeId('aud'),
      tenantId,
      propertyId: log.propertyId,
      type: 'MANUAL_LOG_COMMITTED',
      actor: { uid: actor.uid, role: actor.role },
      meta: { logId, logType: log.type },
```

```
      ts: now
    });

    await idemRef.set({ tenantId, uid: actor.uid, scope:
'manualLogs.commit', clientRequestId: dto.clientRequestId, createdAt:
now, result });
    return result;
  }
}
```

Ajuste pequeno: no `milkTankChecks.create` você já tem DTO próprio. Se preferir, crie um wrapper `createFromManualLog()` para não ficar feio.

---

5) StockMovementsService (stub esperado)

Como você já tem módulo `StockMovements`, o commit espera algo assim:

```
// src/inventory/stock-movements.service.ts
export class StockMovementsService {
  async create(actor: any, dto: any) {
    // dto: { clientRequestId, propertyId, direction:'OUT',
moves:[{itemId, qty, unit}], evidenceIds:[] }
    // aqui aplica:
    // - RulesEngine (evidência obrigatória)
    // - grava inventory/{itemId}/moves append-only
    // - atualiza saldo (balanceBefore/after)
    return { ok: true, movementBatchId: 'mvb_...' };
  }
}
```

---

6) UI para “foto do papel” (fluxo do operador)

1. Tirar foto
2. OCR do celular → `rawText`
3. `POST /api/manual-logs` (com rawText)
4. `POST /api/manual-logs/:id/parse`
5. UI mostra sugestão e permite corrigir
6. `POST /api/manual-logs/:id/commit`

✅ Zero digitação (na prática, só conferência).

---

7) Permissions (para você colocar no JSON)

- `manualLogs.create`
- `manualLogs.parse`
- `manualLogs.commit`
- (e as já existentes do leite/estoque)

Se você quiser, eu mando agora:

- as **rules Firestore** mínimas para `manual_logs` (tenantId + propertyId)
- e a tela mobile "Foto do Papel" (grande e simples) com preview do texto e botão "CONFIRMAR".

Abaixo estão **(1)** regras Firestore mínimas para `manual_logs` (multi-tenant + papéis) e **(2)** a tela mobile "Foto do Papel" (ultra simples) com preview do texto, correção rápida e botão **CONFIRMAR**.

---

1) Firestore Rules (mínimas e seguras)

Premissas:

- Todo doc tem `tenantId`
- Token tem `tenantId`, `role`, `permissions` (claims)
- Vamos permitir:
  - criar/logar manual_logs: OPERATOR/PRODUCER/TECHNICIAN/LOGISTICS/MANAGER/ADMIN
  - ler: mesmos + AUDITOR (read-only)
  - atualizar: apenas campos "não críticos" (rawText/parse/status) e sempre com tenantId igual
  - **não permitir** alterar `createdAt/createdBy/tenantId/propertyId/type` depois de criado

1.1 Helper de permissões (exemplo)

```
// firestore.rules (trecho)
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

    function signedIn() { return request.auth != null; }
    function tenantOk(data) { return signedIn() &&
request.auth.token.tenantId == data.tenantId; }

    function hasPerm(p) {
      return signedIn()
        && request.auth.token.permissions != null
        && request.auth.token.permissions.hasAny([p]);
    }

    function hasRole(roles) {
      return signedIn() && request.auth.token.role in roles;
    }

    function isSameTenant() {
      return signedIn() && request.auth.token.tenantId ==
request.resource.data.tenantId;
```

```
    }
```

1.2 Regras do `manual_logs`

```
    match /manual_logs/{logId} {

      // CREATE
      allow create: if signedIn()
        && isSameTenant()
        &&
hasRole(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','LOGISTI
CS'])
        && hasPerm('manualLogs.create');

      // READ
      allow get, list: if signedIn()
        && tenantOk(resource.data)
        &&
hasRole(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','LOGISTI
CS','AUDITOR'])
        && hasPerm('manualLogs.read');

      // UPDATE (restrito)
      allow update: if signedIn()
        && tenantOk(resource.data)
        &&
hasRole(['ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','LOGISTI
CS'])
        && (hasPerm('manualLogs.parse') ||
hasPerm('manualLogs.commit'))
        // bloqueia alteração de campos críticos
        && request.resource.data.tenantId == resource.data.tenantId
        && request.resource.data.propertyId ==
resource.data.propertyId
        && request.resource.data.type == resource.data.type
        && request.resource.data.createdAt == resource.data.createdAt
        && request.resource.data.createdBy.uid ==
resource.data.createdBy.uid;

      // DELETE: proibido (append-only)
      allow delete: if false;
    }
```

Observação importante

Para o backend NestJS, você normalmente não atualiza direto via app — mas essas rules impedem “jeitinho” no client.

---

2) Tela mobile “Foto do Papel” (Ultra Simples)

O que ela faz

1. usuário tira foto
2. app roda OCR local (ou digita/voz) → gera `rawText`
3. cria manual log
4. parse → mostra sugestão
5. usuário corrige com botões grandes

6. commit → vira evento real (leite ou estoque)

2.1 API helper

```
// src/services/manualLogsApi.ts
import { api } from './api';

export async function createManualLog(payload: any) {
  return (await api.post('/api/manual-logs', payload)).data;
}

export async function parseManualLog(logId: string, payload: any) {
  return (await api.post(`/api/manual-logs/${logId}/parse`,
payload)).data;
}

export async function commitManualLog(logId: string, payload: any) {
  return (await api.post(`/api/manual-logs/${logId}/commit`,
payload)).data;
}
```

2.2 Tela

```
// src/screens/ManualLogCaptureScreen.tsx
import React, { useState } from 'react';
import { ScrollView, Text, TextInput, View } from 'react-native';
import { v4 as uuid } from 'uuid';
import { BigActionButton } from '../ui/BigActionButton';
import { createManualLog, parseManualLog, commitManualLog } from
'../services/manualLogsApi';

type Mode = 'MILK_DAILY'|'STOCK_WITHDRAW';

export function ManualLogCaptureScreen({ navigation, route }: any) {
  const { propertyId, mode, poolId, poolContractId, producerTenantId }
= route.params as {
    propertyId: string;
    mode: Mode;
    poolId?: string;
    poolContractId?: string;
    producerTenantId?: string;
  };

  const [step, setStep] = useState<1|2|3|4>(1);
  const [photoEvidenceId, setPhotoEvidenceId] = useState<string>('');
  const [rawText, setRawText] = useState<string>('');
  const [logId, setLogId] = useState<string>('');
  const [parsed, setParsed] = useState<any>(null);
  const [msg, setMsg] = useState<string>('');

  // campos confirmados (UI grande)
  const [milkDate, setMilkDate] = useState<string>('');
  const [milkLiters, setMilkLiters] = useState<string>('');
  const [milkTemp, setMilkTemp] = useState<string>('');

  const [stockItemsText, setStockItemsText] = useState<string>('');
  // para MVP: usuário confirma via texto simples e você converte
depois
  // ideal: UI por linhas com itemId/qty
```

```
  async function takePhotoMock() {
    // Em produção: Evidence module (upload/commit) retorna evidenceId
    const evg = `evg_photo_${Date.now()}`;
    setPhotoEvidenceId(evg);
    setMsg('✅ Foto capturada');
    setStep(2);
  }

  async function ocrMock() {
    // Em produção: OCR local (ML Kit). Aqui simula texto.
    const demoMilk = '05/02 820L 4,2C';
    const demoStock = 'Sal mineral 2 sacos\nVacina X 10 doses';
    setRawText(mode === 'MILK_DAILY' ? demoMilk : demoStock);
    setMsg('✅ Texto reconhecido (OCR)');
  }

  async function createAndParse() {
    if (!photoEvidenceId) return;

    setMsg('⏳ Enviando...');
    const created = await createManualLog({
      clientRequestId: uuid(),
      propertyId,
      type: mode,
      photoEvidenceId,
      rawText
    });

    setLogId(created.logId);

    const parsedRes = await parseManualLog(created.logId, {
      clientRequestId: uuid(),
      rawText
    });

    setParsed(parsedRes.parsed);

    // pré-preenche campos
    if (mode === 'MILK_DAILY') {
      setMilkDate(parsedRes.parsed?.date || '');
      setMilkLiters(parsedRes.parsed?.liters ?
String(parsedRes.parsed.liters) : '');
      setMilkTemp(parsedRes.parsed?.temperature ?
String(parsedRes.parsed.temperature) : '');
    } else {
      // para estoque, exibir parsed.items em texto
      const items = parsedRes.parsed?.items || [];
      setStockItemsText(items.map((x: any) => `${x.name} - ${x.qty}
${x.unit}`).join('\n'));
    }

    setMsg('✅ Sugestão pronta. Confirme.');
    setStep(3);
  }

  async function voiceFillMock() {
    if (mode === 'MILK_DAILY') {
      setMilkLiters('800');
      setMsg('🎙 Ditado: 800 litros');
```

```
    } else {
      setMsg('🎙 Ditado disponível (estoque)');
    }
  }

  async function confirmCommit() {
    if (!logId) return;
    setMsg('⏳ Confirmando...');

    let confirmed: any = null;
    if (mode === 'MILK_DAILY') {
      confirmed = {
        date: milkDate,
        liters: Number(milkLiters),
        temperature: milkTemp ? Number(milkTemp) : null,
        meterType: 'MANUAL',
        meterValue: null
      };
    } else {
      // MVP: converte cada linha "nome - qty unit" em objeto
      const lines = stockItemsText.split('\n').map(s =>
s.trim()).filter(Boolean);
      confirmed = {
        items: lines.map((ln) => {
          // tentativa simples: "Nome - 2 SACOS"
          const m = ln.match(/^(.+?)\s*-
\s*(\d+(?:[.,]\d+)?)\s+(\w+)$/i);
          if (!m) return { name: ln, qty: 1, unit: 'UN' };
          const qty = Number(String(m[2]).replace(',', '.'));
          return { name: m[1].trim(), qty: Number.isFinite(qty) ? qty
: 1, unit: m[3].toUpperCase() };
        })
      };
    }

    const res = await commitManualLog(logId, {
      clientRequestId: uuid(),
      confirmed,
      poolId,
      poolContractId,
      producerTenantId
    });

    setMsg('✅ Lançado no sistema');
    setStep(4);

    // volta após 1s (opcional)
    setTimeout(() => navigation.goBack(), 800);
    return res;
  }

  return (
    <ScrollView style={{ padding: 16 }}>
      <Text style={{ fontSize: 26, fontWeight: '900' }}>Foto do
Papel</Text>
      <Text style={{ fontSize: 16, opacity: 0.8, marginTop: 6 }}>
        {mode === 'MILK_DAILY'
          ? 'Leite: Foto → OCR → Confirmar litros'
          : 'Estoque: Foto → OCR → Confirmar itens'}
      </Text>
```

```
      {!!msg && (
        <View style={{ marginTop: 12, padding: 12, borderWidth: 1,
borderRadius: 12 }}>
          <Text style={{ fontSize: 16 }}>{msg}</Text>
        </View>
      )}

      {step === 1 && (
        <BigActionButton icon="📷" title="TIRAR FOTO" subtitle="Do
caderno/planilha" onPress={takePhotoMock} />
      )}

      {step === 2 && (
        <>
          <BigActionButton icon="🔎" title="LER TEXTO (OCR)"
subtitle="Reconhecer escrita/foto" onPress={ocrMock} />
          <Text style={{ marginTop: 12, fontSize: 18, fontWeight:
'800' }}>Texto reconhecido</Text>
          <TextInput
            value={rawText}
            onChangeText={setRawText}
            placeholder="Se necessário, corrija aqui"
            multiline
            style={{ borderWidth: 2, borderRadius: 14, padding: 14,
fontSize: 18, minHeight: 120, marginTop: 10 }}
          />
          <BigActionButton icon="✅" title="CONTINUAR" subtitle="Gerar
sugestão automática" onPress={createAndParse} />
        </>
      )}

      {step === 3 && mode === 'MILK_DAILY' && (
        <>
          <Text style={{ marginTop: 12, fontSize: 18, fontWeight:
'900' }}>Confirmar</Text>

          <Text style={{ marginTop: 10, fontSize: 16, fontWeight:
'800' }}>Data</Text>
          <TextInput
            value={milkDate}
            onChangeText={setMilkDate}
            placeholder="05/02 ou 2026-02-05"
            style={{ borderWidth: 2, borderRadius: 14, padding: 14,
fontSize: 20, marginTop: 8 }}
          />

          <Text style={{ marginTop: 10, fontSize: 16, fontWeight:
'800' }}>Litros</Text>
          <TextInput
            value={milkLiters}
            onChangeText={setMilkLiters}
            keyboardType="numeric"
            placeholder="Ex.: 820"
            style={{ borderWidth: 2, borderRadius: 14, padding: 14,
fontSize: 24, marginTop: 8 }}
          />

          <Text style={{ marginTop: 10, fontSize: 16, fontWeight:
'800' }}>Temperatura (opcional)</Text>
          <TextInput
            value={milkTemp}
```

```
            onChangeText={setMilkTemp}
            keyboardType="numeric"
            placeholder="Ex.: 4,2"
            style={{ borderWidth: 2, borderRadius: 14, padding: 14,
fontSize: 20, marginTop: 8 }}
          />

          <BigActionButton icon="🎙" title="FALAR LITROS"
subtitle="Ex.: “oitocentos e vinte”" onPress={voiceFillMock} />
          <BigActionButton icon="✅" title="CONFIRMAR E LANÇAR"
subtitle="Cria tank check + ticket" onPress={confirmCommit}
disabled={!milkLiters} />
        </>
      )}

      {step === 3 && mode === 'STOCK_WITHDRAW' && (
        <>
          <Text style={{ marginTop: 12, fontSize: 18, fontWeight:
'900' }}>Confirmar itens</Text>
          <Text style={{ marginTop: 6, fontSize: 14, opacity: 0.8 }}>
            Revise as linhas. (No próximo passo você pode mapear “name
→ itemId”.)
          </Text>

          <TextInput
            value={stockItemsText}
            onChangeText={setStockItemsText}
            multiline
            style={{ borderWidth: 2, borderRadius: 14, padding: 14,
fontSize: 18, minHeight: 160, marginTop: 10 }}
          />

          <BigActionButton icon="✅" title="CONFIRMAR E LANÇAR"
subtitle="Cria baixa de estoque (OUT)" onPress={confirmCommit}
disabled={!stockItemsText.trim()} />
        </>
      )}

      {step === 4 && (
        <View style={{ marginTop: 20, padding: 16, borderWidth: 2,
borderRadius: 16 }}>
          <Text style={{ fontSize: 22, fontWeight: '900' }}>✅
Pronto!</Text>
          <Text style={{ fontSize: 16, opacity: 0.8, marginTop: 6 }}>
            Lançado no sistema com evidência e auditoria.
          </Text>
        </View>
      )}
    </ScrollView>
  );
}
```

---

3) Permissions (para seu JSON de seeds)

Adicionar:

- `manualLogs.read`

- `milk.routes.readStops` (que criamos)
- (já tinha) `milk.tankCheck.create`, `milk.collection.settle`, etc.

Sugestão rápida:

- `LOGISTICS`: `manualLogs.create`, `manualLogs.parse` (se precisar), `milk.routes.*`
- `OPERATOR/PRODUCER`: `manualLogs.*` para estoque e registros do campo
- `AUDITOR`: `manualLogs.read` (somente leitura)

---

4) Recomendação de "mapear estoque por name → itemId"

Para o MVP:

- você pode manter `item_aliases/{aliasId}`:
  - `"sal mineral" -> itemId: "item_sal_mineral"`
- no commit, se vier só `name`, o backend tenta mapear por alias e devolve "pendente" se não encontrar.

Se quiser, eu mando esse módulo de aliases também (é pequeno e fecha o estoque "por foto" perfeitamente).

Perfeito — segue o módulo **Item Aliases (name → itemId)** completo para fechar 100% o "estoque por foto/papel", com:

- coleção `item_aliases`
- endpoints CRUD mínimos
- auto-sugestão no commit do `manual-logs` (resolve "nome do papel" em `itemId`)
- regra de confiança (auto-aprova só se `confidence >= threshold`)
- auditoria + idempotência

---

1) Firestore

1.1 Catálogo de itens (já existe no seu modelo)

`items/{itemId}` (multi-tenant, com `tenantId`)

1.2 Aliases

`item_aliases/{aliasId}`

```
{
  "tenantId":"TENANT_X",
  "aliasId":"al_001",
  "normalized":"sal mineral",
  "display":"Sal Mineral",
```

```
  "itemId":"item_sal_mineral",
  "unitDefault":"SACO",
  "confidence": 1.0,
  "active": true,
  "createdAt":1730000000000,
  "createdBy":{"uid":"...","role":"MANAGER"}
}
```

Como normalizar

- minúsculo
- sem acentos
- remove espaços duplicados
- remove pontuação simples

---

2) Endpoints

2.1 Criar alias

```
POST /api/catalog/item-aliases
```

Body:

```
{
  "clientRequestId":"uuid",
  "display":"Sal Mineral",
  "alias":"sal mineral",
  "itemId":"item_sal_mineral",
  "unitDefault":"SACO"
}
```

2.2 Buscar alias (para autocomplete)

```
GET /api/catalog/item-aliases?q=sal
```

2.3 Desativar alias

```
POST /api/catalog/item-aliases/:id/deactivate
```

---

3) Implementação NestJS

3.1 Normalizador

```
// src/catalog/utils/normalize.util.ts
export function normalizeText(s: string) {
  return (s || '')
    .toLowerCase()
    .normalize('NFD')
    .replace(/[\u0300-\u036f]/g, '')  // remove acentos
    .replace(/[^a-z0-9\s]/g, ' ')     // tira pontuação
    .replace(/\s+/g, ' ')
    .trim();
}
```

3.2 DTOs

```
// src/catalog/dto/item-alias-create.dto.ts
import { IsNotEmpty, IsOptional, IsString } from 'class-validator';

export class ItemAliasCreateDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsString() @IsNotEmpty()
  display: string;

  @IsString() @IsNotEmpty()
  alias: string;

  @IsString() @IsNotEmpty()
  itemId: string;

  @IsOptional() @IsString()
  unitDefault?: string;
}
```

3.3 Service

```
// src/catalog/item-aliases.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';
import { normalizeText } from './utils/normalize.util';

@Injectable()
export class ItemAliasesService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async create(actor: any, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'catalog.itemAliases.create');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
      if (idem.exists) return idem.data()!.result;

      const normalized = normalizeText(dto.alias);
      if (!normalized) throw new Error('ALIAS_EMPTY');

      // evita duplicidade por tenant + normalized
      const dupSnap = await this.db.collection('item_aliases')
        .where('tenantId', '==', tenantId)
        .where('normalized', '==', normalized)
        .where('active', '==', true)
        .limit(1)
        .get();

      if (!dupSnap.empty) throw new Error('ALIAS_ALREADY_EXISTS');

      // valida itemId existe (opcional mas recomendado)
```

```
      const itemSnap = await
tx.get(this.db.collection('items').doc(dto.itemId));
      if (!itemSnap.exists) throw new Error('ITEM_NOT_FOUND');

      const aliasId = makeId('al');
      const ref = this.db.collection('item_aliases').doc(aliasId);

      tx.create(ref, {
        tenantId,
        aliasId,
        normalized,
        display: dto.display,
        itemId: dto.itemId,
        unitDefault: dto.unitDefault ?? null,
        confidence: 1.0,
        active: true,
        createdAt: now,
        createdBy: { uid: actor.uid, role: actor.role }
      });

      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        type: 'ITEM_ALIAS_CREATED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { aliasId, normalized, itemId: dto.itemId },
        ts: now
      });

      const out = { aliasId, normalized, itemId: dto.itemId };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'catalog.itemAliases.create', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });
      return out;
    });
  }

  async search(actor: any, q: string) {
    const tenantId = actor.tenantId;
    const nq = normalizeText(q);

    // MVP: busca por prefixo usando range
    // prefix = nq, prefixEnd = nq + '\uf8ff'
    const snap = await this.db.collection('item_aliases')
      .where('tenantId', '==', tenantId)
      .where('active', '==', true)
      .where('normalized', '>=', nq)
      .where('normalized', '<=', nq + '\uf8ff')
      .limit(30)
      .get();

    return { items: snap.docs.map(d => d.data()) };
  }

  async deactivate(actor: any, aliasId: string) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const ref = this.db.collection('item_aliases').doc(aliasId);
    const snap = await ref.get();
```

```
    if (!snap.exists) throw new Error('ALIAS_NOT_FOUND');

    const al = snap.data()!;
    if (al.tenantId !== tenantId) throw new Error('FORBIDDEN');

    await ref.set({ active: false, deactivatedAt: now, deactivatedBy:
{ uid: actor.uid, role: actor.role } }, { merge: true });

    await this.db.collection('audit_events').add({
      auditId: makeId('aud'),
      tenantId,
      type: 'ITEM_ALIAS_DEACTIVATED',
      actor: { uid: actor.uid, role: actor.role },
      meta: { aliasId },
      ts: now
    });

    return { aliasId, active: false };
  }

  async resolve(tenantId: string, name: string) {
    const normalized = normalizeText(name);
    if (!normalized) return null;

    const snap = await this.db.collection('item_aliases')
      .where('tenantId', '==', tenantId)
      .where('active', '==', true)
      .where('normalized', '==', normalized)
      .limit(1)
      .get();

    if (snap.empty) return null;
    return snap.docs[0].data();
  }
}
```

3.4 Controller

```
// src/catalog/item-aliases.controller.ts
import { Body, Controller, Get, Param, Post, Query, UseGuards } from
'@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { ItemAliasesService } from './item-aliases.service';
import { ItemAliasCreateDto } from './dto/item-alias-create.dto';

@Controller('/api/catalog/item-aliases')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class ItemAliasesController {
  constructor(private readonly svc: ItemAliasesService) {}

  @Post()
  @Roles('ADMIN','MANAGER')
  @Permissions('catalog.itemAliases.create')
  create(@Actor() actor: any, @Body() dto: ItemAliasCreateDto) {
    return this.svc.create(actor, dto);
  }
```

```
  @Get()

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','LOGISTICS
')
  @Permissions('catalog.itemAliases.read')
  search(@Actor() actor: any, @Query('q') q: string) {
    return this.svc.search(actor, q || '');
  }

  @Post('/:id/deactivate')
  @Roles('ADMIN','MANAGER')
  @Permissions('catalog.itemAliases.deactivate')
  deactivate(@Actor() actor: any, @Param('id') id: string) {
    return this.svc.deactivate(actor, id);
  }
}
```

---

4) Integrar no commit do `manual-logs` (estoque)

Você quer que no commit, se vier `{ name, qty, unit }`, o backend tente resolver `name → itemId`.

4.1 Patch no commit (trecho do ManualLogsService)

No bloco `STOCK_WITHDRAW`, antes de chamar `stockMoves.create`:

```
// dentro do commit (STOCK_WITHDRAW)
const confirmed = dto.confirmed || {};
const items = confirmed.items || [];

const resolvedMoves = [];
const unresolved = [];

for (const it of items) {
  if (it.itemId) {
    resolvedMoves.push({ itemId: it.itemId, qty: it.qty, unit: it.unit
});
    continue;
  }

  const alias = await this.itemAliases.resolve(tenantId, it.name);
  if (alias) {
    resolvedMoves.push({
      itemId: alias.itemId,
      qty: it.qty,
      unit: it.unit || alias.unitDefault || 'UN'
    });
  } else {
    unresolved.push(it);
  }
}

if (unresolved.length) {
  // não lança automático se tem item sem mapear
  throw new Error('UNRESOLVED_ITEMS_' +
unresolved.map((x:any)=>x.name).join('|'));
```

```
}

result = await this.stockMoves.create(actor, {
  clientRequestId: `manuallog_${logId}`,
  propertyId: log.propertyId,
  direction: 'OUT',
  moves: resolvedMoves,
  evidenceIds: [log.photoEvidenceId],
});
```

Melhor UX: em vez de “throw”, você pode salvar no manual_log um campo `needsMapping: unresolved[]` e devolver pro app abrir tela “mapear”.

---

5) Regras Firestore para `item_aliases`

```
match /item_aliases/{aliasId} {
  allow create: if signedIn()
    && request.auth.token.tenantId == request.resource.data.tenantId
    && hasRole(['ADMIN','MANAGER'])
    && hasPerm('catalog.itemAliases.create');

  allow get, list: if signedIn()
    && request.auth.token.tenantId == resource.data.tenantId
    && hasPerm('catalog.itemAliases.read');

  allow update: if signedIn()
    && request.auth.token.tenantId == resource.data.tenantId
    && hasRole(['ADMIN','MANAGER'])
    && hasPerm('catalog.itemAliases.deactivate')
    && request.resource.data.tenantId == resource.data.tenantId
    && request.resource.data.itemId == resource.data.itemId
    && request.resource.data.normalized == resource.data.normalized;

  allow delete: if false;
}
```

---

6) UX final (perfeito no campo)

- Foto do papel → OCR → sugere “sal mineral 2 sacos”
- Se já existe alias, vira itemId automaticamente
- Se não existe, aparece botão grande:
  - “✅ Criar apelido para este item”
  - escolhe item do catálogo (autocomplete)
  - salva alias
  - volta e confirma commit

Se você quiser, eu mando também:

- a tela mobile “Mapear Item Desconhecido” (super simples, 2 cliques)

- e seeds iniciais de aliases comuns do agro (sal mineral, ração, vacina, vermífugo, adubo, calcário, ureia, semente etc.).

A seguir vai **(1)** a tela "Mapear Item Desconhecido" (ultra simples, 2 cliques) e **(2)** seeds iniciais de aliases comuns do agro (bem prontos para começar).

---

1) Tela mobile: Mapear Item Desconhecido

Objetivo

Quando o commit do manual-log (estoque) retornar `UNRESOLVED_ITEMS_*`, o app abre esta tela:

- mostra o **nome** que veio do papel
- usuário escolhe **um item do catálogo** (autocomplete)
- salva alias (e já volta para tentar commit de novo)

1.1 API helpers (catalog items + alias)

```
// src/services/catalogApi.ts
import { api } from './api';

export async function searchItems(q: string) {
  return (await api.get('/api/catalog/items', { params: { q }
})).data;
  // você já tem Items; se não tiver esse endpoint, eu te passo também
}

export async function createItemAlias(payload: any) {
  return (await api.post('/api/catalog/item-aliases', payload)).data;
}
```

1.2 Tela

```
// src/screens/MapUnknownItemScreen.tsx
import React, { useMemo, useState } from 'react';
import { FlatList, Text, TextInput, View } from 'react-native';
import { v4 as uuid } from 'uuid';
import { BigActionButton } from '../ui/BigActionButton';
import { createItemAlias, searchItems } from '../services/catalogApi';

export function MapUnknownItemScreen({ navigation, route }: any) {
  const { unknownName, suggestedUnit } = route.params as {
    unknownName: string;
    suggestedUnit?: string;
  };

  const [qText, setQText] = useState('');
  const [results, setResults] = useState<any[]>([]);
  const [msg, setMsg] = useState('');

  async function doSearch(text: string) {
    setQText(text);
    if (text.trim().length < 2) return setResults([]);
    const r = await searchItems(text.trim());
    setResults(r.items || []);
  }
```

```
  async function chooseItem(item: any) {
    setMsg('⏳ Salvando apelido...');
    await createItemAlias({
      clientRequestId: uuid(),
      display: unknownName,
      alias: unknownName,
      itemId: item.itemId,
      unitDefault: suggestedUnit || item.unitDefault || 'UN'
    });
    setMsg('✅ Apelido criado. Pode lançar novamente.');
    setTimeout(() => navigation.goBack(), 600);
  }

  return (
    <View style={{ padding: 16, flex: 1 }}>
      <Text style={{ fontSize: 26, fontWeight: '900' }}>Mapear
item</Text>
      <Text style={{ fontSize: 16, opacity: 0.85, marginTop: 6 }}>
        Item do papel: <Text style={{ fontWeight: '900'
}}>{unknownName}</Text>
      </Text>

      {!!msg && (
        <View style={{ marginTop: 12, padding: 12, borderWidth: 1,
borderRadius: 12 }}>
          <Text style={{ fontSize: 16 }}>{msg}</Text>
        </View>
      )}

      <Text style={{ marginTop: 12, fontSize: 16, fontWeight: '800'
}}>Buscar no catálogo</Text>
      <TextInput
        value={qText}
        onChangeText={doSearch}
        placeholder="Digite 2 letras…"
        style={{ borderWidth: 2, borderRadius: 14, padding: 14,
fontSize: 18, marginTop: 8 }}
      />

      <FlatList
        style={{ marginTop: 12 }}
        data={results}
        keyExtractor={(it) => it.itemId}
        renderItem={({ item }) => (
          <BigActionButton
            icon="✅"
            title={item.name}
            subtitle={`ID: ${item.itemId}`}
            onPress={() => chooseItem(item)}
          />
        )}
      />
    </View>
  );
}
```

Como usar no fluxo do ManualLog

Quando o backend retornar erro `UNRESOLVED_ITEMS_...`:

- você pega o primeiro nome não resolvido e abre:
    - `navigation.navigate('MapUnknownItem', { unknownName: 'Sal mineral', suggestedUnit: 'SACO' })`

Depois volta e tenta `commit` novamente.

---

2) Endpoint de busca de itens (se ainda não tiver)

Se você ainda não tem, aqui vai o mais simples:

```
GET /api/catalog/items?q=...

// src/catalog/items-read.controller.ts
import { Controller, Get, Query, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { ItemsReadService } from './items-read.service';

@Controller('/api/catalog/items')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class ItemsReadController {
  constructor(private readonly svc: ItemsReadService) {}

  @Get()

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','LOGISTICS
','SUPPLIER')
  @Permissions('catalog.items.read')
  list(@Actor() actor: any, @Query('q') q: string) {
    return this.svc.search(actor, q || '');
  }
}
// src/catalog/items-read.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { normalizeText } from './utils/normalize.util';

@Injectable()
export class ItemsReadService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async search(actor: any, q: string) {
    const tenantId = actor.tenantId;
    const nq = normalizeText(q);

    // MVP: se você tiver campo normalizedName em items, fica
perfeito.
    // Se não tiver, crie e indexe.
    const snap = await this.db.collection('items')
```

```
      .where('tenantId', '==', tenantId)
      .where('normalizedName', '>=', nq)
      .where('normalizedName', '<=', nq + '\uf8ff')
      .limit(30)
      .get();

    return { items: snap.docs.map(d => d.data()) };
  }
}
```

---

3) Seeds iniciais (agro) — alias + itens

3.1 Modelo de seed (JSON)

Você pode rodar seeds no backend ou script. Formato:

```
{
  "items": [
    { "itemId":"item_sal_mineral", "name":"Sal mineral",
"unitDefault":"SACO", "normalizedName":"sal mineral" },
    { "itemId":"item_racao_bovina", "name":"Ração bovina",
"unitDefault":"SACO", "normalizedName":"racao bovina" },
    { "itemId":"item_vacina_aftosa", "name":"Vacina aftosa",
"unitDefault":"DOSE", "normalizedName":"vacina aftosa" }
  ],
  "aliases": [
    { "alias":"sal mineral", "itemId":"item_sal_mineral",
"unitDefault":"SACO" },
    { "alias":"sal", "itemId":"item_sal_mineral", "unitDefault":"SACO"
},
    { "alias":"racao", "itemId":"item_racao_bovina",
"unitDefault":"SACO" },
    { "alias":"vacina aftosa", "itemId":"item_vacina_aftosa",
"unitDefault":"DOSE" }
  ]
}
```

3.2 Lista sugerida (bem completa, comum no campo)

Nutrição

- Sal mineral (saco)
- Sal proteinado (saco)
- Núcleo mineral (kg/saco)
- Ração bovina (saco)
- Ração ovinos (saco)
- Ração aves (saco)
- Ração suínos (saco)
- Milho (kg/saco/ton)
- Farelo de soja (kg/saco/ton)
- Silagem (ton)
- Feno (fardo)

Sanidade

- Vacina aftosa (dose)
- Vacina raiva (dose)
- Vacina clostridioses (dose)
- Vermífugo (ml/dose)
- Carrapaticida (ml/l)
- Mosquicida (ml/l)
- Antibiótico (dose)
- Seringa (un)
- Agulha (un)

Insumos agrícolas

- Ureia (kg/saco)
- Calcário (ton)
- Gesso agrícola (ton)
- Adubo NPK (saco)
- Herbicida glifosato (L)
- Inseticida (L)
- Fungicida (L)
- Semente milho (saco)
- Semente soja (saco)

Silvicultura

- Muda eucalipto (un)
- Muda palmeira (un)
- Muda açaí (un)
- Muda pupunha (un)
- Insumo formicida (kg)
- Adubo plantio (kg)

Leite (insumos)

- Detergente alcalino (L)
- Detergente ácido (L)
- Papel toalha (pct)
- Luvas (cx)

---

4) Script de seed (NestJS simples)

```
// scripts/seedCatalog.ts (exemplo)
import * as admin from 'firebase-admin';
import { normalizeText } from '../src/catalog/utils/normalize.util';

async function run() {
  admin.initializeApp();
  const db = admin.firestore();
  const tenantId = 'TENANT_X';

  const items = [
```

```
    { itemId:'item_sal_mineral', name:'Sal mineral',
unitDefault:'SACO' },
    { itemId:'item_racao_bovina', name:'Ração bovina',
unitDefault:'SACO' },
    { itemId:'item_vacina_aftosa', name:'Vacina aftosa',
unitDefault:'DOSE' },
  ];

  for (const it of items) {
    await db.collection('items').doc(it.itemId).set({
      tenantId,
      ...it,
      normalizedName: normalizeText(it.name),
      active: true,
      createdAt: Date.now()
    }, { merge: true });
  }

  const aliases = [
    { alias:'sal mineral', itemId:'item_sal_mineral',
unitDefault:'SACO' },
    { alias:'sal', itemId:'item_sal_mineral', unitDefault:'SACO' },
    { alias:'racao bovina', itemId:'item_racao_bovina',
unitDefault:'SACO' },
    { alias:'racao', itemId:'item_racao_bovina', unitDefault:'SACO' },
    { alias:'vacina aftosa', itemId:'item_vacina_aftosa',
unitDefault:'DOSE' },
    { alias:'aftosa', itemId:'item_vacina_aftosa', unitDefault:'DOSE'
},
  ];

  for (const al of aliases) {
    const aliasId = 'al_' +
normalizeText(al.alias).replace(/\s/g,'_');
    await db.collection('item_aliases').doc(aliasId).set({
      tenantId,
      aliasId,
      normalized: normalizeText(al.alias),
      display: al.alias,
      itemId: al.itemId,
      unitDefault: al.unitDefault,
      confidence: 1.0,
      active: true,
      createdAt: Date.now()
    }, { merge: true });
  }

  console.log('Seeds ok');
}

run().catch(e => { console.error(e); process.exit(1); });
```

---

5) Fechamento: como fica perfeito no campo

✅ Foto do papel → OCR → itens sugeridos
✅ se algum item desconhecido:

- abre tela “Mapear item”

- salva alias
- volta e lança

E pronto: **sem digitação**, com lastro e auditoria.

Se você quiser continuar, o próximo módulo natural é:

- **Tabela de conversão de unidades** (SACO→KG, TON→KG, DOSE→ML), para não errar saldo e custo.

Fechado — segue o módulo **Conversão de Unidades** (SACO→KG, TON→KG, CX→UN, etc.) completo, com:

- coleção `unit_conversions`
- resolução automática na movimentação de estoque
- bloqueio se unidade não reconhecida
- UI simples para cadastrar conversões
- seeds agro (kg/ton/saco/fardo/caixa/dose/L/ml)

---

1) Firestore

1.1 unit_conversions/{convId}

Cada conversão é **por item** (porque "1 saco" pode ter pesos diferentes conforme produto).

```
{
  "tenantId":"TENANT_X",
  "convId":"uc_item_sal_mineral_SACO_KG",
  "itemId":"item_sal_mineral",
  "fromUnit":"SACO",
  "toUnit":"KG",
  "multiplier": 25,
  "active": true,
  "source":"DEFAULT|MANUAL|NF",
  "createdAt":1730000000000,
  "createdBy":{"uid":"...","role":"MANAGER"}
}
```

Recomendação:

- Defina um **unitBase** por item (ex.: `KG` ou `UN` ou `L`)
- Toda movimentação converte para `unitBase` antes de atualizar saldo.

---

2) Ajuste no Item

Em `items/{itemId}`:

```
{
  "unitBase":"KG",
  "unitDefault":"SACO"
}
```

---

3) Serviço de conversão (NestJS)

3.1 UnitConversionService

```
// src/catalog/unit-conversions.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

@Injectable()
export class UnitConversionsService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async getConversion(tenantId: string, itemId: string, fromUnit:
string, toUnit: string) {
    if (fromUnit === toUnit) return { multiplier: 1 };

    const convId = `uc_${itemId}_${fromUnit}_${toUnit}`;
    const ref = this.db.collection('unit_conversions').doc(convId);
    const snap = await ref.get();
    if (snap.exists) {
      const d = snap.data()!;
      if (d.tenantId !== tenantId || !d.active) return null;
      return { multiplier: Number(d.multiplier || 0) };
    }

    // fallback: tenta lookup por query (caso convId não seja
determinístico)
    const q = await this.db.collection('unit_conversions')
      .where('tenantId', '==', tenantId)
      .where('itemId', '==', itemId)
      .where('fromUnit', '==', fromUnit)
      .where('toUnit', '==', toUnit)
      .where('active', '==', true)
      .limit(1)
      .get();

    if (q.empty) return null;
    const d = q.docs[0].data()!;
    return { multiplier: Number(d.multiplier || 0) };
  }

  async toBaseUnit(tenantId: string, item: any, qty: number, unit:
string) {
    const base = item.unitBase || unit;
    if (unit === base) return { qtyBase: qty, unitBase: base };

    const conv = await this.getConversion(tenantId, item.itemId, unit,
base);
    if (!conv || !conv.multiplier) {
```

```
      throw new
Error(`UNIT_CONVERSION_NOT_FOUND_${item.itemId}_${unit}_TO_${base}`);
    }

    return { qtyBase: qty * conv.multiplier, unitBase: base };
  }
}
```

---

4) Aplicar na movimentação de estoque

Na sua `StockMovementsService.create`, antes de atualizar saldo:

4.1 Exemplo de normalização

```
// trecho dentro stockMoves.create(...)
for (const mv of dto.moves) {
  const itemSnap = await
tx.get(db.collection('items').doc(mv.itemId));
  if (!itemSnap.exists) throw new Error('ITEM_NOT_FOUND');

  const item = itemSnap.data()!;
  if (item.tenantId !== tenantId) throw new Error('FORBIDDEN');

  const qty = Number(mv.qty || 0);
  const unit = String(mv.unit || item.unitDefault || item.unitBase ||
'UN').toUpperCase();

  // converte para base
  const { qtyBase, unitBase } = await
this.unitConv.toBaseUnit(tenantId, item, qty, unit);

  // agora atualiza saldo usando qtyBase / unitBase
  // balanceBefore/After, etc.
}
```

Resultado: o saldo do estoque fica sempre coerente (ex.: sempre em KG).

---

5) Endpoints para cadastrar conversões

5.1 Criar conversão

```
POST /api/catalog/unit-conversions

{
  "clientRequestId":"uuid",
  "itemId":"item_sal_mineral",
  "fromUnit":"SACO",
  "toUnit":"KG",
  "multiplier":25
}
```

5.2 Buscar conversões do item

```
GET /api/catalog/unit-conversions?itemId=item_sal_mineral
```

5.3 Desativar conversão

```
POST /api/catalog/unit-conversions/:id/deactivate
```

---

6) Controller + Service (CRUD mínimo)

```
// src/catalog/unit-conversions.controller.ts
import { Body, Controller, Get, Param, Post, Query, UseGuards } from
'@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { UnitConversionsCrudService } from './unit-
conversions.crud.service';

@Controller('/api/catalog/unit-conversions')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class UnitConversionsController {
  constructor(private readonly svc: UnitConversionsCrudService) {}

  @Post()
  @Roles('ADMIN','MANAGER')
  @Permissions('catalog.unitConversions.create')
  create(@Actor() actor: any, @Body() dto: any) {
    return this.svc.create(actor, dto);
  }

  @Get()
  @Roles('ADMIN','MANAGER')
  @Permissions('catalog.unitConversions.read')
  list(@Actor() actor: any, @Query('itemId') itemId: string) {
    return this.svc.list(actor, itemId);
  }

  @Post('/:id/deactivate')
  @Roles('ADMIN','MANAGER')
  @Permissions('catalog.unitConversions.deactivate')
  deactivate(@Actor() actor: any, @Param('id') id: string) {
    return this.svc.deactivate(actor, id);
  }
}
// src/catalog/unit-conversions.crud.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class UnitConversionsCrudService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async create(actor: any, dto: any) {
```

```
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'catalog.unitConversions.create');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
      if (idem.exists) return idem.data()!.result;

      const fromUnit = String(dto.fromUnit).toUpperCase();
      const toUnit = String(dto.toUnit).toUpperCase();
      const multiplier = Number(dto.multiplier);

      if (!dto.itemId || !fromUnit || !toUnit ||
!Number.isFinite(multiplier) || multiplier <= 0) {
        throw new Error('INVALID_CONVERSION');
      }

      const itemSnap = await
tx.get(this.db.collection('items').doc(dto.itemId));
      if (!itemSnap.exists) throw new Error('ITEM_NOT_FOUND');
      const item = itemSnap.data()!;
      if (item.tenantId !== tenantId) throw new Error('FORBIDDEN');

      const convId = `uc_${dto.itemId}_${fromUnit}_${toUnit}`;
      const ref = this.db.collection('unit_conversions').doc(convId);

      tx.set(ref, {
        tenantId,
        convId,
        itemId: dto.itemId,
        fromUnit,
        toUnit,
        multiplier,
        active: true,
        source: dto.source ?? 'MANUAL',
        createdAt: now,
        createdBy: { uid: actor.uid, role: actor.role }
      }, { merge: true });

      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        type: 'UNIT_CONVERSION_CREATED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { convId, itemId: dto.itemId, fromUnit, toUnit,
multiplier },
        ts: now
      });

      const out = { convId, itemId: dto.itemId, fromUnit, toUnit,
multiplier };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'catalog.unitConversions.create', clientRequestId:
dto.clientRequestId, createdAt: now, result: out });
      return out;
    });
  }
```

```
  async list(actor: any, itemId: string) {
    const tenantId = actor.tenantId;
    const snap = await this.db.collection('unit_conversions')
      .where('tenantId', '==', tenantId)
      .where('itemId', '==', itemId)
      .where('active', '==', true)
      .limit(200)
      .get();

    return { items: snap.docs.map(d => d.data()) };
  }

  async deactivate(actor: any, id: string) {
    const now = Date.now();
    const tenantId = actor.tenantId;
    const ref = this.db.collection('unit_conversions').doc(id);
    const snap = await ref.get();
    if (!snap.exists) throw new Error('CONVERSION_NOT_FOUND');
    const d = snap.data()!;
    if (d.tenantId !== tenantId) throw new Error('FORBIDDEN');

    await ref.set({ active: false, deactivatedAt: now, deactivatedBy:
{ uid: actor.uid, role: actor.role } }, { merge: true });
    return { convId: id, active: false };
  }
}
```

---

7) Seeds agro (conversões típicas)

Atenção: por item varia. Aqui são *defaults* recomendados para começar e depois ajustar.

Exemplos (JSON)

```
{
  "unitConversions": [
    { "itemId":"item_sal_mineral", "fromUnit":"SACO", "toUnit":"KG",
"multiplier":25 },
    { "itemId":"item_racao_bovina", "fromUnit":"SACO", "toUnit":"KG",
"multiplier":40 },
    { "itemId":"item_adubo_npk", "fromUnit":"SACO", "toUnit":"KG",
"multiplier":50 },
    { "itemId":"item_calcario", "fromUnit":"TON", "toUnit":"KG",
"multiplier":1000 },
    { "itemId":"item_gesso_agricola", "fromUnit":"TON", "toUnit":"KG",
"multiplier":1000 },
    { "itemId":"item_glifosato", "fromUnit":"L", "toUnit":"ML",
"multiplier":1000 },
    { "itemId":"item_vacina_aftosa", "fromUnit":"CX", "toUnit":"DOSE",
"multiplier":50 }
  ]
}
```

---

8) UI simples para cadastrar conversões (opcional, mas útil)

Tela “Conversões do item”:

- buscar item
- lista conversões
- botão “Adicionar”
- 3 campos grandes: “De”, “Para”, “Multiplicador”

Se você quiser, eu mando essa tela também.

Próximo módulo que fecha a cadeia (e evita erro financeiro)

Custo médio / lote de entrada (NF/compra) + rastreio de lote para split e auditoria.

Fechado — agora vamos fechar a cadeia com o que você pediu lá atrás (rastreio por identidade, NF, split por fornecedor e custo), com um módulo de **Lote/Entrada (NF ou qualquer rastro)** + **custo médio** + **split automático no PDV**.

Vou estruturar em 6 partes:

1. Modelo de **Lote (Lot)** e **Entrada (Inbound)**
2. Como calcular **custo médio** (e alternativa FIFO)
3. Como vender no **PDV** com **split por fornecedor** (mesmo com vários fornecedores na mesma cesta)
4. Regras (RulesEngine) e auditoria
5. Endpoints NestJS mínimos
6. Seeds/indices recomendados

---

1) Modelo de dados

1.1 Identidade obrigatória do produto

Todo produto cadastrado carrega uma identidade:

- `itemId` (produto catálogo)
- `lotId` (lote de entrada; se não tiver NF, pode ser “lote manual”)
- `trace` (qualquer rastro: NF, código, QR, RFID, etc.)

Coleção: lots/{lotId}

```
{
  "tenantId": "TENANT_STORE",
  "lotId": "lot_2026_02_05_nf123_itemA_01",
  "itemId": "item_sal_mineral",
  "supplierTenantId": "TENANT_SUPPLIER_77",
  "trace": {
```

```
    "sourceType": "NF|MANUAL|IMPORT|API",
    "docKey": "NF-352602-123456789-001",
    "externalCode": "QR123|RFID999|BARCODE...",
    "batch": "LoteFabricante123"
  },
  "pricing": {
    "unitBase": "KG",
    "qtyBaseIn": 1000,
    "unitCostCentsBase": 350,
    "totalCostCents": 350000
  },
  "stock": {
    "qtyBaseAvailable": 1000,
    "reservedBase": 0
  },
  "createdAt": 1730000000000,
  "createdBy": { "uid": "u1", "role": "MANAGER" },
  "active": true
}
```

1.2 Entrada (NF / inbound)

inbound_docs/{inboundId}

```
{
  "tenantId":"TENANT_STORE",
  "inboundId":"inb_nf_3526_123",
  "propertyId":"store_01",
  "supplierTenantId":"TENANT_SUPPLIER_77",
  "type":"NF|MANUAL",
  "docKey":"NF-352602-123456789-001",
  "date":"2026-02-05",
  "items":[
    {
      "itemId":"item_sal_mineral",
      "qty":40,
      "unit":"SACO",
      "qtyBase":1000,
      "unitBase":"KG",
      "unitCostCentsBase":350,
      "lotId":"lot_..."
    }
  ],
  "status":"POSTED",
  "evidenceIds":["evg_nf_photo"],
  "createdAt":1730000000000
}
```

Observação: **NF é documento fiscal**. A foto é só para facilitar cadastro; a NF (XML/chave) é o lastro fiscal.

---

2) Custo médio (e alternativa FIFO)

2.1 Custo médio ponderado (recomendado para MVP)

Manter por item um “ledger”:

item_costs/{itemId}

```
{
  "tenantId":"TENANT_STORE",
  "itemId":"item_sal_mineral",
  "unitBase":"KG",
  "avgCostCentsBase": 340,
  "qtyBaseOnHand": 2500,
  "updatedAt": 1730000000000
}
```

Atualização no inbound (entrada)

Se tinha:

- `Q0` e `C0`
  Entrou:
- `Q1` com custo unitário `C1`
  Novo custo médio:
- `Cavg = (Q0*C0 + Q1*C1) / (Q0+Q1)`
  Novo estoque:
- `Q = Q0+Q1`

Saída (venda)

Na saída, você pode:

- **(A)** baixar pelo custo médio `Cavg` (simples)
- **(B)** baixar por lote (mais fiel para split/fornecedor; recomendado)

2.2 FIFO (opcional)

FIFO exige consumir lotes por ordem (primeiro que entra, primeiro que sai).
Como você já quer rastrear por lote para split, FIFO pode ser ativado depois.

---

3) PDV com split por fornecedor na mesma cesta

Aqui está o coração do seu requisito:

“ao vender vários produtos da mesma cesta de diferentes fornecedores deve identificar o fornecedor no pagamento e dividir split; rastreio da NF/entrada em estoque; vale para toda cadeia”.

3.1 Carrinho sempre referencia lotes

No PDV, cada item vendido precisa ter:

- `itemId`

- `lotAllocations`: lista de lotes consumidos (e quanto de cada)

Exemplo: vendeu 10kg de sal mineral, consumiu:

- `lotA 6kg` (Fornecedor 77)
- `lotB 4kg` (Fornecedor 12)

sales/{saleId}

```
{
  "tenantId":"TENANT_STORE",
  "saleId":"sale_0001",
  "pdvId":"pdv_01",
  "buyer": { "customerId":"cust_01" },
  "items":[
    {
      "itemId":"item_sal_mineral",
      "qtyBase":10,
      "unitBase":"KG",
      "unitPriceCentsBase":500,
      "lotAllocations":[
        { "lotId":"lotA", "supplierTenantId":"TENANT_SUPPLIER_77",
"qtyBase":6 },
        { "lotId":"lotB", "supplierTenantId":"TENANT_SUPPLIER_12",
"qtyBase":4 }
      ]
    },
    {
      "itemId":"item_vacina_aftosa",
      "qtyBase":2,
      "unitBase":"DOSE",
      "unitPriceCentsBase":1200,
      "lotAllocations":[
        { "lotId":"lotC", "supplierTenantId":"TENANT_SUPPLIER_77",
"qtyBase":2 }
      ]
    }
  ],
  "amountCents": 0,
  "splitPlan": {
    "targets":[
      { "supplierTenantId":"TENANT_SUPPLIER_77", "amountCents": 0 },
      { "supplierTenantId":"TENANT_SUPPLIER_12", "amountCents": 0 }
    ]
  },
  "escrow": { "escrowId":"esc_sale_0001", "status":"REQUESTED" },
  "status":"PENDING_PAYMENT|PAID|CANCELLED",
  "createdAt":1730000000000
}
```

3.2 Como calcular split

Regra simples e correta:

- Para cada item:
  - `revenueItem = unitPrice * qty`
  - rateia a receita por lote proporcional ao `qtyBase` consumido
- Soma por `supplierTenantId`

- (Opcional) retém taxa do marketplace/loja antes de dividir

E o split só executa após pagamento escrow.

---

4) Regras (RulesEngine) e auditoria

4.1 Regras obrigatórias

- `IdentityRequiredRule`: todo item vendido deve ter `lotAllocations` válidas
- `LotHasStockRule`: lote tem `qtyBaseAvailable >= qtyBaseAllocated`
- `SplitPlanMatchesLotsRule`: splitPlan deve bater com lotAllocations
- `EscrowRequiredForReservationRule`: reserva de estoque só após escrow (como você pediu)
- `CancellationPenaltyRule`: cancelou → multa e auditoria

4.2 Auditoria (append-only)

Cada evento:

- inbound posted
- lot created
- sale created
- escrow paid
- split executed
- stock movement out

Tudo com hash/evidências conforme seu núcleo.

---

5) Endpoints NestJS mínimos

5.1 Entrada (NF / manual)

- `POST /api/inbound` (cria draft)
- `POST /api/inbound/:id/post` (gera lots + movimenta estoque + atualiza custo médio)

5.2 Lotes e alocação

- `GET /api/lots?itemId=...&propertyId=...` (lista lotes com saldo)
- `POST /api/lots/allocate` (aloca automaticamente por FIFO ou por regras)

Payload exemplo:

```
{
  "clientRequestId":"uuid",
  "propertyId":"store_01",
```

```
  "itemId":"item_sal_mineral",
  "qtyBase":10,
  "strategy":"FIFO"
}
```

Resposta:

```
{
  "allocations":[

{"lotId":"lotA","supplierTenantId":"TENANT_SUPPLIER_77","qtyBase":6},

{"lotId":"lotB","supplierTenantId":"TENANT_SUPPLIER_12","qtyBase":4}
  ]
}
```

5.3 Venda e escrow

- `POST /api/sales` (cria venda PENDING_PAYMENT, calcula splitPlan)
- `POST /api/sales/:id/pay` (cria escrow Asaas)
- Webhook Asaas → `POST /api/webhooks/asaas` (marca PAID, executa split, baixa estoque)

---

6) Pontos críticos que deixam "perfeito" para o seu caso

1. **Todo produto sempre tem identidade**: itemId + lotId + qualquer rastro (NF, QR, RFID, código interno).
2. **Split deriva do lote** (não só da NF). NF é um tipo de rastro; se tiver outro rastro, serve igual.
3. **Mistura de ativos/mercadorias**:
   - identificável → por identidade (brinco/RFID/QR/lote unitário)
   - não separável → proporcional por quota/NF (isso vira um "pool lot" com quotas, que dá para fazer depois se você quiser)
4. **Reserva só após pagamento**: PDV só "segura" estoque quando escrow está HELD.
5. **Cancelamento com multa**: evento de multa + retenção no escrow.

Próximo (se você disser "sim" de novo)

Eu monto o **serviço de alocação de lotes (FIFO)** + **cálculo de split** em código NestJS (bem direto), e o webhook Asaas "PAID → split → baixa estoque".

Sim — para ficar **100% fiel ao fornecedor** (escrow + split), a regra-mãe é:

✅ **o split é calculado e travado a partir dos LOTES (lotAllocations)** e **o pagamento nunca passa "livre" pela loja**: ele já nasce com **destinos (suppliers)** e só libera conforme evidência/estado.

Abaixo está o modelo completo (bem objetivo e implementável).

---

1) Verdade única: lote → fornecedor → split

Estrutura que garante fidelidade

- Cada item vendido no PDV **obrigatoriamente** tem `lotAllocations[]`
- Cada `lotId` tem `supplierTenantId` imutável
- O `splitPlan` é derivado **apenas** de `lotAllocations` (nunca do "item genérico")

Isso impede:

- "misturar" fornecedor depois
- "editar" split após pagamento
- pagar fornecedor errado

---

2) Escrow fiel por design

Opção A (mais forte): Escrow com split nativo no PSP

Se o provedor (ex.: Asaas) suportar split/transferência automática, você cria a cobrança com:

- `amount`
- `splitRules` por `supplierWalletId` (subcontas)
- e registra o "espelho" no Firestore

Resultado: mesmo que a loja tente fraudar, o dinheiro já está roteado.

Opção B (robusta também): Escrow lógico + "sub-escrows" por fornecedor

Se o PSP não fizer split nativo:

- você cria 1 cobrança (escrow)
- na confirmação de pagamento, seu backend executa transferências/payouts **por fornecedor**
- tudo travado por auditoria e idempotência

Resultado: dinheiro fica retido até seu motor liberar, mas o split continua fiel.

---

3) Cálculo do split fiel ao fornecedor (por lote)

Regra recomendada

O “quem recebe” é por lote; o “quanto recebe” pode ser:

(1) Receita proporcional (simples)
Fornecedor recebe proporcional ao **volume** do lote consumido naquele item, aplicado sobre a **receita**.

(2) Recomendado para fidelidade total (mais justo):
Fornecedor recebe **o preço dele** (custo do lote + markup acordado) e a loja fica com a margem.

Isso evita a loja “inflar” preço e distorcer pagamento do fornecedor.

Eu recomendo o (2) para o seu caso.

---

4) Modelo de dados mínimo para split fiel

lot (imutável no fornecedor)

```
lots/{lotId}

{
  "tenantId":"TENANT_STORE",
  "lotId":"lotA",
  "itemId":"item_sal_mineral",
  "supplierTenantId":"TENANT_SUPPLIER_77",
  "supplierTerms": {
    "settlementMode":"FIXED_PRICE",
    "unitPayCentsBase": 420,
    "unitBase":"KG"
  },
  "stock": { "qtyBaseAvailable": 1000, "reservedBase": 0 },
  "trace": { "sourceType":"NF", "docKey":"..." }
}
```

sale item com lot allocations travadas

```
sales/{saleId}

{
  "items":[
    {
      "itemId":"item_sal_mineral",
      "qtyBase":10,
      "unitBase":"KG",
      "unitPriceCentsBase":500,
      "lotAllocations":[
```

```
{"lotId":"lotA","supplierTenantId":"TENANT_SUPPLIER_77","qtyBase":6,"u
nitPayCentsBase":420},

{"lotId":"lotB","supplierTenantId":"TENANT_SUPPLIER_12","qtyBase":4,"u
nitPayCentsBase":410}
      ],
      "pricingModel":"FIXED_SUPPLIER_PRICE"
    }
  ]
}
```


splitPlan derivado (imutável depois de “PAID”)


```
{
  "splitPlan":{
    "targets":[
      {"supplierTenantId":"TENANT_SUPPLIER_77","amountCents":2520},
      {"supplierTenantId":"TENANT_SUPPLIER_12","amountCents":1640}
    ],
    "storeAmountCents": 840,
    "policy": "SUPPLIER_FIXED_PRICE"
  }
}
```


---

5) Algoritmo (fiel ao fornecedor)

5.1 Gerar splitPlan a partir de lotAllocations

- Para cada allocation:
  - `paySupplier = qtyBase * unitPayCentsBase`
- Soma por fornecedor
- Total do fornecedor = soma allocations dele
- Total venda = soma (qtyBase * unitPrice)
- Loja = Total venda − Soma fornecedores − taxas (se houver)

Código (função pura)

```
type LotAlloc = { lotId: string; supplierTenantId: string; qtyBase:
number; unitPayCentsBase: number };
type SaleItem = { qtyBase: number; unitPriceCentsBase: number;
lotAllocations: LotAlloc[] };

export function computeSplitFixedSupplier(items: SaleItem[]) {
  const supplierMap = new Map<string, number>();
  let totalSale = 0;
  let totalSuppliers = 0;

  for (const it of items) {
    totalSale += Math.round(it.qtyBase * it.unitPriceCentsBase);

    for (const al of it.lotAllocations) {
      const amt = Math.round(al.qtyBase * al.unitPayCentsBase);
      supplierMap.set(al.supplierTenantId,
(supplierMap.get(al.supplierTenantId) || 0) + amt);
      totalSuppliers += amt;
```

```
    }
  }

  const storeAmount = totalSale - totalSuppliers;
  if (storeAmount < 0) throw new Error('SPLIT_NEGATIVE_STORE_MARGIN');

  return {
    totalSaleCents: totalSale,
    targets: Array.from(supplierMap.entries()).map(([supplierTenantId,
amountCents]) => ({ supplierTenantId, amountCents })),
    storeAmountCents: storeAmount
  };
}
```

---

6) Travas obrigatórias (para impedir qualquer "jeitinho")

6.1 No "criar venda"

Regras do RulesEngine:

1. `IdentityRequiredRule`: cada item tem `lotAllocations` e soma `qtyBase == total do item`
2. `LotSupplierImmutableRule`: `supplierTenantId` do allocation deve bater com o do lot
3. `LotHasStockRule`: lote tem `available >= qty`
4. `SplitEqualsAllocRule`: splitPlan calculado deve bater (recalcular e comparar) antes de gravar

6.2 No "pagar"

- Ao iniciar pagamento:
  - muda venda para `PENDING_PAYMENT`
  - reserva lotes: `reservedBase += qtyBase` (não baixa ainda)
- Somente quando `escrow = HELD/PAID`:
  - baixa estoque (OUT)
  - executa split/payout por fornecedor
  - muda venda para `PAID`

6.3 Após "PAID"

- `lotAllocations` e `splitPlan` viram **imutáveis**
- tentativa de update bloqueia e registra em auditoria

---

7) Entrega/evidência e liberação (quando não é PDV imediato)

Você citou casos como boitel/ativos/entrega com evidência.

Aí o fluxo fica:

- Pagou escrow → fica `HELD`
- Só libera fornecedor quando:
  - evidência de entrega validada (foto + GPS + assinatura, etc.)
  - máquina de estados muda para `DELIVERED_VERIFIED`
- Se houver disputa, fica retido e auditor decide.

Isso é exatamente o núcleo Ciclo+.

---

8) O que eu implementaria agora (ordem ideal)

1. `POST /api/lots/allocate` (FIFO por item/propriedade)
2. `POST /api/sales` (gera venda + splitPlan FIXED_SUPPLIER_PRICE)
3. `POST /api/sales/:id/pay` (escrow + reserva)
4. Webhook `PAID` → baixa estoque + split/payout por fornecedor + auditoria

Se você disser "sim", eu mando esses **4 serviços em NestJS** (com transação Firestore + idempotência + regras) já no padrão que você está usando.

A seguir estão os **4 serviços completos em NestJS** (com Firestore transaction + idempotência + regras "fiéis ao fornecedor" via lot → split), no padrão que você já está usando:

1. `POST /api/lots/allocate` (FIFO)
2. `POST /api/sales` (cria venda + trava lotAllocations + gera splitPlan FIXED_SUPPLIER_PRICE)
3. `POST /api/sales/:id/pay` (cria escrow, reserva estoque do lote)
4. Webhook `PAID` (baixa estoque + executa split/payout + auditoria)

Vou manter **FIXED_SUPPLIER_PRICE** como política padrão (fornecedor recebe "o dele", loja fica com a margem).
Se quiser depois, adicionamos o modo "REVENUE_PROPORTIONAL".

---

0) Convenções rápidas

- Coleções:
  - `lots`
  - `sales`
  - `escrows_sales`
  - `audit_events`
  - `idempotency`
- Campos esperados:
  - `lots.stock.qtyBaseAvailable`
  - `lots.stock.reservedBase`
  - `lots.supplierTerms.unitPayCentsBase` (quanto o fornecedor deve receber por unidade base)
- Idempotência:
  - `clientRequestId` obrigatório em todos os POST

1) POST /api/lots/allocate (FIFO)

DTO

```
// src/lots/dto/lot-allocate.dto.ts
import { IsIn, IsNotEmpty, IsNumber, IsString } from 'class-
validator';

export class LotAllocateDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsString() @IsNotEmpty()
  propertyId: string;

  @IsString() @IsNotEmpty()
  itemId: string;

  @IsNumber()
  qtyBase: number;

  @IsIn(['FIFO'])
  strategy: 'FIFO';
}
```

Service

```
// src/lots/lots-allocation.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';

@Injectable()
export class LotsAllocationService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async allocate(actor: any, dto: any) {
    const tenantId = actor.tenantId;
    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'lots.allocate');
    const idemRef = this.db.collection('idempotency').doc(key);

    // leitura fora de tx (ok) — apenas sugestão; a reserva real
acontece no PAY.
    const snap = await this.db.collection('lots')
      .where('tenantId', '==', tenantId)
      .where('propertyId', '==', dto.propertyId)
      .where('itemId', '==', dto.itemId)
      .where('active', '==', true)
      .orderBy('createdAt', 'asc')
      .limit(200)
      .get();

    let need = Number(dto.qtyBase || 0);
    if (!(need > 0)) throw new Error('INVALID_QTY');

    const allocs: any[] = [];
    for (const d of snap.docs) {
      if (need <= 0) break;
      const lot = d.data();
```

```
      const avail = Number(lot.stock?.qtyBaseAvailable || 0) -
Number(lot.stock?.reservedBase || 0);
      if (avail <= 0) continue;

      const take = Math.min(avail, need);
      allocs.push({
        lotId: lot.lotId,
        supplierTenantId: lot.supplierTenantId,
        qtyBase: take,
        unitPayCentsBase: Number(lot.supplierTerms?.unitPayCentsBase
|| 0)
      });
      need -= take;
    }

    if (need > 0) throw new Error('INSUFFICIENT_STOCK_FOR_FIFO');

    // idempotência só para consistência do retorno (não escreve nada
crítico)
    const idemSnap = await idemRef.get();
    if (idemSnap.exists) return idemSnap.data()!.result;

    const out = { allocations: allocs };
    await idemRef.set({ tenantId, uid: actor.uid, scope:
'lots.allocate', clientRequestId: dto.clientRequestId, createdAt:
Date.now(), result: out });
    return out;
  }
}
```

Controller

```
// src/lots/lots.controller.ts
import { Body, Controller, Post, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { LotsAllocationService } from './lots-allocation.service';
import { LotAllocateDto } from './dto/lot-allocate.dto';

@Controller('/api/lots')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class LotsController {
  constructor(private readonly alloc: LotsAllocationService) {}

  @Post('/allocate')
  @Roles('ADMIN','MANAGER','OPERATOR')
  @Permissions('lots.allocate')
  allocate(@Actor() actor: any, @Body() dto: LotAllocateDto) {
    return this.alloc.allocate(actor, dto);
  }
}
```

---

2) POST /api/sales (cria venda + splitPlan fiel)

DTO

```
// src/sales/dto/sale-create.dto.ts
import { IsArray, IsNotEmpty, IsString, ValidateNested } from 'class-
validator';
import { Type } from 'class-transformer';

class LotAllocationDto {
  @IsString() @IsNotEmpty() lotId: string;
  @IsString() @IsNotEmpty() supplierTenantId: string;
  qtyBase: number;
}

class SaleItemDto {
  @IsString() @IsNotEmpty() itemId: string;
  qtyBase: number;
  unitBase: string;
  unitPriceCentsBase: number;

  @IsArray()
  @ValidateNested({ each: true })
  @Type(() => LotAllocationDto)
  lotAllocations: LotAllocationDto[];
}

export class SaleCreateDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsString() @IsNotEmpty()
  propertyId: string;

  @IsArray()
  @ValidateNested({ each: true })
  @Type(() => SaleItemDto)
  items: SaleItemDto[];
}
```

Split function (FIXED_SUPPLIER_PRICE)

```
// src/sales/utils/computeSplit.ts
export function computeSplitFixedSupplier(items: any[]) {
  const supplierMap = new Map<string, number>();
  let totalSale = 0;
  let totalSuppliers = 0;

  for (const it of items) {
    totalSale += Math.round(Number(it.qtyBase) *
Number(it.unitPriceCentsBase));

    for (const al of it.lotAllocations) {
      const amt = Math.round(Number(al.qtyBase) *
Number(al.unitPayCentsBase));
      supplierMap.set(al.supplierTenantId,
(supplierMap.get(al.supplierTenantId) || 0) + amt);
      totalSuppliers += amt;
    }
  }

  const storeAmount = totalSale - totalSuppliers;
  if (storeAmount < 0) throw new Error('SPLIT_NEGATIVE_STORE_MARGIN');
```

```
  return {
    totalSaleCents: totalSale,
    targets: Array.from(supplierMap.entries()).map(([supplierTenantId,
amountCents]) => ({ supplierTenantId, amountCents })),
    storeAmountCents: storeAmount,
    policy: 'SUPPLIER_FIXED_PRICE'
  };
}
```

Service (com regras fortes)

```
// src/sales/sales-create.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';
import { computeSplitFixedSupplier } from './utils/computeSplit';

@Injectable()
export class SalesCreateService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async create(actor: any, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'sales.create');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
      if (idem.exists) return idem.data()!.result;

      if (!dto.items?.length) throw new Error('EMPTY_SALE');

      // 1) validar lotAllocations: soma bate com qtyBase e lot existe
e supplier confere e tem unitPay
      const enrichedItems: any[] = [];

      for (const it of dto.items) {
        const qtyBase = Number(it.qtyBase || 0);
        if (!(qtyBase > 0)) throw new Error('INVALID_ITEM_QTY');

        const allocs = it.lotAllocations || [];
        if (!allocs.length) throw new
Error('MISSING_LOT_ALLOCATIONS');

        const sumAlloc = allocs.reduce((a: number, x: any) => a +
Number(x.qtyBase || 0), 0);
        if (Math.round(sumAlloc * 1e6) !== Math.round(qtyBase * 1e6))
throw new Error('ALLOC_QTY_MISMATCH');

        const enrichedAllocs: any[] = [];
        for (const al of allocs) {
          const lotRef = this.db.collection('lots').doc(al.lotId);
          const lotSnap = await tx.get(lotRef);
          if (!lotSnap.exists) throw new Error('LOT_NOT_FOUND');

          const lot = lotSnap.data()!;
```

```
          if (lot.tenantId !== tenantId) throw new
Error('FORBIDDEN_LOT');
          if (lot.itemId !== it.itemId) throw new
Error('LOT_ITEM_MISMATCH');
          if (lot.supplierTenantId !== al.supplierTenantId) throw new
Error('LOT_SUPPLIER_MISMATCH');

          const unitPay = Number(lot.supplierTerms?.unitPayCentsBase
|| 0);
          if (!(unitPay > 0)) throw new
Error('SUPPLIER_PAY_TERM_MISSING');

          enrichedAllocs.push({
            lotId: lot.lotId,
            supplierTenantId: lot.supplierTenantId,
            qtyBase: Number(al.qtyBase),
            unitPayCentsBase: unitPay
          });
        }

        enrichedItems.push({
          itemId: it.itemId,
          qtyBase,
          unitBase: String(it.unitBase || 'UN').toUpperCase(),
          unitPriceCentsBase: Number(it.unitPriceCentsBase || 0),
          pricingModel: 'FIXED_SUPPLIER_PRICE',
          lotAllocations: enrichedAllocs
        });
      }

      // 2) compute split (fiel ao fornecedor)
      const split = computeSplitFixedSupplier(enrichedItems);

      const saleId = makeId('sale');
      const saleRef = this.db.collection('sales').doc(saleId);

      tx.create(saleRef, {
        tenantId,
        saleId,
        propertyId: dto.propertyId,
        items: enrichedItems,
        amountCents: split.totalSaleCents,
        splitPlan: { targets: split.targets, storeAmountCents:
split.storeAmountCents, policy: split.policy },
        status: 'PENDING_PAYMENT',
        createdAt: now,
        createdBy: { uid: actor.uid, role: actor.role }
      });

      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        type: 'SALE_CREATED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { saleId, amountCents: split.totalSaleCents, suppliers:
split.targets.length },
        ts: now
      });
```

```
      const out = { saleId, status: 'PENDING_PAYMENT', amountCents:
split.totalSaleCents, splitPlan: split };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'sales.create', clientRequestId: dto.clientRequestId, createdAt: now,
result: out });
      return out;
    });
  }
}
```

Controller

```
// src/sales/sales.controller.ts
import { Body, Controller, Param, Post, UseGuards } from
'@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { SalesCreateService } from './sales-create.service';

@Controller('/api/sales')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class SalesController {
  constructor(private readonly createSvc: SalesCreateService) {}

  @Post()
  @Roles('ADMIN','MANAGER','OPERATOR')
  @Permissions('sales.create')
  create(@Actor() actor: any, @Body() dto: any) {
    return this.createSvc.create(actor, dto);
  }
}
```

---

3) POST /api/sales/:id/pay (reserva + cria escrow)

Aqui a regra é: **só reserva estoque após início de pagamento** (e você pediu: reserva só após pagamento escrow — dá para escolher).
Como "reserva só após pagamento" é duro (pode vender e alguém levar antes), o padrão ideal é:

- `PENDING_PAYMENT` → cria cobrança e marca `PAYMENT_STARTED`
- **reserva imediatamente** (para segurar o lote), mas se não pagar em X min, cancela e estorna reserva com multa (se aplicável).

Se você quer literal "reserva somente após pagamento", troque a reserva para o webhook PAID. Eu recomendo reservar no pay.

DTO

```
// src/sales/dto/sale-pay.dto.ts
import { IsNotEmpty, IsString } from 'class-validator';

export class SalePayDto {
```

```
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsString() @IsNotEmpty()
  payMethod: 'PIX'|'BOLETO'|'CREDIT_CARD';
}
```

Service (reserva em transação)

```
// src/sales/sales-pay.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class SalesPayService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async pay(actor: any, saleId: string, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'sales.pay');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
      if (idem.exists) return idem.data()!.result;

      const saleRef = this.db.collection('sales').doc(saleId);
      const saleSnap = await tx.get(saleRef);
      if (!saleSnap.exists) throw new Error('SALE_NOT_FOUND');
      const sale = saleSnap.data()!;
      if (sale.tenantId !== tenantId) throw new Error('FORBIDDEN');
      if (sale.status !== 'PENDING_PAYMENT') throw new
Error('SALE_NOT_PAYABLE');

      // 1) reservar lotes
      for (const it of sale.items) {
        for (const al of it.lotAllocations) {
          const lotRef = this.db.collection('lots').doc(al.lotId);
          const lotSnap = await tx.get(lotRef);
          if (!lotSnap.exists) throw new Error('LOT_NOT_FOUND');
          const lot = lotSnap.data()!;
          if (lot.tenantId !== tenantId) throw new
Error('FORBIDDEN_LOT');

          const avail = Number(lot.stock?.qtyBaseAvailable || 0) -
Number(lot.stock?.reservedBase || 0);
          if (avail < Number(al.qtyBase)) throw new
Error('LOT_INSUFFICIENT_STOCK');

          tx.set(lotRef, {
            stock: {
              ...lot.stock,
              reservedBase: Number(lot.stock?.reservedBase || 0) +
Number(al.qtyBase)
            }
          }, { merge: true });
        }
```

```
      }

      // 2) criar escrow doc (integração PSP fica aqui; por enquanto
stub)
      const escrowId = makeId('esc_sale');
      const escrowRef =
this.db.collection('escrows_sales').doc(escrowId);

      tx.create(escrowRef, {
        tenantId,
        escrowId,
        saleId,
        status: 'REQUESTED',
        amountCents: sale.amountCents,
        splitPlan: sale.splitPlan,           // "espelho" do split
        payMethod: dto.payMethod,
        createdAt: now
      });

      tx.set(saleRef, {
        status: 'PAYMENT_STARTED',
        escrow: { escrowId, status: 'REQUESTED' },
        paymentStartedAt: now
      }, { merge: true });

      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        type: 'SALE_PAYMENT_STARTED_AND_LOTS_RESERVED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { saleId, escrowId, amountCents: sale.amountCents },
        ts: now
      });

      const out = { saleId, escrowId, status: 'PAYMENT_STARTED' };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'sales.pay', clientRequestId: dto.clientRequestId, createdAt: now,
result: out });
      return out;
    });
  }
}
```

Controller

```
// src/sales/sales-pay.controller.ts
import { Body, Controller, Param, Post, UseGuards } from
'@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { SalesPayService } from './sales-pay.service';

@Controller('/api/sales')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class SalesPayController {
  constructor(private readonly paySvc: SalesPayService) {}
```

```
  @Post('/:id/pay')
  @Roles('ADMIN','MANAGER','OPERATOR')
  @Permissions('sales.pay')
  pay(@Actor() actor: any, @Param('id') id: string, @Body() dto: any)
{
    return this.paySvc.pay(actor, id, dto);
  }
}
```

---

4) Webhook PAID → baixa estoque + split/payout por fornecedor

Aqui é onde fica **fiel ao fornecedor** de vez.

Objetivo

- Verifica escrow
- Marca escrow PAID
- Marca sale PAID
- **Baixa estoque**: reduz `qtyBaseAvailable` e reduz `reservedBase`
- Executa **payout por fornecedor** (stub), registrando `supplier_payouts`
- Auditoria

Service

```
// src/webhooks/asaas-sales-paid.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class AsaasSalesPaidService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async handlePaid(escrowId: string, paidAt: number, pspRef?: any) {
    const escrowRef =
this.db.collection('escrows_sales').doc(escrowId);

    return this.db.runTransaction(async (tx) => {
      const escSnap = await tx.get(escrowRef);
      if (!escSnap.exists) throw new Error('ESCROW_NOT_FOUND');
      const esc = escSnap.data()!;
      if (esc.status === 'PAID') return { ok: true, already: true };

      const saleRef = this.db.collection('sales').doc(esc.saleId);
      const saleSnap = await tx.get(saleRef);
      if (!saleSnap.exists) throw new Error('SALE_NOT_FOUND');
      const sale = saleSnap.data()!;

      // trava: split do escrow deve bater com split do sale
      // (protege contra alteração maliciosa)
      const spSale = JSON.stringify(sale.splitPlan || {});
      const spEsc = JSON.stringify(esc.splitPlan || {});
      if (spSale !== spEsc) throw new
Error('SPLIT_MISMATCH_ESCROW_VS_SALE');
```

```
      // 1) marcar escrow PAID
      tx.set(escrowRef, { status: 'PAID', paidAt, pspRef: pspRef ??
null }, { merge: true });

      // 2) baixar estoque por lote (fidelidade)
      for (const it of sale.items) {
        for (const al of it.lotAllocations) {
          const lotRef = this.db.collection('lots').doc(al.lotId);
          const lotSnap = await tx.get(lotRef);
          if (!lotSnap.exists) throw new Error('LOT_NOT_FOUND');
          const lot = lotSnap.data()!;

          const avail = Number(lot.stock?.qtyBaseAvailable || 0);
          const reserved = Number(lot.stock?.reservedBase || 0);
          const q = Number(al.qtyBase);

          if (reserved < q) throw new Error('RESERVE_INCONSISTENT');
          if (avail < q) throw new Error('AVAILABLE_INCONSISTENT');

          tx.set(lotRef, {
            stock: {
              ...lot.stock,
              qtyBaseAvailable: avail - q,
              reservedBase: reserved - q
            }
          }, { merge: true });

          // opcional: criar stock movement append-only por lote
          const mvId = makeId('mv');
          tx.create(this.db.collection('stock_moves').doc(mvId), {
            tenantId: lot.tenantId,
            moveId: mvId,
            type: 'SALE_OUT',
            saleId: sale.saleId,
            lotId: lot.lotId,
            itemId: lot.itemId,
            qtyBase: q,
            unitBase: lot.pricing?.unitBase || it.unitBase,
            ts: paidAt
          });
        }
      }

      // 3) criar payouts por fornecedor (stub - aqui integra PSP)
      const targets = sale.splitPlan?.targets || [];
      for (const t of targets) {
        const payoutId = makeId('payout');

tx.create(this.db.collection('supplier_payouts').doc(payoutId), {
          tenantId: sale.tenantId,
          payoutId,
          saleId: sale.saleId,
          supplierTenantId: t.supplierTenantId,
          amountCents: t.amountCents,
          status: 'QUEUED', // depois: SENT/CONFIRMED
          createdAt: paidAt
        });
      }

      // 4) marcar sale PAID + imutabilidade lógica
      tx.set(saleRef, {
```

```
        status: 'PAID',
        paidAt,
        escrow: { escrowId, status: 'PAID' }
      }, { merge: true });

      // 5) auditoria
      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId: sale.tenantId,
        type: 'SALE_PAID_STOCK_DEBITED_SPLIT_QUEUED',
        meta: { saleId: sale.saleId, escrowId, suppliers:
targets.length, amountCents: sale.amountCents },
        ts: paidAt
      });

      return { ok: true, saleId: sale.saleId, escrowId, payouts:
targets.length };
    });
  }
}
```

Controller do webhook (stub)

```
// src/webhooks/asaas.controller.ts
import { Body, Controller, Post } from '@nestjs/common';
import { AsaasSalesPaidService } from './asaas-sales-paid.service';

@Controller('/api/webhooks/asaas')
export class AsaasWebhookController {
  constructor(private readonly paidSvc: AsaasSalesPaidService) {}

  @Post()
  async handle(@Body() body: any) {
    // aqui você detecta evento "PAYMENT_CONFIRMED" do Asaas
    // e extrai escrowId (você guarda no metadata da cobrança)
    const escrowId = body?.payment?.metadata?.escrowId;
    const status = body?.payment?.status;
    if (status === 'CONFIRMED' || status === 'RECEIVED') {
      return this.paidSvc.handlePaid(escrowId, Date.now(), body);
    }
    return { ok: true };
  }
}
```

---

5) Travas anti-fraude que você pediu (resumo)

- **Split sempre do lote** (fiel ao fornecedor)
- No webhook PAID: `split(escrow) == split(sale)` obrigatório
- Baixa por lote (não por item genérico)
- Auditoria de tudo
- Idempotência em todos endpoints

---

6) Permissions que entram aqui

- `lots.allocate`
- `sales.create`
- `sales.pay`
- `webhooks.asaas.receive` (interno, normalmente sem auth pública)
- (opcional) `supplier.payouts.read`

Se você quiser o próximo passo, eu fecho também:

- **cancelamento com multa** (se não pagar em X minutos: estorna reserva e aplica multa)
- e **modo "reserva somente após pagamento"** (para cumprir literalmente sua regra, com risco de falta de estoque — você escolhe).

Fechado — abaixo estão **(A)** cancelamento com multa + estorno de reserva (timeout) e **(B)** a opção "reserva só após pagamento" (literal), para você escolher no código (deixo as duas prontas). Também incluo o **RulesEngine** aplicado nesses serviços.

A) Timeout de pagamento + Cancelamento com multa + estorno de reserva

1) Regras de negócio (claras e auditáveis)

1. Venda criada → `PENDING_PAYMENT`
2. Iniciou pagamento → `PAYMENT_STARTED` (opcional)
3. Se não pagar em `TTL_MIN`:
    - muda para `CANCELLED_TIMEOUT`
    - estorna reserva (reservedBase -= qty)
    - aplica multa (se houver) como `penaltyCents`
    - registra auditoria "por quê, quando, quem"

A multa pode ser:

- percentual do total (ex.: 2%)
- valor fixo
- ou "zero" para MVP

2) Campos novos em sale

`sales/{saleId}` adiciona:

```
{
  "payment": {
    "ttlMinutes": 15,
    "expiresAt": 1730000000000,
```

```
    "penaltyPolicy": { "type":"PERCENT", "value":2 },
    "penaltyCents": 0
  }
}
```

---

3) Service: cancelar por timeout (robusto)

Função: `cancelExpiredSales()` (rodar por cron/worker)

- Busca sales `PAYMENT_STARTED` com `expiresAt < now`
- Para cada sale: chama `cancelSaleTimeout(saleId)`

Como você não quer tarefas async aqui, deixo o serviço pronto; você pluga num scheduler depois (Cloud Scheduler / cron job / queue).

Service: cancelSaleTimeout (transação)

```
// src/sales/sales-timeout-cancel.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';

function computePenalty(amountCents: number, policy: any) {
  if (!policy) return 0;
  if (policy.type === 'PERCENT') return Math.round(amountCents *
(Number(policy.value || 0) / 100));
  if (policy.type === 'FIXED') return Math.round(Number(policy.value
|| 0));
  return 0;
}

@Injectable()
export class SalesTimeoutCancelService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async cancelTimeout(tenantId: string, saleId: string, reason =
'TIMEOUT') {
    const now = Date.now();

    return this.db.runTransaction(async (tx) => {
      const saleRef = this.db.collection('sales').doc(saleId);
      const saleSnap = await tx.get(saleRef);
      if (!saleSnap.exists) throw new Error('SALE_NOT_FOUND');
      const sale = saleSnap.data()!;
      if (sale.tenantId !== tenantId) throw new Error('FORBIDDEN');

      if
(!['PAYMENT_STARTED','PENDING_PAYMENT'].includes(sale.status)) {
        return { ok: true, skipped: true, status: sale.status };
      }

      // já pago? não cancela
      const escId = sale.escrow?.escrowId;
      if (escId) {
        const escSnap = await
tx.get(this.db.collection('escrows_sales').doc(escId));
```

```
        if (escSnap.exists && escSnap.data()!.status === 'PAID') {
          return { ok: true, skipped: true, status: 'PAID_ALREADY' };
        }
      }

      // estornar reserva (se tiver reservado)
      for (const it of sale.items || []) {
        for (const al of it.lotAllocations || []) {
          const lotRef = this.db.collection('lots').doc(al.lotId);
          const lotSnap = await tx.get(lotRef);
          if (!lotSnap.exists) throw new Error('LOT_NOT_FOUND');
          const lot = lotSnap.data()!;
          if (lot.tenantId !== tenantId) throw new
Error('FORBIDDEN_LOT');

          const reserved = Number(lot.stock?.reservedBase || 0);
          const q = Number(al.qtyBase || 0);
          if (reserved < q) {
            // inconsistência: registra e zera o que der para não
travar o sistema
            tx.set(lotRef, { stock: { ...lot.stock, reservedBase:
Math.max(0, reserved - q) } }, { merge: true });
          } else {
            tx.set(lotRef, { stock: { ...lot.stock, reservedBase:
reserved - q } }, { merge: true });
          }
        }
      }

      const penaltyPolicy = sale.payment?.penaltyPolicy || null;
      const penaltyCents = computePenalty(Number(sale.amountCents ||
0), penaltyPolicy);

      tx.set(saleRef, {
        status: 'CANCELLED_TIMEOUT',
        cancelledAt: now,
        cancelReason: reason,
        payment: {
          ...sale.payment,
          penaltyCents
        }
      }, { merge: true });

      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        type: 'SALE_CANCELLED_TIMEOUT_RESERVATION_REVERSED',
        meta: { saleId, penaltyCents, reason },
        ts: now
      });

      return { ok: true, saleId, status: 'CANCELLED_TIMEOUT',
penaltyCents };
    });
  }
}
```

4) Ajuste no `sales/:id/pay`: setar TTL e expiresAt

Dentro da transação do pay (onde você cria escrow), adicione:

```
const ttlMinutes = 15;
const expiresAt = now + ttlMinutes * 60_000;

tx.set(saleRef, {
  status: 'PAYMENT_STARTED',
  escrow: { escrowId, status: 'REQUESTED' },
  payment: {
    ttlMinutes,
    expiresAt,
    penaltyPolicy: { type: 'PERCENT', value: 2 } // exemplo
  }
}, { merge: true });
```

B) Opção “reserva só após pagamento” (literal)

Se você quer cumprir literalmente “só reserva estoque após pagamento escrow”, então:

1) No `sales/:id/pay`:

- NÃO mexe em lots.reservedBase
- só cria escrow e marca `PAYMENT_STARTED`

2) No webhook PAID:

- primeiro valida disponibilidade do lote (ainda tem estoque?)
- se tiver:
  - baixa estoque
  - executa split
- se não tiver:
  - marca venda como `FAILED_OUT_OF_STOCK_AFTER_PAYMENT`
  - coloca dinheiro em “dispute/chargeback” (manual)
  - auditoria forte

Patch no webhook

Antes de baixar:

```
const avail = Number(lot.stock?.qtyBaseAvailable || 0);
if (avail < q) throw new Error('OUT_OF_STOCK_AFTER_PAYMENT');
```

✅ fiel ao seu requisito, mas **risco** de “vender e acabar” aumenta.
Minha recomendação prática: **reservar ao iniciar pagamento** e manter TTL curto.

C) Aplicar RulesEngine (como você pediu)

Vou te dar um RulesEngine bem simples (plugável) e como chamar nos serviços.

1) Interface

```
// src/rules/rules.types.ts
export type RuleContext = {
  tenantId: string;
  actor: any;
  action: string;
  payload: any;
  now: number;
};

export type RuleResult = { ok: true } | { ok: false; code: string;
message?: string; details?: any };

export interface Rule {
  name: string;
  run(ctx: RuleContext): Promise<RuleResult>;
}
```

2) Engine

```
// src/rules/rules-engine.service.ts
import { Injectable } from '@nestjs/common';
import { Rule, RuleContext } from './rules.types';

@Injectable()
export class RulesEngine {
  async enforce(rules: Rule[], ctx: RuleContext) {
    for (const r of rules) {
      const res = await r.run(ctx);
      if (!res.ok) {
        const err = new Error(res.code);
        (err as any).details = res.details;
        throw err;
      }
    }
  }
}
```

3) Regras essenciais do split fiel

```
// src/sales/rules/identity-required.rule.ts
import { Rule } from '../../rules/rules.types';

export class IdentityRequiredRule implements Rule {
  name = 'IdentityRequiredRule';
  async run(ctx: any) {
    const items = ctx.payload.items || [];
    for (const it of items) {
      if (!it.lotAllocations || !it.lotAllocations.length) return {
ok:false, code:'MISSING_LOT_ALLOCATIONS' };
      const sum =
it.lotAllocations.reduce((a:number,x:any)=>a+Number(x.qtyBase||0),0);
      if (Math.round(sum*1e6) !==
Math.round(Number(it.qtyBase||0)*1e6)) return { ok:false,
code:'ALLOC_QTY_MISMATCH' };
    }
    return { ok:true };
  }
```

```
}
```

Você chama assim no `SalesCreateService` antes de gravar:

```
await this.rules.enforce([new IdentityRequiredRule()], {
  tenantId,
  actor,
  action: 'sales.create',
  payload: dto,
  now
});
```

D) Permissions novas

- `sales.cancelTimeout` (interno/manager)
- `sales.read` (para conferir)
- `sales.refund` (futuro)

E) O que fica faltando para "produção real" (curto e objetivo)

1. Worker/Cron para timeout (Cloud Scheduler → endpoint protegido)
2. Integração real Asaas (criar cobrança com metadata `escrowId`)
3. Payout real por fornecedor (subconta/transferência)
4. Tela PDV: "Pagamento pendente (expira em X min)"

Se você quiser, eu mando agora:

- o endpoint **protegido** `/api/internal/sales/cancel-expired` (somente ADMIN + secret)
- e o "refund/dispute flow" para o caso raro de "pagou mas ficou sem estoque" (no modo literal).

Perfeito — abaixo vai:

1. **Endpoint interno protegido** `/api/internal/sales/cancel-expired` (Cloud Scheduler / cron)
2. **Fluxo completo de refund/disputa** (caso "pagou e ficou sem estoque" ou disputa de entrega)
3. Regras + estados + coleções Firestore para ficar auditável e "à prova de jeitinho"

---

1) Endpoint interno: cancelar vendas expiradas

1.1 Segurança: Secret + allowlist simples

- Header: `x-internal-secret: <SEGREDO>`
- (Opcional) checar `x-forwarded-for` ou `cf-connecting-ip` (não recomendo depender só disso)
- Não expõe para app/mobile.

Config (env)

- `INTERNAL_WEBHOOK_SECRET=...`

1.2 Controller

```
// src/internal/internal-sales.controller.ts
import { Controller, Post, Headers, UnauthorizedException, UseGuards }
from '@nestjs/common';
import { SalesCancelExpiredService } from './sales-cancel-
expired.service';

@Controller('/api/internal/sales')
export class InternalSalesController {
  constructor(private readonly svc: SalesCancelExpiredService) {}

  @Post('/cancel-expired')
  async cancelExpired(@Headers('x-internal-secret') secret: string) {
    if (!secret || secret !== process.env.INTERNAL_WEBHOOK_SECRET) {
      throw new UnauthorizedException('INVALID_INTERNAL_SECRET');
    }
    return this.svc.run();
  }
}
```

1.3 Service (varre e cancela em lote)

- Busca vendas com `status=PAYMENT_STARTED` e `payment.expiresAt < now`
- Cancela uma por uma usando o `SalesTimeoutCancelService.cancelTimeout()`

```
// src/internal/sales-cancel-expired.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { SalesTimeoutCancelService } from '../sales/sales-timeout-
cancel.service';

@Injectable()
export class SalesCancelExpiredService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly cancelSvc: SalesTimeoutCancelService
  ) {}

  async run() {
    const now = Date.now();
```

```
    // Firestore: precisa index (tenantId + status +
payment.expiresAt) ou apenas status+expiresAt
    // Para multi-tenant, você pode rodar por tenant. Aqui: varredura
global (depende do seu modelo).
    const snap = await this.db.collection('sales')
      .where('status', '==', 'PAYMENT_STARTED')
      .where('payment.expiresAt', '<', now)
      .limit(200)
      .get();

    const results: any[] = [];
    for (const doc of snap.docs) {
      const sale = doc.data()!;
      try {
        const r = await this.cancelSvc.cancelTimeout(sale.tenantId,
sale.saleId, 'TIMEOUT');
        results.push({ saleId: sale.saleId, ok: true, result: r });
      } catch (e: any) {
        results.push({ saleId: sale.saleId, ok: false, error:
String(e?.message || e) });
      }
    }

    return { ok: true, checked: snap.size, results };
  }
}
```

Como agendar

- Cloud Scheduler → chama `POST /api/internal/sales/cancel-expired`
- Frequência: a cada 1–5 minutos
- Header `x-internal-secret`

---

2) Fluxo refund/disputa (pagou mas sem estoque / disputa de entrega)

Você pediu fidelidade total e segurança jurídica. Então o fluxo precisa de:

- **estado de exceção** na venda e no escrow
- **freeze** (bloqueio) de qualquer movimentação relacionada
- **registro formal** do motivo + evidências
- **ação de auditor** (AUDITOR) para decidir: refund / reentrega / substituição / liberação parcial

2.1 Estados recomendados

Sale.status

- `PENDING_PAYMENT`
- `PAYMENT_STARTED`
- `PAID`

- FAILED_OUT_OF_STOCK_AFTER_PAYMENT ✅
- DISPUTE_OPEN ✅
- REFUND_REQUESTED ✅
- REFUNDED ✅
- CANCELLED_TIMEOUT

Escrow.status

- REQUESTED
- PAID
- HOLD_FOR_DISPUTE ✅
- REFUND_REQUESTED ✅
- REFUNDED ✅

---

3) Coleções para disputa e refund

3.1 disputes/{disputeId}

```
{
  "tenantId":"TENANT_STORE",
  "disputeId":"disp_001",
  "saleId":"sale_0001",
  "escrowId":"esc_sale_0001",
  "type":"OUT_OF_STOCK|DELIVERY_DISPUTE|QUALITY",
  "reason":"Produto indisponível após pagamento",
  "evidenceIds":["evg_photo", "evg_gps"],
  "status":"OPEN|UNDER_REVIEW|RESOLVED",
  "resolution": {

"type":"REFUND_FULL|REFUND_PARTIAL|REDELIVER|SUBSTITUTE|RELEASE_TO_SUP
PLIER",
    "amountCents": 0,
    "notes":"..."
  },
  "createdAt":1730000000000,
  "createdBy":{"uid":"...","role":"OPERATOR"},
  "reviewedBy":null
}
```

3.2 refunds/{refundId}

```
{
  "tenantId":"TENANT_STORE",
  "refundId":"ref_001",
  "saleId":"sale_0001",
  "escrowId":"esc_sale_0001",
  "amountCents": 10000,
  "status":"REQUESTED|SENT|CONFIRMED|FAILED",
  "pspRef": null,
  "createdAt": 1730000000000,
  "createdBy": {"uid":"...","role":"AUDITOR"}
}
```

Observação: se o dinheiro ainda não foi “payout” para fornecedor, refund é direto.
Se já houve payout, o fluxo vira **chargeback interno** com “recolhimento” — mas no

seu design ideal, pagamento só libera quando máquina de estados permite (entrega validada), então normalmente o payout ainda não saiu.

---

4) Endpoint: marcar falha “pagou sem estoque” automaticamente

Isso acontece no modo literal “reserva só após pagamento”.
No webhook PAID, se faltar estoque, você **não debita lotes**, abre disputa e coloca escrow em hold.

4.1 Patch no webhook PAID (trecho)

```
// dentro do loop, antes de debitar:
if (avail < q) {
  // 1) marca escrow hold
  tx.set(escrowRef, { status: 'HOLD_FOR_DISPUTE', holdReason:
'OUT_OF_STOCK_AFTER_PAYMENT' }, { merge: true });

  // 2) marca sale falha
  tx.set(saleRef, { status: 'FAILED_OUT_OF_STOCK_AFTER_PAYMENT' }, {
merge: true });

  // 3) cria disputa automaticamente
  const disputeId = makeId('disp');
  tx.create(this.db.collection('disputes').doc(disputeId), {
    tenantId: sale.tenantId,
    disputeId,
    saleId: sale.saleId,
    escrowId,
    type: 'OUT_OF_STOCK',
    reason: 'Falta de estoque detectada após pagamento',
    evidenceIds: [],
    status: 'OPEN',
    createdAt: paidAt,
    createdBy: { uid: 'system', role: 'SYSTEM' }
  });

  // 4) auditoria
  tx.create(this.db.collection('audit_events').doc(makeId('aud')), {
    auditId: makeId('aud'),
    tenantId: sale.tenantId,
    type: 'PAID_BUT_OUT_OF_STOCK_DISPUTE_OPENED',
    meta: { saleId: sale.saleId, escrowId },
    ts: paidAt
  });

  // encerra com erro controlado
  throw new Error('OUT_OF_STOCK_AFTER_PAYMENT');
}
```

---

5) Endpoint: abrir disputa manual (ex.: entrega/qualidade)

Controller

```
// src/disputes/disputes.controller.ts
import { Body, Controller, Post, UseGuards } from '@nestjs/common';
```

```
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { DisputesService } from './disputes.service';

@Controller('/api/disputes')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class DisputesController {
  constructor(private readonly svc: DisputesService) {}

  @Post()

@Roles('ADMIN','MANAGER','AUDITOR','OPERATOR','PRODUCER','TECHNICIAN')
  @Permissions('disputes.create')
  create(@Actor() actor: any, @Body() dto: any) {
    return this.svc.create(actor, dto);
  }
}
```

Service (criar disputa e travar escrow)

```
// src/disputes/disputes.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class DisputesService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async create(actor: any, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const saleRef = this.db.collection('sales').doc(dto.saleId);
    const saleSnap = await saleRef.get();
    if (!saleSnap.exists) throw new Error('SALE_NOT_FOUND');
    const sale = saleSnap.data()!;
    if (sale.tenantId !== tenantId) throw new Error('FORBIDDEN');

    const escrowId = sale.escrow?.escrowId;
    if (!escrowId) throw new Error('ESCROW_NOT_FOUND');

    return this.db.runTransaction(async (tx) => {
      // trava escrow
      const escrowRef =
this.db.collection('escrows_sales').doc(escrowId);
      const escSnap = await tx.get(escrowRef);
      if (!escSnap.exists) throw new Error('ESCROW_NOT_FOUND');
      const esc = escSnap.data()!;
      if (esc.status === 'REFUNDED') throw new
Error('ALREADY_REFUNDED');

      tx.set(escrowRef, { status: 'HOLD_FOR_DISPUTE', holdReason:
dto.type || 'DISPUTE' }, { merge: true });
      tx.set(saleRef, { status: 'DISPUTE_OPEN' }, { merge: true });
```

```
      const disputeId = makeId('disp');
      tx.create(this.db.collection('disputes').doc(disputeId), {
        tenantId,
        disputeId,
        saleId: dto.saleId,
        escrowId,
        type: dto.type || 'DELIVERY_DISPUTE',
        reason: dto.reason || '',
        evidenceIds: dto.evidenceIds || [],
        status: 'OPEN',
        createdAt: now,
        createdBy: { uid: actor.uid, role: actor.role }
      });

      tx.create(this.db.collection('audit_events').doc(makeId('aud')),
{
        auditId: makeId('aud'),
        tenantId,
        type: 'DISPUTE_CREATED_ESCROW_HELD',
        actor: { uid: actor.uid, role: actor.role },
        meta: { disputeId, saleId: dto.saleId, escrowId },
        ts: now
      });

      return { disputeId, status: 'OPEN' };
    });
  }
}
```

---

6) Endpoint: resolver disputa (AUDITOR decide refund/liberação)

Controller

```
// src/disputes/disputes-resolve.controller.ts
import { Body, Controller, Param, Post, UseGuards } from
'@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { DisputesResolveService } from './disputes-resolve.service';

@Controller('/api/disputes')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class DisputesResolveController {
  constructor(private readonly svc: DisputesResolveService) {}

  @Post('/:id/resolve')
  @Roles('AUDITOR','ADMIN')
  @Permissions('disputes.resolve')
  resolve(@Actor() actor: any, @Param('id') id: string, @Body() dto:
any) {
    return this.svc.resolve(actor, id, dto);
  }
}
```

Service (resolve → refund ou liberar)

```
// src/disputes/disputes-resolve.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class DisputesResolveService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async resolve(actor: any, disputeId: string, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    return this.db.runTransaction(async (tx) => {
      const dispRef = this.db.collection('disputes').doc(disputeId);
      const dispSnap = await tx.get(dispRef);
      if (!dispSnap.exists) throw new Error('DISPUTE_NOT_FOUND');
      const disp = dispSnap.data()!;
      if (disp.tenantId !== tenantId) throw new Error('FORBIDDEN');
      if (disp.status !== 'OPEN') throw new Error('DISPUTE_NOT_OPEN');

      const saleRef = this.db.collection('sales').doc(disp.saleId);
      const saleSnap = await tx.get(saleRef);
      if (!saleSnap.exists) throw new Error('SALE_NOT_FOUND');
      const sale = saleSnap.data()!;

      const escrowRef =
this.db.collection('escrows_sales').doc(disp.escrowId);
      const escSnap = await tx.get(escrowRef);
      if (!escSnap.exists) throw new Error('ESCROW_NOT_FOUND');
      const esc = escSnap.data()!;

      const resolutionType = dto.type; // REFUND_FULL | REFUND_PARTIAL
| RELEASE_TO_SUPPLIER | REDELIVER | SUBSTITUTE
      const amountCents = Number(dto.amountCents || 0);

      // Atualiza disputa
      tx.set(dispRef, {
        status: 'RESOLVED',
        resolvedAt: now,
        reviewedBy: { uid: actor.uid, role: actor.role },
        resolution: { type: resolutionType, amountCents, notes:
dto.notes || '' }
      }, { merge: true });

      // Aplica efeito
      if (resolutionType === 'REFUND_FULL') {
        tx.set(escrowRef, { status: 'REFUND_REQUESTED' }, { merge:
true });
        tx.set(saleRef, { status: 'REFUND_REQUESTED' }, { merge: true
});

        const refundId = makeId('ref');
        tx.create(this.db.collection('refunds').doc(refundId), {
          tenantId,
          refundId,
          saleId: sale.saleId,
          escrowId: disp.escrowId,
          amountCents: Number(sale.amountCents || 0),
```

```
          status: 'REQUESTED',
          createdAt: now,
          createdBy: { uid: actor.uid, role: actor.role }
        });
      }

      if (resolutionType === 'REFUND_PARTIAL') {
        if (!(amountCents > 0) || amountCents >=
Number(sale.amountCents || 0)) throw new
Error('INVALID_PARTIAL_AMOUNT');
        tx.set(escrowRef, { status: 'REFUND_REQUESTED' }, { merge:
true });
        tx.set(saleRef, { status: 'REFUND_REQUESTED' }, { merge: true
});

        const refundId = makeId('ref');
        tx.create(this.db.collection('refunds').doc(refundId), {
          tenantId,
          refundId,
          saleId: sale.saleId,
          escrowId: disp.escrowId,
          amountCents,
          status: 'REQUESTED',
          createdAt: now,
          createdBy: { uid: actor.uid, role: actor.role }
        });
      }

      if (resolutionType === 'RELEASE_TO_SUPPLIER') {
        // volta escrow para fluxo normal (você executa payout)
        tx.set(escrowRef, { status: 'PAID' }, { merge: true });
        tx.set(saleRef, { status: 'PAID' }, { merge: true });
      }

      if (resolutionType === 'REDELIVER' || resolutionType ===
'SUBSTITUTE') {
        // mantém hold e abre "tarefa operacional" (opcional)
        tx.set(escrowRef, { status: 'HOLD_FOR_DISPUTE' }, { merge:
true });
        tx.set(saleRef, { status: 'DISPUTE_OPEN' }, { merge: true });
      }

      tx.create(this.db.collection('audit_events').doc(makeId('aud')),
{
        auditId: makeId('aud'),
        tenantId,
        type: 'DISPUTE_RESOLVED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { disputeId, saleId: sale.saleId, escrowId:
disp.escrowId, resolutionType, amountCents },
        ts: now
      });

      return { ok: true, disputeId, resolutionType };
    });
  }
}
```

7) Executar refund no PSP (stub + idempotência)

Você terá um worker/handler que pega `refunds` com status `REQUESTED` e chama o PSP.

Service stub (marca CONFIRMED)

```
// src/refunds/refunds-processor.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class RefundsProcessorService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async markRefundConfirmed(refundId: string, pspRef: any) {
    const now = Date.now();
    const ref = this.db.collection('refunds').doc(refundId);
    const snap = await ref.get();
    if (!snap.exists) throw new Error('REFUND_NOT_FOUND');
    const r = snap.data()!;

    await this.db.runTransaction(async (tx) => {
      tx.set(ref, { status: 'CONFIRMED', confirmedAt: now, pspRef }, {
merge: true });

      tx.set(this.db.collection('escrows_sales').doc(r.escrowId), {
status: 'REFUNDED', refundedAt: now }, { merge: true });
      tx.set(this.db.collection('sales').doc(r.saleId), { status:
'REFUNDED', refundedAt: now }, { merge: true });

      tx.create(this.db.collection('audit_events').doc(makeId('aud')),
{
        auditId: makeId('aud'),
        tenantId: r.tenantId,
        type: 'REFUND_CONFIRMED',
        meta: { refundId, saleId: r.saleId, escrowId: r.escrowId,
amountCents: r.amountCents },
        ts: now
      });
    });

    return { ok: true, refundId, status: 'CONFIRMED' };
  }
}
```

---

8) Permissions (mínimas)

- `internal.sales.cancelExpired` (não via JWT, via secret)
- `disputes.create`
- `disputes.resolve` (AUDITOR/ADMIN)
- `refunds.read` (AUDITOR)
- `refunds.process` (interno)

---

9) Pontos que deixam “blindado”

- Escrow fica `HOLD_FOR_DISPUTE` quando precisa
- Auditor decide, e tudo fica em **append-only audit_events**
- Refund só muda para `REFUNDED` após confirmação do PSP
- Venda e split ficam consistentes e travados

Se quiser, o próximo módulo que complementa perfeitamente seu agro é:

- **Bloqueio por CAR/Inscrição Estadual** (inadimplência/descumprimento) com “lock registry” e regras ambientais — isso fecha seu requisito de impedir comercialização/nota quando em descumprimento.

Fechado — segue o módulo **Bloqueio por CAR / Inscrição Estadual** (e também por CNPJ/CPF se quiser), com **lock registry**, **regras ambientais**, **efeito em comercialização/nota**, auditoria e endpoints.

A ideia é simples e muito forte juridicamente:

✅ **Um “Lock” é um registro técnico imutável (append-only) que impede ações específicas** (vender, emitir nota, transferir ativo, etc.) enquanto o motivo existir.
✅ O lock pode ser disparado por: **inadimplência/descumprimento contratual**, **excesso ambiental**, **ativos de terceiros em custódia**, **disputa**, etc.
✅ O lock pode ser amarrado a: `carId`, `inscricaoEstadual`, `propertyId`, `tenantId`, `counterpartyTenantId`.

---

1) Modelo de dados

1.1 Propriedade com chaves CAR/IE

```
properties/{propertyId}
```

```
{
  "tenantId":"TENANT_PRODUCER_1",
  "propertyId":"prop_123",
  "name":"Fazenda X",
  "registry": {
    "carId":"MT-1234567-ABCDEF1234567890ABCDEF12",
    "inscricaoEstadual":"123456789",
    "municipio":"...",
    "uf":"MT"
  },
  "geo": { "lat": -15.7, "lng": -56.1 },
  "active": true
}
```

1.2 Locks (registro de bloqueio)

Coleção: `locks/{lockId}` (append-only: **não apaga**, apenas "resolve")

```
{
  "tenantId":"TENANT_PRODUCER_1",
  "lockId":"lock_001",
  "scope": {
    "type":"PROPERTY|CAR|IE|TENANT",
    "propertyId":"prop_123",
    "carId":"MT-123...",
    "inscricaoEstadual":"123456789"
  },
  "reason": {

"code":"CONTRACT_BREACH|ENV_LIMIT|THIRD_PARTY_CUSTODY|NON_PAYMENT|AUDI
T_HOLD",
    "title":"Descumprimento contratual",
    "details":"Não cumpriu marco X do contrato Y"
  },
  "effects": {
    "blockSale": true,
    "blockInvoice": true,
    "blockTransferAssets": true,
    "blockNewContracts": true
  },
  "links": {
    "contractId":"ctr_001",
    "saleId":null,
    "escrowId":null,
    "disputeId":null
  },
  "status":"ACTIVE|RESOLVED",
  "createdAt":1730000000000,
  "createdBy":{"uid":"...","role":"AUDITOR"},
  "resolvedAt": null,
  "resolvedBy": null
}
```

1.3 Índice auxiliar para busca rápida (opcional, recomendado)

```
lock_index/{indexKey}

{
  "tenantId":"TENANT_PRODUCER_1",
  "indexKey":"CAR:MT-
123...|IE:123456789|PROP:prop_123|TENANT:TENANT_PRODUCER_1",
  "activeLocks":[ "lock_001", "lock_044" ],
  "updatedAt":1730000000000
}
```

Isso acelera muito o "check locks" sem varrer coleção.

---

2) Regras de negócio (o que bloqueia o quê)

2.1 Ações que podem ser bloqueadas (effects)

- `blockSale` → impede criar venda / marketplace / PDV
- `blockInvoice` → impede emitir NF-e / lançar nota / "finalizar faturamento"
- `blockTransferAssets` → impede transferência de ativos (bovinos, mercadoria, quotas)
- `blockNewContracts` → impede novos contratos (produção sob demanda etc.)

2.2 Triggers típicos (reason.code)

- `CONTRACT_BREACH` (marco não cumprido)
- `NON_PAYMENT` (inadimplência)
- `THIRD_PARTY_CUSTODY` (ativos de terceiros na propriedade sem quitação)
- `ENV_LIMIT` (limite ambiental / irregularidade / bloqueio por compliance)
- `AUDIT_HOLD` (auditoria em andamento)
- `DISPUTE_OPEN` (disputa aberta)

---

3) RulesEngine: LockGuard (obrigatório em serviços críticos)

3.1 LockCheckService (consulta locks ativos por scope)

```
// src/locks/lock-check.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';

@Injectable()
export class LockCheckService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async getActiveLocksForProperty(tenantId: string, propertyId:
string) {
    // busca property para extrair CAR/IE
    const propSnap = await
this.db.collection('properties').doc(propertyId).get();
    if (!propSnap.exists) throw new Error('PROPERTY_NOT_FOUND');
    const prop = propSnap.data()!;
    if (prop.tenantId !== tenantId) throw new
Error('FORBIDDEN_PROPERTY');

    const carId = prop.registry?.carId || null;
    const ie = prop.registry?.inscricaoEstadual || null;

    // consulta locks ativos por (propertyId OU carId OU ie OU tenant)
    const queries: Promise<FirebaseFirestore.QuerySnapshot>[] = [];

    queries.push(
      this.db.collection('locks')
        .where('tenantId','==',tenantId)
        .where('status','==','ACTIVE')
        .where('scope.type','==','PROPERTY')
        .where('scope.propertyId','==',propertyId)
        .limit(50)
```

```
        .get()
    );

    if (carId) queries.push(
      this.db.collection('locks')
        .where('tenantId','==',tenantId)
        .where('status','==','ACTIVE')
        .where('scope.type','==','CAR')
        .where('scope.carId','==',carId)
        .limit(50)
        .get()
    );

    if (ie) queries.push(
      this.db.collection('locks')
        .where('tenantId','==',tenantId)
        .where('status','==','ACTIVE')
        .where('scope.type','==','IE')
        .where('scope.inscricaoEstadual','==',ie)
        .limit(50)
        .get()
    );

    queries.push(
      this.db.collection('locks')
        .where('tenantId','==',tenantId)
        .where('status','==','ACTIVE')
        .where('scope.type','==','TENANT')
        .where('scope.tenantId','==',tenantId)
        .limit(50)
        .get()
    );

    const snaps = await Promise.all(queries);
    const locks = snaps.flatMap(s => s.docs.map(d => d.data()));
    return { locks, carId, inscricaoEstadual: ie };
  }

  enforceEffects(locks: any[], requiredNoBlocks: Array<keyof any>) {
    // requiredNoBlocks ex: ['blockSale','blockInvoice']
    for (const l of locks) {
      const ef = l.effects || {};
      for (const k of requiredNoBlocks) {
        if (ef[k] === true) {
          const err = new Error('LOCK_BLOCKED_' + String(k));
          (err as any).details = { lockId: l.lockId, reason: l.reason
};
          throw err;
        }
      }
    }
  }
}
```

3.2 Usar nos serviços críticos

Exemplo 1: criar venda (marketplace/PDV)

No `SalesCreateService.create()`:

```
const { locks } = await
this.lockCheck.getActiveLocksForProperty(tenantId, dto.propertyId);
this.lockCheck.enforceEffects(locks,
['blockSale','blockNewContracts']);
```

Exemplo 2: emitir nota / finalizar faturamento

No serviço de emissão de NF:

```
const { locks } = await
this.lockCheck.getActiveLocksForProperty(tenantId, propertyId);
this.lockCheck.enforceEffects(locks, ['blockInvoice']);
```

Exemplo 3: transferir ativos (bovinos/mercadorias)

```
this.lockCheck.enforceEffects(locks, ['blockTransferAssets']);
```

✅ Isso garante tecnicamente: “imóvel bloqueado por CAR/IE não comercializa”.

---

4) Endpoints Locks (criar/consultar/resolver)

4.1 POST /api/locks (criar lock)

Somente `AUDITOR/ADMIN` (ou MANAGER com permissão)

```
{
  "clientRequestId":"uuid",
  "scopeType":"PROPERTY|CAR|IE|TENANT",
  "propertyId":"prop_123",
  "carId":"MT-123...",
  "inscricaoEstadual":"123456789",
  "reason":{
    "code":"ENV_LIMIT",
    "title":"Limite ambiental excedido",
    "details":"Área embargada detectada"
  },
  "effects":{
    "blockSale":true,
    "blockInvoice":true,
    "blockTransferAssets":true,
    "blockNewContracts":true
  },
  "links":{ "contractId":"ctr_001" }
}
```

4.2 GET /api/locks?propertyId=prop_123 (listar locks ativos)

Para `AUDITOR/ADMIN/MANAGER` e também para o produtor (read-only) se você quiser transparência.

4.3 POST /api/locks/:id/resolve (resolver lock)

Somente `AUDITOR/ADMIN`, com evidência (ex.: laudo, documento, decisão)

```
{
  "clientRequestId":"uuid",
  "resolutionNotes":"Regularizado, anexo laudo.",
  "evidenceIds":["evg_laudo_pdf"]
}
```

---

5) Implementação NestJS (CRUD mínimo)

5.1 DTOs

```
// src/locks/dto/lock-create.dto.ts
import { IsIn, IsNotEmpty, IsObject, IsOptional, IsString } from
'class-validator';

export class LockCreateDto {
  @IsString() @IsNotEmpty() clientRequestId: string;

  @IsIn(['PROPERTY','CAR','IE','TENANT'])
  scopeType: 'PROPERTY'|'CAR'|'IE'|'TENANT';

  @IsOptional() @IsString() propertyId?: string;
  @IsOptional() @IsString() carId?: string;
  @IsOptional() @IsString() inscricaoEstadual?: string;

  @IsObject() reason: any;
  @IsObject() effects: any;

  @IsOptional() @IsObject() links?: any;
}
// src/locks/dto/lock-resolve.dto.ts
import { IsArray, IsNotEmpty, IsOptional, IsString } from 'class-
validator';

export class LockResolveDto {
  @IsString() @IsNotEmpty() clientRequestId: string;
  @IsOptional() @IsString() resolutionNotes?: string;
  @IsOptional() @IsArray() evidenceIds?: string[];
}
```

5.2 Service

```
// src/locks/locks.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class LocksService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async create(actor: any, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'locks.create');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
```

```
      if (idem.exists) return idem.data()!.result;

      const lockId = makeId('lock');
      const ref = this.db.collection('locks').doc(lockId);

      const scope: any = { type: dto.scopeType };
      if (dto.scopeType === 'PROPERTY') scope.propertyId =
dto.propertyId;
      if (dto.scopeType === 'CAR') scope.carId = dto.carId;
      if (dto.scopeType === 'IE') scope.inscricaoEstadual =
dto.inscricaoEstadual;
      if (dto.scopeType === 'TENANT') scope.tenantId = tenantId;

      tx.create(ref, {
        tenantId,
        lockId,
        scope,
        reason: dto.reason,
        effects: dto.effects,
        links: dto.links ?? {},
        status: 'ACTIVE',
        createdAt: now,
        createdBy: { uid: actor.uid, role: actor.role }
      });

      tx.create(this.db.collection('audit_events').doc(makeId('aud')),
{
        auditId: makeId('aud'),
        tenantId,
        type: 'LOCK_CREATED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { lockId, scope, reason: dto.reason, effects:
dto.effects },
        ts: now
      });

      const out = { lockId, status: 'ACTIVE' };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'locks.create', clientRequestId: dto.clientRequestId, createdAt: now,
result: out });
      return out;
    });
  }

  async listActive(actor: any, propertyId?: string) {
    const tenantId = actor.tenantId;

    if (!propertyId) {
      const snap = await this.db.collection('locks')
        .where('tenantId','==',tenantId)
        .where('status','==','ACTIVE')
        .limit(100)
        .get();
      return { items: snap.docs.map(d => d.data()) };
    }

    // se quiser, pode chamar o LockCheckService para incluir CAR/IE
automaticamente
    const snap = await this.db.collection('locks')
      .where('tenantId','==',tenantId)
      .where('status','==','ACTIVE')
```

```
      .where('scope.type','==','PROPERTY')
      .where('scope.propertyId','==',propertyId)
      .limit(100)
      .get();
    return { items: snap.docs.map(d => d.data()) };
  }

  async resolve(actor: any, lockId: string, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'locks.resolve');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
      if (idem.exists) return idem.data()!.result;

      const ref = this.db.collection('locks').doc(lockId);
      const snap = await tx.get(ref);
      if (!snap.exists) throw new Error('LOCK_NOT_FOUND');
      const lock = snap.data()!;
      if (lock.tenantId !== tenantId) throw new Error('FORBIDDEN');
      if (lock.status !== 'ACTIVE') return { lockId, status:
lock.status };

      tx.set(ref, {
        status: 'RESOLVED',
        resolvedAt: now,
        resolvedBy: { uid: actor.uid, role: actor.role },
        resolutionNotes: dto.resolutionNotes ?? '',
        resolutionEvidenceIds: dto.evidenceIds ?? []
      }, { merge: true });

      tx.create(this.db.collection('audit_events').doc(makeId('aud')),
{
        auditId: makeId('aud'),
        tenantId,
        type: 'LOCK_RESOLVED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { lockId, notes: dto.resolutionNotes ?? '', evidenceIds:
dto.evidenceIds ?? [] },
        ts: now
      });

      const out = { lockId, status: 'RESOLVED' };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'locks.resolve', clientRequestId: dto.clientRequestId, createdAt: now,
result: out });
      return out;
    });
  }
}
```

5.3 Controller

```
// src/locks/locks.controller.ts
import { Body, Controller, Get, Param, Post, Query, UseGuards } from
'@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
```

```
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { LocksService } from './locks.service';

@Controller('/api/locks')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class LocksController {
  constructor(private readonly svc: LocksService) {}

  @Post()
  @Roles('AUDITOR','ADMIN','MANAGER')
  @Permissions('locks.create')
  create(@Actor() actor: any, @Body() dto: any) {
    return this.svc.create(actor, dto);
  }

  @Get()
  @Roles('AUDITOR','ADMIN','MANAGER','PRODUCER')
  @Permissions('locks.read')
  list(@Actor() actor: any, @Query('propertyId') propertyId?: string)
{
    return this.svc.listActive(actor, propertyId);
  }

  @Post('/:id/resolve')
  @Roles('AUDITOR','ADMIN')
  @Permissions('locks.resolve')
  resolve(@Actor() actor: any, @Param('id') id: string, @Body() dto:
any) {
    return this.svc.resolve(actor, id, dto);
  }
}
```

---

6) Firestore Rules (coleção locks)

```
match /locks/{lockId} {
  allow create: if signedIn()
    && request.auth.token.tenantId == request.resource.data.tenantId
    && hasPerm('locks.create')
    && request.auth.token.role in ['AUDITOR','ADMIN','MANAGER'];

  allow get, list: if signedIn()
    && request.auth.token.tenantId == resource.data.tenantId
    && hasPerm('locks.read');

  // update só resolve (não deixa mudar reason/effects/scope)
  allow update: if signedIn()
    && request.auth.token.tenantId == resource.data.tenantId
    && hasPerm('locks.resolve')
    && request.auth.token.role in ['AUDITOR','ADMIN']
    && request.resource.data.tenantId == resource.data.tenantId
    && request.resource.data.scope == resource.data.scope
    && request.resource.data.reason == resource.data.reason
    && request.resource.data.effects == resource.data.effects;

  allow delete: if false;
}
```

---

7) Integração com “ativos de terceiros em custódia” e NF travada

Você pediu:

“travar a emissão de notas da propriedade onde se encontra ativos de terceiros em custódia, sem verificação ou pagamento… ao gerar a nota o escrow split deve funcionar”.

Como aplicar com locks:

✅ quando existe custódia ativa (bovinos/mercadoria de terceiros) **e** ainda não houve split/pagamento:

- criar lock `THIRD_PARTY_CUSTODY` com `effects.blockInvoice = true` e `blockTransferAssets = true`

✅ quando o escrow/split da operação for liquidado:

- resolver lock automaticamente (ou criar evento de “unlock”)

Isso fecha fraude de “vender e não repassar”.

---

8) Permissions sugeridas

- `locks.create` (AUDITOR/ADMIN/MANAGER)
- `locks.read` (AUDITOR/ADMIN/MANAGER/PRODUCER)
- `locks.resolve` (AUDITOR/ADMIN)

Se você disser “sim”, o próximo bloco que fecha 100% o agro é:

- **Ativos identificáveis vs não-identificáveis** (brinco/RFID + regra “filho segue mãe” + quotas proporcionais quando não separável)
  com split e custódia.

Fechado — abaixo vai o módulo completo de **Ativos Identificáveis vs Não-Identificáveis** (com bovinos “brinco/RFID”, regra “filho segue mãe”, custódia/terceiros, quotas proporcionais quando não separável) e como isso se liga ao **escrow/split** e aos **locks (CAR/IE)**.

---

1) Conceito central: 2 classes de ativos

A) Identificáveis (unitários)

Ex.: bovino com brinco/RFID, máquina com chassi, saco com serial, produto com QR unitário.

- ✅ Cada unidade tem `assetId` único
- ✅ Transferência, custódia, venda e split são por unidade

B) Não-identificáveis / Misturáveis

Ex.: grãos em silo, leite em tanque (mistura), insumo a granel, lote de produção sem segregação.

- ✅ O "ativo" é um **pool**
- ✅ Propriedade é por **quotas** (% ou frações)
- ✅ Saídas (vendas) fazem split proporcional às quotas (ou regra de contrato)

---

2) Modelo de dados (Firestore)

2.1 Ativos unitários (bovinos etc.)

```
assets/{assetId}

{
  "tenantId":"TENANT_FARM",
  "assetId":"cow_000123",
  "type":"BOVINE|OVINE|FISH|MACHINE|UNIT_GOOD",
  "identity":{
    "idType":"EAR_TAG|RFID|QR|SERIAL",
    "idValue":"BR123456",
    "secondary":"RFID-9988"
  },
  "species":"BOVINE",
  "sex":"F|M",
  "birth":{
    "birthDate":"2025-09-01",
    "motherAssetId":"cow_000045",
    "fatherAssetId":null
  },
  "ownership":{
    "mode":"DIRECT|CUSTODY|LEASE",
    "ownerTenantId":"TENANT_OWNER_ABC",
    "custodianTenantId":"TENANT_FARM",
    "titleDocRefs":["CTR_001","NF_..."],
    "locked":false
  },
  "location":{
    "propertyId":"prop_123",
    "geofenceId":"gf_01"
  },
  "status":"ACTIVE|SOLD|DEAD|TRANSFERRED",
```

```
  "createdAt":1730000000000
}
```

Regra “filho segue mãe”

- Quando nasce um bezerro (novo asset):
  - por padrão `ownerTenantId = ownerTenantId da mãe`
  - custódia = propriedade onde nasceu (se diferente do owner)

Isso garante rastreabilidade e direito de propriedade.

2.2 Pools (mistura / não separável)

```
asset_pools/{poolId}
```

```
{
  "tenantId":"TENANT_FARM",
  "poolId":"pool_milk_tank_01",
  "type":"MILK_TANK|GRAIN_SILO|BULK_INVENTORY",
  "unitBase":"L|KG",
  "propertyId":"prop_123",
  "currentQtyBase": 12000,
  "status":"ACTIVE",
  "createdAt":1730000000000
}
```

Quotas do pool (quem é dono de quanto)

```
asset_pools/{poolId}/shares/{shareId}
```

```
{
  "tenantId":"TENANT_FARM",
  "poolId":"pool_milk_tank_01",
  "shareId":"sh_ownerA",
  "ownerTenantId":"TENANT_OWNER_A",
  "basis":"PERCENT|UNITS",
  "percent": 0.40,
  "unitsBase": null,
  "lockedUnitsBase": 0,
  "createdAt":1730000000000
}
```

Para ser juridicamente forte, prefira **UNITS** quando possível (litros/kg) e use PERCENT quando a medição for contínua.

---

3) Custódia e ativos de terceiros (o que você pediu)

Quando o ativo está na fazenda, mas pertence a terceiros:

- `ownership.mode = CUSTODY`

- `ownerTenantId` ≠ `custodianTenantId`

3.1 Lock automático (impedir fraude)

Se houver qualquer ativo em custódia na propriedade e houver operação de NF/venda que possa prejudicar o dono:

✅ criar lock `THIRD_PARTY_CUSTODY` com:

- `blockInvoice = true`
- `blockTransferAssets = true`
- (opcional) `blockSale = true` para lotes suspeitos

A liberação do lock ocorre quando:

- escrow/split do dono estiver liquidado **e**
- evidência de operação concluída estiver validada

---

4) Operações principais (com split e escrow)

4.1 Transferir ativo unitário (bovino)

```
POST /api/assets/:id/transfer
```

- exige evidência Tipo B (foto + GPS + timestamp)
- pode exigir assinatura digital do técnico/auditor conforme risco

Fluxo:

1. criar evento `ASSET_TRANSFER_REQUESTED` (escrow opcional se venda)
2. validar máquina de estados
3. se "transferência com venda":
    - cria escrow
    - paga → libera
4. baixa custódia e muda owner/custodian

4.2 Venda de bovinos em boitel (segurança do dono)

Você quer "segurança total do dono do ativo".

A modelagem ideal:

- Boitel é `custodianTenantId`
- Dono é `ownerTenantId`
- Toda "movimentação" do boitel vira evento com evidência
- Se boitel tentar vender/transferir sem autorização:
    - bloqueia via lock + RBAC + RulesEngine

E o pagamento:

- Ao vender, o escrow split manda direto:
  - parte do dono (quase tudo)
  - parte do boitel (taxa por diária/serviço), se prevista no contrato

Isso impede calote: boitel não "segura" o dinheiro do dono.

4.3 Pool de leite (resolvendo o problema crônico)

Você descreveu: indústria paga com atraso, produtor não sabe preço, calotes.

Solução técnica com pool + escrow por coleta:

A) Medição diária (ticket)

- `tank_checks` diários (conferente) com evidência Tipo B
- gera "ticket diário" imutável
- soma do período (30 dias) fica calculável e auditável

B) Contrato de compra (demanda/volume)

- Indústria define:
  - preço base e fórmula (ex.: index + qualidade)
  - janela de coleta
  - volume mínimo do grupo
- Produtores formam grupo regional para garantir volume

C) Escrow por coleta (o pulo do gato)

- A cada coleta:
  - produtor lança entrega + evidência no tanque
  - conferente valida
  - indústria deposita no escrow
  - split automático para cada produtor conforme volume (e qualidade)

Mesmo se o volume oscilar, o modelo funciona porque:

- escrow é por coleta/periodicidade
- preço pode ser fórmula; pagamento usa dados do ticket + análise laboratorial
- se indústria não depositar → lock `NON_PAYMENT`/`CONTRACT_BREACH` pode bloquear compras futuras e ações internas do sistema

---

5) Quando ativos não são separáveis: split proporcional por quota/NF

Você pediu:

“se for impossível separar identidade deve ser proporcional às quotas… se mercadoria não identificável proporcional à NF”.

Implementação:

5.1 Saída de pool (ex.: leite do tanque)

`POST /api/pools/:poolId/withdraw`
Payload:

```
{
  "clientRequestId":"uuid",
  "qtyBase": 2000,
  "reason":"COLLECTION",
  "evidenceIds":["evg_photo","evg_gps"]
}
```

O backend:

1. verifica `pool.currentQtyBase >= qtyBase`
2. calcula split por shares:
   - se shares em UNITS: proporcional às unidades
   - se shares em PERCENT: `qty * percent`
3. gera `escrow` com split para cada `ownerTenantId`
4. debita pool e debita shares (lockedUnitsBase se necessário)

Isso faz a proporcionalidade “automática e justa”.

---

6) Endpoints mínimos (NestJS)

6.1 Assets (unitários)

- `POST /api/assets` (criar / cadastrar)
- `POST /api/assets/birth` (nascimento; aplica regra filho→mãe)
- `POST /api/assets/:id/transfer`
- `GET /api/assets?propertyId=...`

6.2 Pools

- `POST /api/pools` (criar pool tanque/silo)
- `POST /api/pools/:id/shares` (definir quotas)
- `POST /api/pools/:id/deposit` (entrada)
- `POST /api/pools/:id/withdraw` (saída com split escrow)

6.3 Custódia + locks

- `POST /api/locks` (já tem)
- `POST /api/locks/:id/resolve`

---

7) Regras críticas (RulesEngine)

7.1 Filho segue mãe (BirthRule)

- se `motherAssetId` existe:
  - pega owner da mãe
  - define owner do filho igual

7.2 Custódia (CustodyNoTransferRule)

- se `ownership.mode=CUSTODY` e actor não é owner nem auditor:
  - bloqueia transferência/venda

7.3 Pool withdraw (PoolSharesIntegrityRule)

- soma shares percent = 100% (ou unidades total coerente)
- não permitir withdraw que deixe share negativo
- evidência obrigatória se for "coleita/venda"

7.4 Locks (já implementado)

- bloquear NF/venda/transferência se houver lock ativo (CAR/IE/property)

---

8) O que eu recomendaria como próximo passo (para fechar "agro todo")

1. **Laboratório/qualidade** (leite, grãos): laudo entra como evidência e muda preço automático
2. **Drones/maquinário/IA**: anexar laudos e mapas como evidência Tipo B+ (mídia + georreferência)
3. **Integração NF-e**: o "blockInvoice" vira impedimento real no fluxo de emissão
4. **Módulo Leiloeiro** (ativos não comprometidos): já encaixa com o asset unitário + escrow

---

Se você quiser, eu envio agora o **código NestJS** de:

- `assets/birth` (filho segue mãe)

- `pools/withdraw` (split proporcional + escrow)
- e o gatilho automático de lock `THIRD_PARTY_CUSTODY` quando existir custódia pendente na propriedade.

A seguir está o **código NestJS completo** para:

1. `POST /api/assets/birth` (**filho segue mãe**)
2. `POST /api/pools/:id/withdraw` (**split proporcional + escrow**)
3. **Gatilho automático**: cria lock `THIRD_PARTY_CUSTODY` quando existir custódia pendente na propriedade e tentarem vender/emitir nota/transferir

Vou manter o padrão: Firestore + transaction + idempotência + auditoria.

---

1) `POST /api/assets/birth` — filho segue mãe

1.1 DTO

```
// src/assets/dto/asset-birth.dto.ts
import { IsNotEmpty, IsOptional, IsString } from 'class-validator';

export class AssetBirthDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsString() @IsNotEmpty()
  propertyId: string;

  @IsString() @IsNotEmpty()
  species: 'BOVINE'|'OVINE'|'FISH'|'AVIAN';

  @IsString() @IsNotEmpty()
  motherAssetId: string;

  @IsOptional() @IsString()
  fatherAssetId?: string;

  @IsOptional() @IsString()
  birthDate?: string; // YYYY-MM-DD

  @IsOptional() @IsString()
  sex?: 'F'|'M';

  @IsOptional() @IsString()
  earTag?: string;

  @IsOptional() @IsString()
  rfid?: string;

  @IsOptional()
  evidenceIds?: string[];
}
```

1.2 Service

```
// src/assets/assets-birth.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
```

```
import { makeId } from '../common/utils/id.util';

@Injectable()
export class AssetsBirthService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async birth(actor: any, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'assets.birth');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
      if (idem.exists) return idem.data()!.result;

      // property
      const propRef =
this.db.collection('properties').doc(dto.propertyId);
      const propSnap = await tx.get(propRef);
      if (!propSnap.exists) throw new Error('PROPERTY_NOT_FOUND');
      const prop = propSnap.data()!;
      if (prop.tenantId !== tenantId) throw new
Error('FORBIDDEN_PROPERTY');

      // mother
      const motherRef =
this.db.collection('assets').doc(dto.motherAssetId);
      const motherSnap = await tx.get(motherRef);
      if (!motherSnap.exists) throw new Error('MOTHER_NOT_FOUND');
      const mother = motherSnap.data()!;
      if (mother.tenantId !== tenantId) throw new
Error('FORBIDDEN_MOTHER');

      // regra: filho segue mãe
      const ownerTenantId = mother.ownership?.ownerTenantId;
      if (!ownerTenantId) throw new Error('MOTHER_OWNER_MISSING');

      const assetId = makeId('asset');
      const childRef = this.db.collection('assets').doc(assetId);

      const identity: any = {};
      if (dto.earTag) { identity.idType = 'EAR_TAG'; identity.idValue
= dto.earTag; }
      if (dto.rfid) { identity.secondary = dto.rfid; }
      if (!identity.idType) { identity.idType = 'INTERNAL';
identity.idValue = assetId; }

      tx.create(childRef, {
        tenantId,
        assetId,
        type: dto.species === 'BOVINE' ? 'BOVINE' : 'UNIT_ASSET',
        species: dto.species,
        sex: dto.sex ?? null,
        birth: {
          birthDate: dto.birthDate ?? null,
          motherAssetId: dto.motherAssetId,
          fatherAssetId: dto.fatherAssetId ?? null
```

```
        },
        identity,
        ownership: {
          mode: mother.ownership?.mode ?? 'DIRECT',
          ownerTenantId,                 // ✅ filho herda o owner da
mãe
          custodianTenantId: tenantId,   // custódia na fazenda atual
          titleDocRefs: [],
          locked: false
        },
        location: {
          propertyId: dto.propertyId,
          geofenceId: prop.geofenceId ?? null
        },
        status: 'ACTIVE',
        createdAt: now,
        createdBy: { uid: actor.uid, role: actor.role }
      });

      // auditoria
      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        type: 'ASSET_BIRTH_CREATED_CHILD_INHERITS_MOTHER_OWNER',
        actor: { uid: actor.uid, role: actor.role },
        meta: { assetId, motherAssetId: dto.motherAssetId,
ownerTenantId, propertyId: dto.propertyId, evidenceIds:
dto.evidenceIds ?? [] },
        ts: now
      });

      const out = { assetId, ownerTenantId, status: 'ACTIVE' };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'assets.birth', clientRequestId: dto.clientRequestId, createdAt: now,
result: out });
      return out;
    });
  }
}
```

1.3 Controller

```
// src/assets/assets.controller.ts
import { Body, Controller, Post, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { AssetsBirthService } from './assets-birth.service';

@Controller('/api/assets')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class AssetsController {
  constructor(private readonly birthSvc: AssetsBirthService) {}

  @Post('/birth')
  @Roles('ADMIN','MANAGER','OPERATOR','TECHNICIAN','PRODUCER')
```

```
  @Permissions('assets.birth')
  birth(@Actor() actor: any, @Body() dto: any) {
    return this.birthSvc.birth(actor, dto);
  }
}
```

---

2) `POST /api/pools/:id/withdraw` — split proporcional + escrow

2.1 DTO

```
// src/pools/dto/pool-withdraw.dto.ts
import { IsNotEmpty, IsNumber, IsOptional, IsString } from 'class-
validator';

export class PoolWithdrawDto {
  @IsString() @IsNotEmpty()
  clientRequestId: string;

  @IsNumber()
  qtyBase: number;

  @IsOptional() @IsString()
  reason?: string;

  @IsOptional()
  evidenceIds?: string[];

  // opcional: preço do período/coleta (se contrato definir)
  @IsOptional()
  unitPriceCentsBase?: number;
}
```

2.2 Service

```
// src/pools/pools-withdraw.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';

type Share = { ownerTenantId: string; basis: 'PERCENT'|'UNITS';
percent?: number; unitsBase?: number; lockedUnitsBase?: number };

function allocateByShares(qtyBase: number, shares: Share[]) {
  // retorna allocations: ownerTenantId -> qtyBaseAlloc
  const alloc: Record<string, number> = {};

  const basisUnits = shares.some(s => s.basis === 'UNITS' &&
(s.unitsBase ?? 0) > 0);
  if (basisUnits) {
    // proporcional às unidades disponíveis
    const totalUnits = shares.reduce((a, s) => a + Math.max(0,
Number(s.unitsBase || 0) - Number(s.lockedUnitsBase || 0)), 0);
    if (totalUnits <= 0) throw new
Error('POOL_SHARES_NO_AVAILABLE_UNITS');
    let remaining = qtyBase;

    // primeiro rateia por proporção, depois ajusta restos
    for (const s of shares) {
      const avail = Math.max(0, Number(s.unitsBase || 0) -
Number(s.lockedUnitsBase || 0));
```

```
      const part = Math.min(avail, Math.floor((qtyBase * avail) /
totalUnits));
      alloc[s.ownerTenantId] = (alloc[s.ownerTenantId] || 0) + part;
      remaining -= part;
    }
    // distribui o restante 1 a 1 em quem ainda tem disponibilidade
    for (const s of shares) {
      if (remaining <= 0) break;
      const avail = Math.max(0, Number(s.unitsBase || 0) -
Number(s.lockedUnitsBase || 0));
      const used = alloc[s.ownerTenantId] || 0;
      if (used < avail) {
        alloc[s.ownerTenantId] = used + 1;
        remaining -= 1;
      }
    }
    if (remaining > 0) throw new
Error('POOL_SHARES_INSUFFICIENT_UNITS');
    return alloc;
  }

  // PERCENT
  const sumPct = shares.reduce((a, s) => a + Number(s.percent || 0),
0);
  if (Math.round(sumPct * 1000) !== 100000) { // 100.000%
    throw new Error('POOL_SHARES_PERCENT_NOT_100');
  }

  let remaining = qtyBase;
  for (const s of shares) {
    const part = Math.floor(qtyBase * (Number(s.percent || 0) / 100));
    alloc[s.ownerTenantId] = (alloc[s.ownerTenantId] || 0) + part;
    remaining -= part;
  }
  // ajusta restos
  const owners = shares.map(s => s.ownerTenantId);
  let i = 0;
  while (remaining > 0) {
    const o = owners[i % owners.length];
    alloc[o] = (alloc[o] || 0) + 1;
    remaining -= 1;
    i++;
  }
  return alloc;
}

@Injectable()
export class PoolsWithdrawService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async withdraw(actor: any, poolId: string, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'pools.withdraw');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
```

```
      if (idem.exists) return idem.data()!.result;

      const poolRef = this.db.collection('asset_pools').doc(poolId);
      const poolSnap = await tx.get(poolRef);
      if (!poolSnap.exists) throw new Error('POOL_NOT_FOUND');
      const pool = poolSnap.data()!;
      if (pool.tenantId !== tenantId) throw new
Error('FORBIDDEN_POOL');

      const qtyBase = Number(dto.qtyBase || 0);
      if (!(qtyBase > 0)) throw new Error('INVALID_QTY');

      const cur = Number(pool.currentQtyBase || 0);
      if (cur < qtyBase) throw new Error('POOL_INSUFFICIENT_QTY');

      // carrega shares
      const sharesSnap = await tx.get(

this.db.collection('asset_pools').doc(poolId).collection('shares')
      );

      const shares: Share[] = sharesSnap.docs.map(d => d.data() as
any);
      if (!shares.length) throw new Error('POOL_SHARES_EMPTY');

      // aloca por shares
      const alloc = allocateByShares(qtyBase, shares);

      // cria "escrow de pool" (sem PSP aqui, apenas espelho)
      const escrowId = makeId('esc_pool');
      const amountCents = dto.unitPriceCentsBase
        ? Math.round(qtyBase * Number(dto.unitPriceCentsBase))
        : 0; // se preço for calculado depois (ex.: qualidade)

      const targets = Object.entries(alloc).map(([ownerTenantId, q])
=> ({
        ownerTenantId,
        qtyBase: q,
        amountCents: dto.unitPriceCentsBase ? Math.round(q *
Number(dto.unitPriceCentsBase)) : 0
      }));

      tx.create(this.db.collection('escrows_pool').doc(escrowId), {
        tenantId,
        escrowId,
        poolId,
        qtyBase,
        unitBase: pool.unitBase,
        amountCents,
        targets,
        status: 'HELD', // ou REQUESTED se quiser integrar PSP por
coleta
        createdAt: now,
        createdBy: { uid: actor.uid, role: actor.role }
      });

      // debita pool (quantidade)
      tx.set(poolRef, { currentQtyBase: cur - qtyBase }, { merge: true
});

      // debita shares (UNITS) ou mantém percent (PERCENT)
```

```
      for (const sDoc of sharesSnap.docs) {
        const s = sDoc.data() as any;
        if (s.basis === 'UNITS') {
          const take = Number(alloc[s.ownerTenantId] || 0);
          if (take > 0) {
            const newUnits = Number(s.unitsBase || 0) - take;
            if (newUnits < 0) throw new Error('SHARE_NEGATIVE_UNITS');
            tx.set(sDoc.ref, { unitsBase: newUnits }, { merge: true
});
          }
        }
      }

      // auditoria
      const auditId = makeId('aud');
      tx.create(this.db.collection('audit_events').doc(auditId), {
        auditId,
        tenantId,
        type: 'POOL_WITHDRAW_ESCROW_SPLIT_CREATED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { poolId, escrowId, qtyBase, unitBase: pool.unitBase,
targets, evidenceIds: dto.evidenceIds ?? [], reason: dto.reason ?? ''
},
        ts: now
      });

      const out = { poolId, escrowId, qtyBase, targets, status: 'HELD'
};
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'pools.withdraw', clientRequestId: dto.clientRequestId, createdAt:
now, result: out });
      return out;
    });
  }
}
```

2.3 Controller

```
// src/pools/pools.controller.ts
import { Body, Controller, Param, Post, UseGuards } from
'@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { PoolsWithdrawService } from './pools-withdraw.service';

@Controller('/api/pools')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class PoolsController {
  constructor(private readonly withdrawSvc: PoolsWithdrawService) {}

  @Post('/:id/withdraw')
  @Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN')
  @Permissions('pools.withdraw')
  withdraw(@Actor() actor: any, @Param('id') id: string, @Body() dto:
any) {
    return this.withdrawSvc.withdraw(actor, id, dto);
  }
```

```
}
```

---

3) Gatilho automático: lock `THIRD_PARTY_CUSTODY` na propriedade

Você quer que, se existirem ativos de terceiros na propriedade **sem quitação**, o sistema bloqueie:

- emissão de nota
- transferência
- venda

3.1 Como detectar "custódia pendente"

Critério técnico:

- existe `assets` onde:
  - `location.propertyId == propertyId`
  - `ownership.mode == 'CUSTODY'`
  - `ownership.ownerTenantId != tenantId`
  - `status == 'ACTIVE'`
  - e (opcional) `ownership.locked == true` ou "sem evento de quitação"

3.2 Service: CustodyLockEnforcer

```
// src/locks/custody-lock-enforcer.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class CustodyLockEnforcer {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async ensureCustodyLockIfNeeded(tenantId: string, propertyId:
string, link: any) {
    // procura ativos de terceiros em custódia na propriedade
    const snap = await this.db.collection('assets')
      .where('tenantId', '==', tenantId)
      .where('location.propertyId', '==', propertyId)
      .where('ownership.mode', '==', 'CUSTODY')
      .where('status', '==', 'ACTIVE')
      .limit(50)
      .get();

    const hasThirdParty = snap.docs.some(d => {
      const a = d.data()!;
      return a.ownership?.ownerTenantId && a.ownership.ownerTenantId
!== tenantId;
    });

    if (!hasThirdParty) return { ok: true, created: false };

    // já existe lock ativo?
    const lockSnap = await this.db.collection('locks')
      .where('tenantId','==',tenantId)
```

```
      .where('status','==','ACTIVE')
      .where('reason.code','==','THIRD_PARTY_CUSTODY')
      .where('scope.type','==','PROPERTY')
      .where('scope.propertyId','==',propertyId)
      .limit(1)
      .get();

    if (!lockSnap.empty) return { ok: true, created: false, lockId:
lockSnap.docs[0].data()!.lockId };

    // cria lock
    const lockId = makeId('lock');
    await this.db.collection('locks').doc(lockId).set({
      tenantId,
      lockId,
      scope: { type: 'PROPERTY', propertyId },
      reason: {
        code: 'THIRD_PARTY_CUSTODY',
        title: 'Ativos de terceiros em custódia',
        details: 'Existem ativos de terceiros na propriedade sem
quitação/split confirmado.'
      },
      effects: {
        blockSale: true,
        blockInvoice: true,
        blockTransferAssets: true,
        blockNewContracts: false
      },
      links: link ?? {},
      status: 'ACTIVE',
      createdAt: Date.now(),
      createdBy: { uid: 'system', role: 'SYSTEM' }
    }, { merge: false });

    await this.db.collection('audit_events').add({
      auditId: makeId('aud'),
      tenantId,
      type: 'AUTO_LOCK_THIRD_PARTY_CUSTODY_CREATED',
      meta: { lockId, propertyId, link },
      ts: Date.now()
    });

    return { ok: true, created: true, lockId };
  }
}
```

3.3 Onde chamar (pontos críticos)

- Ao tentar criar venda `sales.create`
- Ao tentar emitir NF (seu módulo fiscal)
- Ao tentar transferir ativo `assets.transfer`

Exemplo no `SalesCreateService` antes do LockCheck:

```
await this.custodyLock.ensureCustodyLockIfNeeded(tenantId,
dto.propertyId, { saleId: 'PENDING' });
const { locks } = await
this.lockCheck.getActiveLocksForProperty(tenantId, dto.propertyId);
this.lockCheck.enforceEffects(locks, ['blockSale']);
```

✅ Isso automatiza o bloqueio — sem depender de humano.

---

4) Permissions (mínimas)

- `assets.birth`
- `pools.withdraw`
- `locks.read`
- `locks.create` (só humano; o auto-lock usa service account/sistema)

---

5) Observações práticas (para não dar dor de cabeça)

1. **Indexação Firestore**: você precisará de índices compostos para consultas com `.where` múltiplos (console do Firestore aponta).
2. Para `shares` em PERCENT, o withdraw não altera shares; para UNITS, altera — escolha conforme o tipo de pool.
3. No leite, recomendo shares por **UNITS** (litros por produtor por período/coleta) porque fica incontestável.

Se você quiser, o próximo passo que deixa o leite perfeito é:

- `tank_checks` (ticket diário) + integração com análise laboratorial + cálculo automático do preço por fórmula + escrow por coleta (split por produtor).

Perfeito — abaixo vai o módulo **LEITE COMPLETO**: `tank_checks` (ticket diário), análise laboratorial, fórmula de preço, grupos regionais, escrow por coleta com split por produtor e travas anti-calote.

Vou entregar em blocos:

1. Modelo de dados (Firestore)
2. Fluxo operacional (dia a dia)
3. Fórmula de preço (motor de regras)
4. Endpoints NestJS (com transação + auditoria)
5. Locks anti-calote e governança do grupo

---

1) Modelo de dados (Firestore)

1.1 Grupo regional de produtores (para garantir volume)

```
milk_groups/{groupId}
```

```
{
  "tenantId":"TENANT_PLATFORM",
  "groupId":"grp_milk_mt_01",
  "name":"Grupo Leite MT Norte",
  "geo": { "centerLat": -15.7, "centerLng": -56.1, "radiusKm": 80 },
  "industryTenantId":"TENANT_INDUSTRY_01",
  "status":"ACTIVE",
  "rules": {
    "minVolumePerCollectionL": 5000,
    "collectionWindowHours": 24,
    "ticketDailyRequired": true
  },
  "createdAt":1730000000000
}
```

1.2 Membros do grupo

```
milk_groups/{groupId}/members/{memberId}
```

```
{
  "tenantId":"TENANT_PLATFORM",
  "groupId":"grp_milk_mt_01",
  "memberId":"TENANT_PRODUCER_A",
  "producerTenantId":"TENANT_PRODUCER_A",
  "propertyId":"prop_A1",
  "tankPoolId":"pool_milk_tank_A1",
  "status":"ACTIVE",
  "payment": {
    "walletRef":"asaasWalletProducerA"
  }
}
```

1.3 Ticket diário do tanque (conferente)

```
tank_checks/{checkId}
```

```
{
  "tenantId":"TENANT_PLATFORM",
  "checkId":"chk_2026_02_05_A1",
  "groupId":"grp_milk_mt_01",
  "producerTenantId":"TENANT_PRODUCER_A",
  "propertyId":"prop_A1",
  "tankPoolId":"pool_milk_tank_A1",
  "date":"2026-02-05",
  "measuredLiters": 820,
  "temperatureC": 4.2,
  "notes":"Sem anomalias",
  "evidenceIds":["evg_photo_tank","evg_gps","evg_voice"],
  "performedBy": { "uid":"u_conf_01", "role":"TECHNICIAN" },
  "status":"RECORDED|VALIDATED",
  "createdAt":1730000000000
}
```

1.4 Coleta (evento que gera escrow)

```
milk_collections/{collectionId}
```

```
{
  "tenantId":"TENANT_PLATFORM",
  "collectionId":"col_2026_02_05_grp01",
  "groupId":"grp_milk_mt_01",
```

```
  "industryTenantId":"TENANT_INDUSTRY_01",
  "date":"2026-02-05",
  "status":"DRAFT|READY_FOR_PAYMENT|PAID|DISPUTE",
  "totalLiters": 0,
  "members":[
    { "producerTenantId":"TENANT_PRODUCER_A", "liters": 820,
"checkIds":["chk_..."] },
    { "producerTenantId":"TENANT_PRODUCER_B", "liters": 610,
"checkIds":["chk_..."] }
  ],
  "lab": {
    "labReportId": null,
    "qualityIndex": null
  },
  "pricing": {
    "formulaId":"milk_formula_v1",
    "basePriceCentsPerL": 0,
    "finalPriceCentsPerL": 0
  },
  "escrow": { "escrowId": null, "status": null },
  "createdAt":1730000000000
}
```


1.5 Laudo laboratorial

```
lab_reports/{labReportId}
```


```
{
  "tenantId":"TENANT_PLATFORM",
  "labReportId":"lab_2026_02_05_grp01",
  "groupId":"grp_milk_mt_01",
  "collectionId":"col_2026_02_05_grp01",
  "metrics":{
    "fat": 3.6,
    "protein": 3.2,
    "somaticCells": 280000,
    "bacterialCount": 12000
  },
  "evidenceIds":["evg_pdf_lab","evg_signature"],
  "issuedAt":1730000000000
}
```


1.6 Escrow e split por produtor (o anti-calote)

```
escrows_milk/{escrowId}
```


```
{
  "tenantId":"TENANT_PLATFORM",
  "escrowId":"esc_milk_001",
  "groupId":"grp_milk_mt_01",
  "collectionId":"col_2026_02_05_grp01",
  "industryTenantId":"TENANT_INDUSTRY_01",
  "amountCents": 0,
  "targets":[
    { "producerTenantId":"TENANT_PRODUCER_A", "liters": 820,
"amountCents": 0 },
    { "producerTenantId":"TENANT_PRODUCER_B", "liters": 610,
"amountCents": 0 }
  ],
  "status":"REQUESTED|PAID|HOLD_FOR_DISPUTE|REFUNDED",
  "createdAt":1730000000000
```

```
}
```

---

2) Fluxo operacional (simples no campo)

2.1 Todo dia (ticket)

1. Conferente vai ao tanque (ou produtor com validação posterior)
2. Mede litros e temperatura
3. Tira foto do tanque + GPS + hora
4. (Opcional) comando de voz descrevendo
5. Envia → `tank_check RECORDED`

2.2 No dia da coleta

1. Sistema junta tickets do período (ex.: 24h ou último dia)
2. Gera `milk_collection` com total por produtor
3. Indústria anexa laudo (ou posterior)
4. Motor calcula preço final (fórmula)
5. Cria cobrança/escrow e split por produtor
6. Pagou → libera automaticamente para produtores (ou na próxima coleta, conforme contrato)

✅ produtor ganha "direito a receber" e vê o valor previsto.

---

3) Fórmula de preço (RulesEngine)

3.1 Fórmula armazenada

```
pricing_formulas/{formulaId}

{
  "tenantId":"TENANT_PLATFORM",
  "formulaId":"milk_formula_v1",
  "type":"MILK",
  "basePriceCentsPerL": 220,
  "adjustments":[
    { "metric":"fat", "op":">=", "value":3.5, "deltaCents": +10 },
    { "metric":"protein", "op":">=", "value":3.1, "deltaCents": +10 },
    { "metric":"somaticCells", "op":"<=", "value":400000,
"deltaCents": +5 },
    { "metric":"bacterialCount", "op":"<=", "value":20000,
"deltaCents": +5 },
    { "metric":"somaticCells", "op":">", "value":800000, "deltaCents":
-30 }
  ],
  "floorCentsPerL": 160,
  "capCentsPerL": 320,
  "active": true
}
```

3.2 Cálculo

```
finalPrice = clamp(base + soma(adjustments aplicáveis), floor, cap)
```

---

4) Endpoints NestJS (com transação + auditoria)

4.1 `POST /api/milk/tank-checks`

Cria ticket diário.

Service (resumo, padrão transação)

- valida membership no grupo
- exige evidência Tipo B (foto+gps+timestamp) para “VALIDATED”
- grava check
- auditoria

4.2 `POST /api/milk/collections/build`

Constrói uma coleta do dia:

- junta `tank_checks` do dia/intervalo por membro
- soma litros por produtor
- cria `milk_collection` com status `DRAFT`

4.3 `POST /api/milk/collections/:id/attach-lab`

Anexa laudo + evidências.

4.4 `POST /api/milk/collections/:id/price`

Calcula preço usando fórmula.

4.5 `POST /api/milk/collections/:id/pay`

Cria escrow e split por produtor (REQUESTED).

4.6 Webhook “PAID”

- marca escrow PAID
- marca collection PAID
- cria payouts por produtor (ou split nativo se PSP suportar)

---

5) Travas anti-calote (o que resolve seu problema)

5.1 Lock por inadimplência da indústria

Se indústria não pagar no prazo:

- cria lock `NON_PAYMENT` para `industryTenantId`
- efeitos:
  - `blockNewContracts = true`
  - `blockMarketplaceAccess = true` (se quiser)
  - e notifica auditores

5.2 Produtor sempre vê:

- litros validados
- preço base e fórmula
- previsão de pagamento
- status do escrow

5.3 "Ticket" é direito de receber

O ticket diário + coleta consolidada viram **lastro probatório**:

- se não pagar, o sistema gera pacote de evidências (audit log + hashes)

Quer o código agora?

Se você disser "sim", eu mando o **código NestJS completo** (controllers + services) para:

- `tank-checks` (criar/validar)
- `collections/build`
- `attach-lab`
- `price`
- `pay`
- webhook PAID com split por produtor

(com idempotência + auditoria + locks por inadimplência).

A seguir está o **código NestJS completo** (controllers + services) para o módulo do **leite**:

- `POST /api/milk/tank-checks` (criar ticket diário)
- `POST /api/milk/collections/build` (montar coleta consolidada)
- `POST /api/milk/collections/:id/attach-lab` (anexar laudo)
- `POST /api/milk/collections/:id/price` (calcular preço por fórmula)
- `POST /api/milk/collections/:id/pay` (criar escrow + split por produtor)
- `POST /api/webhooks/asaas/milk` (PAID → payouts + auditoria)
- Locks por inadimplência da indústria (`NON_PAYMENT`)

Observação: estou usando Firestore transaction + idempotência + auditoria como você definiu. Integração real com Asaas fica stubada (mas com metadata para plugar).

---

0) Coleções usadas

- `milk_groups`, `milk_groups/{groupId}/members`
- `tank_checks`
- `milk_collections`
- `lab_reports`
- `pricing_formulas`
- `escrows_milk`
- `producer_payouts`
- `locks`
- `audit_events`
- `idempotency`

---

1) Util: idempotência e ids

(Pressupondo que você já tem `idemKey` e `makeId`)

---

2) Tank Checks

2.1 DTO

```
// src/milk/dto/tank-check-create.dto.ts
import { IsNotEmpty, IsNumber, IsOptional, IsString } from 'class-
validator';

export class TankCheckCreateDto {
  @IsString() @IsNotEmpty() clientRequestId: string;

  @IsString() @IsNotEmpty() groupId: string;
  @IsString() @IsNotEmpty() producerTenantId: string;
  @IsString() @IsNotEmpty() propertyId: string;
  @IsString() @IsNotEmpty() tankPoolId: string;

  @IsString() @IsNotEmpty() date: string; // YYYY-MM-DD

  @IsNumber() measuredLiters: number;

  @IsOptional() temperatureC?: number;
  @IsOptional() notes?: string;

  @IsOptional() evidenceIds?: string[];

  // se true, exige evidência tipo B e papel técnico/auditor para
VALIDATED
  @IsOptional() validateNow?: boolean;
}
```

2.2 Service

```
// src/milk/tank-checks.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class TankChecksService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async create(actor: any, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId; // plataforma ou tenant do
contexto

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'milk.tankChecks.create');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
      if (idem.exists) return idem.data()!.result;

      if (!(Number(dto.measuredLiters) > 0)) throw new
Error('INVALID_MEASURED_LITERS');

      // valida que o produtor é membro do grupo
      const memRef =
this.db.collection('milk_groups').doc(dto.groupId)
        .collection('members').doc(dto.producerTenantId);

      const memSnap = await tx.get(memRef);
      if (!memSnap.exists) throw new Error('NOT_GROUP_MEMBER');
      const mem = memSnap.data()!;
      if (mem.status !== 'ACTIVE') throw new Error('MEMBER_INACTIVE');
      if (mem.propertyId !== dto.propertyId) throw new
Error('PROPERTY_MISMATCH');
      if (mem.tankPoolId !== dto.tankPoolId) throw new
Error('POOL_MISMATCH');

      const validateNow = !!dto.validateNow;
      const evidenceIds = dto.evidenceIds || [];
      let status: 'RECORDED'|'VALIDATED' = 'RECORDED';

      if (validateNow) {
        // regra simples: para validar agora precisa ser
TECHNICIAN/AUDITOR e ter evidência
        if (!['TECHNICIAN','AUDITOR','ADMIN'].includes(actor.role))
throw new Error('VALIDATION_ROLE_REQUIRED');
        if (!evidenceIds.length) throw new
Error('EVIDENCE_REQUIRED_FOR_VALIDATION');
        status = 'VALIDATED';
      }

      const checkId = makeId('chk');
      tx.create(this.db.collection('tank_checks').doc(checkId), {
        tenantId,
        checkId,
        groupId: dto.groupId,
```

```
        producerTenantId: dto.producerTenantId,
        propertyId: dto.propertyId,
        tankPoolId: dto.tankPoolId,
        date: dto.date,
        measuredLiters: Number(dto.measuredLiters),
        temperatureC: dto.temperatureC ?? null,
        notes: dto.notes ?? '',
        evidenceIds,
        performedBy: { uid: actor.uid, role: actor.role },
        status,
        createdAt: now
      });

      tx.create(this.db.collection('audit_events').doc(makeId('aud')),
{
        auditId: makeId('aud'),
        tenantId,
        type: 'TANK_CHECK_CREATED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { checkId, groupId: dto.groupId, producerTenantId:
dto.producerTenantId, date: dto.date, liters: dto.measuredLiters,
status },
        ts: now
      });

      const out = { checkId, status };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'milk.tankChecks.create', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });
      return out;
    });
  }
}
```

2.3 Controller

```
// src/milk/milk.controller.ts (parte)
import { Body, Controller, Param, Post, UseGuards } from
'@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { TankChecksService } from './tank-checks.service';

@Controller('/api/milk')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class MilkController {
  constructor(private readonly tankChecks: TankChecksService) {}

  @Post('/tank-checks')

@Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN','AUDITOR')
  @Permissions('milk.tankChecks.create')
  createTankCheck(@Actor() actor: any, @Body() dto: any) {
    return this.tankChecks.create(actor, dto);
  }
}
```

3) Build Collection (consolidar tickets em coleta)

3.1 DTO

```
// src/milk/dto/collection-build.dto.ts
import { IsNotEmpty, IsOptional, IsString } from 'class-validator';

export class CollectionBuildDto {
  @IsString() @IsNotEmpty() clientRequestId: string;
  @IsString() @IsNotEmpty() groupId: string;

  @IsString() @IsNotEmpty() date: string; // YYYY-MM-DD
  @IsOptional() onlyValidated?: boolean;  // default true
}
```

3.2 Service

```
// src/milk/collections-build.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class CollectionsBuildService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async build(actor: any, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'milk.collections.build');
    const idemRef = this.db.collection('idempotency').doc(key);

    const onlyValidated = dto.onlyValidated !== false;

    // consulta tickets do dia (fora de tx ok; mas a gravação é tx)
    let q = this.db.collection('tank_checks')
      .where('groupId', '==', dto.groupId)
      .where('date', '==', dto.date);

    if (onlyValidated) q = q.where('status', '==', 'VALIDATED');

    const checksSnap = await q.limit(500).get();
    if (checksSnap.empty) throw new Error('NO_TANK_CHECKS_FOR_DATE');

    // agrupa por produtor
    const map = new Map<string, { liters: number; checkIds: string[]
}>();
    for (const d of checksSnap.docs) {
      const c = d.data()!;
      const cur = map.get(c.producerTenantId) || { liters: 0,
checkIds: [] };
      cur.liters += Number(c.measuredLiters || 0);
      cur.checkIds.push(c.checkId);
      map.set(c.producerTenantId, cur);
    }
```

```
    const members = Array.from(map.entries()).map(([producerTenantId,
v]) => ({
      producerTenantId,
      liters: Math.round(v.liters),
      checkIds: v.checkIds
    }));
    const totalLiters = members.reduce((a, x) => a + Number(x.liters
|| 0), 0);
    if (!(totalLiters > 0)) throw new Error('TOTAL_LITERS_ZERO');

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
      if (idem.exists) return idem.data()!.result;

      // pega grupo
      const grpRef =
this.db.collection('milk_groups').doc(dto.groupId);
      const grpSnap = await tx.get(grpRef);
      if (!grpSnap.exists) throw new Error('GROUP_NOT_FOUND');
      const grp = grpSnap.data()!;
      if (grp.status !== 'ACTIVE') throw new Error('GROUP_INACTIVE');

      const collectionId = makeId('col');
      const colRef =
this.db.collection('milk_collections').doc(collectionId);

      tx.create(colRef, {
        tenantId,
        collectionId,
        groupId: dto.groupId,
        industryTenantId: grp.industryTenantId,
        date: dto.date,
        status: 'DRAFT',
        totalLiters,
        members,
        lab: { labReportId: null, qualityIndex: null },
        pricing: { formulaId: grp.pricingFormulaId ??
'milk_formula_v1', basePriceCentsPerL: 0, finalPriceCentsPerL: 0 },
        escrow: { escrowId: null, status: null },
        createdAt: now,
        createdBy: { uid: actor.uid, role: actor.role }
      });

      tx.create(this.db.collection('audit_events').doc(makeId('aud')),
{
        auditId: makeId('aud'),
        tenantId,
        type: 'MILK_COLLECTION_BUILT_FROM_TANK_CHECKS',
        actor: { uid: actor.uid, role: actor.role },
        meta: { collectionId, groupId: dto.groupId, date: dto.date,
totalLiters, membersCount: members.length, onlyValidated },
        ts: now
      });

      const out = { collectionId, status: 'DRAFT', totalLiters,
membersCount: members.length };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'milk.collections.build', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });
      return out;
    });
```

```
  }
}
```

3.3 Controller

```
// src/milk/milk.controller.ts (parte)
import { CollectionsBuildService } from './collections-build.service';

@Post('/collections/build')
@Roles('ADMIN','MANAGER','TECHNICIAN','AUDITOR')
@Permissions('milk.collections.build')
build(@Actor() actor: any, @Body() dto: any) {
  return this.collectionsBuild.build(actor, dto);
}
```

---

4) Attach Lab Report

4.1 DTO

```
// src/milk/dto/collection-attach-lab.dto.ts
import { IsNotEmpty, IsObject, IsString } from 'class-validator';

export class CollectionAttachLabDto {
  @IsString() @IsNotEmpty() clientRequestId: string;
  @IsObject() metrics: any;
  evidenceIds?: string[];
}
```

4.2 Service

```
// src/milk/collections-lab.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class CollectionsLabService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async attachLab(actor: any, collectionId: string, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'milk.collections.attachLab');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
      if (idem.exists) return idem.data()!.result;

      const colRef =
this.db.collection('milk_collections').doc(collectionId);
      const colSnap = await tx.get(colRef);
      if (!colSnap.exists) throw new Error('COLLECTION_NOT_FOUND');
      const col = colSnap.data()!;
      if (col.tenantId !== tenantId) throw new Error('FORBIDDEN');
      if (!['DRAFT','READY_FOR_PAYMENT'].includes(col.status)) throw
new Error('COLLECTION_NOT_EDITABLE');

      const labReportId = makeId('lab');
```

```
      tx.create(this.db.collection('lab_reports').doc(labReportId), {
        tenantId,
        labReportId,
        groupId: col.groupId,
        collectionId,
        metrics: dto.metrics,
        evidenceIds: dto.evidenceIds ?? [],
        issuedAt: now
      });

      tx.set(colRef, { lab: { labReportId, qualityIndex: null } }, {
merge: true });

      tx.create(this.db.collection('audit_events').doc(makeId('aud')),
{
        auditId: makeId('aud'),
        tenantId,
        type: 'MILK_COLLECTION_LAB_ATTACHED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { collectionId, labReportId },
        ts: now
      });

      const out = { collectionId, labReportId };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'milk.collections.attachLab', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });
      return out;
    });
  }
}
```

---

5) Price Collection (fórmula)

5.1 Service

```
// src/milk/collections-price.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';

function applyFormula(base: number, metrics: any, adjustments: any[],
floor: number, cap: number) {
  let v = base;
  for (const a of adjustments || []) {
    const m = Number(metrics?.[a.metric]);
    if (!Number.isFinite(m)) continue;

    const op = a.op;
    const cmp = Number(a.value);
    let ok = false;
    if (op === '>=') ok = m >= cmp;
    else if (op === '<=') ok = m <= cmp;
    else if (op === '>') ok = m > cmp;
    else if (op === '<') ok = m < cmp;
    else if (op === '==') ok = m === cmp;

    if (ok) v += Number(a.deltaCents || 0);
  }
```

```
  if (Number.isFinite(floor)) v = Math.max(v, floor);
  if (Number.isFinite(cap)) v = Math.min(v, cap);
  return v;
}

@Injectable()
export class CollectionsPriceService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async price(actor: any, collectionId: string, clientRequestId:
string) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, clientRequestId,
'milk.collections.price');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
      if (idem.exists) return idem.data()!.result;

      const colRef =
this.db.collection('milk_collections').doc(collectionId);
      const colSnap = await tx.get(colRef);
      if (!colSnap.exists) throw new Error('COLLECTION_NOT_FOUND');
      const col = colSnap.data()!;
      if (col.tenantId !== tenantId) throw new Error('FORBIDDEN');
      if (!['DRAFT','READY_FOR_PAYMENT'].includes(col.status)) throw
new Error('COLLECTION_NOT_PRICEABLE');

      // formula
      const formulaId = col.pricing?.formulaId || 'milk_formula_v1';
      const fSnap = await
tx.get(this.db.collection('pricing_formulas').doc(formulaId));
      if (!fSnap.exists) throw new Error('FORMULA_NOT_FOUND');
      const f = fSnap.data()!;
      if (!f.active) throw new Error('FORMULA_INACTIVE');

      // metrics (lab opcional; se não tiver, usa somente base)
      let metrics = {};
      if (col.lab?.labReportId) {
        const labSnap = await
tx.get(this.db.collection('lab_reports').doc(col.lab.labReportId));
        if (labSnap.exists) metrics = labSnap.data()!.metrics || {};
      }

      const base = Number(f.basePriceCentsPerL || 0);
      const final = applyFormula(
        base,
        metrics,
        f.adjustments || [],
        Number(f.floorCentsPerL || 0),
        Number(f.capCentsPerL || 999999)
      );

      tx.set(colRef, {
        pricing: {
          ...col.pricing,
          basePriceCentsPerL: base,
```

```
          finalPriceCentsPerL: final
        },
        status: 'READY_FOR_PAYMENT'
      }, { merge: true });

      tx.create(this.db.collection('audit_events').doc(makeId('aud')),
{
        auditId: makeId('aud'),
        tenantId,
        type: 'MILK_COLLECTION_PRICED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { collectionId, formulaId, basePriceCentsPerL: base,
finalPriceCentsPerL: final },
        ts: now
      });

      const out = { collectionId, status: 'READY_FOR_PAYMENT',
basePriceCentsPerL: base, finalPriceCentsPerL: final };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'milk.collections.price', clientRequestId, createdAt: now, result: out
});
      return out;
    });
  }
}
```

---

6) Pay Collection (escrow + split por produtor)

6.1 DTO

```
// src/milk/dto/collection-pay.dto.ts
import { IsNotEmpty, IsString } from 'class-validator';

export class CollectionPayDto {
  @IsString() @IsNotEmpty() clientRequestId: string;
  @IsString() @IsNotEmpty() payMethod: 'PIX'|'BOLETO'|'TRANSFER';
}
```

6.2 Service

```
// src/milk/collections-pay.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class CollectionsPayService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async pay(actor: any, collectionId: string, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'milk.collections.pay');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
      if (idem.exists) return idem.data()!.result;
```

```
      const colRef =
this.db.collection('milk_collections').doc(collectionId);
      const colSnap = await tx.get(colRef);
      if (!colSnap.exists) throw new Error('COLLECTION_NOT_FOUND');
      const col = colSnap.data()!;
      if (col.tenantId !== tenantId) throw new Error('FORBIDDEN');
      if (col.status !== 'READY_FOR_PAYMENT') throw new
Error('COLLECTION_NOT_READY_FOR_PAYMENT');

      const price = Number(col.pricing?.finalPriceCentsPerL || 0);
      if (!(price > 0)) throw new Error('PRICE_NOT_SET');

      const targets: any[] = [];
      let amountCents = 0;

      for (const m of col.members || []) {
        const liters = Number(m.liters || 0);
        if (liters <= 0) continue;
        const amt = Math.round(liters * price);
        targets.push({ producerTenantId: m.producerTenantId, liters,
amountCents: amt });
        amountCents += amt;
      }

      if (!(amountCents > 0)) throw new Error('AMOUNT_ZERO');

      const escrowId = makeId('esc_milk');
      tx.create(this.db.collection('escrows_milk').doc(escrowId), {
        tenantId,
        escrowId,
        groupId: col.groupId,
        collectionId,
        industryTenantId: col.industryTenantId,
        amountCents,
        targets,
        status: 'REQUESTED',
        payMethod: dto.payMethod,
        createdAt: now,
        createdBy: { uid: actor.uid, role: actor.role },
        // metadata p/ PSP:
        metadata: { escrowId, collectionId, groupId: col.groupId }
      });

      tx.set(colRef, {
        escrow: { escrowId, status: 'REQUESTED' },
        status: 'PAYMENT_REQUESTED'
      }, { merge: true });

      tx.create(this.db.collection('audit_events').doc(makeId('aud')),
{
        auditId: makeId('aud'),
        tenantId,
        type: 'MILK_COLLECTION_ESCROW_REQUESTED_SPLIT_CREATED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { collectionId, escrowId, amountCents, targetsCount:
targets.length, priceCentsPerL: price },
        ts: now
      });
```

```
      const out = { collectionId, escrowId, status:
'PAYMENT_REQUESTED', amountCents };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'milk.collections.pay', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });
      return out;
    });
  }
}
```

---

7) Webhook PAID (Asaas) → payouts por produtor

7.1 Controller

```
// src/webhooks/asaas-milk.controller.ts
import { Body, Controller, Post } from '@nestjs/common';
import { AsaasMilkPaidService } from './asaas-milk-paid.service';

@Controller('/api/webhooks/asaas/milk')
export class AsaasMilkWebhookController {
  constructor(private readonly svc: AsaasMilkPaidService) {}

  @Post()
  async handle(@Body() body: any) {
    // extrai metadata escrowId do PSP
    const escrowId = body?.payment?.metadata?.escrowId;
    const status = body?.payment?.status;
    if (!escrowId) return { ok: true };

    if (status === 'CONFIRMED' || status === 'RECEIVED') {
      return this.svc.handlePaid(escrowId, Date.now(), body);
    }
    return { ok: true };
  }
}
```

7.2 Service

```
// src/webhooks/asaas-milk-paid.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class AsaasMilkPaidService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async handlePaid(escrowId: string, paidAt: number, pspRef: any) {
    const escRef = this.db.collection('escrows_milk').doc(escrowId);

    return this.db.runTransaction(async (tx) => {
      const escSnap = await tx.get(escRef);
      if (!escSnap.exists) throw new Error('ESCROW_NOT_FOUND');
      const esc = escSnap.data()!;
      if (esc.status === 'PAID') return { ok: true, already: true };

      // marca escrow PAID
      tx.set(escRef, { status: 'PAID', paidAt, pspRef }, { merge: true
});

      // marca collection PAID
```

```
      const colRef =
this.db.collection('milk_collections').doc(esc.collectionId);
      const colSnap = await tx.get(colRef);
      if (colSnap.exists) {
        tx.set(colRef, { status: 'PAID', paidAt, escrow: { escrowId,
status: 'PAID' } }, { merge: true });
      }

      // cria payouts por produtor (stub)
      let payouts = 0;
      for (const t of esc.targets || []) {
        const payoutId = makeId('payout');

tx.create(this.db.collection('producer_payouts').doc(payoutId), {
          tenantId: esc.tenantId,
          payoutId,
          escrowId,
          collectionId: esc.collectionId,
          producerTenantId: t.producerTenantId,
          liters: t.liters,
          amountCents: t.amountCents,
          status: 'QUEUED',
          createdAt: paidAt
        });
        payouts++;
      }

      tx.create(this.db.collection('audit_events').doc(makeId('aud')),
{
        auditId: makeId('aud'),
        tenantId: esc.tenantId,
        type: 'MILK_ESCROW_PAID_PAYOUTS_QUEUED',
        meta: { escrowId, collectionId: esc.collectionId, payouts },
        ts: paidAt
      });

      return { ok: true, escrowId, payouts };
    });
  }
}
```

---

8) Lock por inadimplência da indústria (anti-calote)

8.1 Regra simples

Se a indústria não pagar um escrow em X horas:

- cria lock `NON_PAYMENT` no escopo `TENANT` para `industryTenantId`
- bloqueia novos contratos e compras no marketplace (efeitos)

Onde detectar

- Job interno tipo `/api/internal/milk/locks/non-payment-scan`
- Varre `escrows_milk` status `REQUESTED` com `createdAt < now - TTL`

8.2 Service (scanner)

```
// src/milk/non-payment-lock-scan.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class NonPaymentLockScanService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async run(ttlHours = 48) {
    const now = Date.now();
    const cutoff = now - ttlHours * 3600_000;

    const snap = await this.db.collection('escrows_milk')
      .where('status','==','REQUESTED')
      .where('createdAt','<', cutoff)
      .limit(200)
      .get();

    const created: any[] = [];
    for (const d of snap.docs) {
      const e = d.data()!;
      // cria lock para a indústria
      const lockId = makeId('lock');
      await this.db.collection('locks').doc(lockId).set({
        tenantId: e.tenantId,
        lockId,
        scope: { type: 'TENANT', tenantId: e.industryTenantId },
        reason: { code: 'NON_PAYMENT', title: 'Inadimplência',
details: `Escrow leite não pago: ${e.escrowId}` },
        effects: { blockNewContracts: true, blockSale: true,
blockInvoice: false, blockTransferAssets: false },
        links: { escrowId: e.escrowId, collectionId: e.collectionId },
        status: 'ACTIVE',
        createdAt: now,
        createdBy: { uid: 'system', role: 'SYSTEM' }
      }, { merge: false });

      await this.db.collection('audit_events').add({
        auditId: makeId('aud'),
        tenantId: e.tenantId,
        type: 'AUTO_LOCK_NON_PAYMENT_INDUSTRY_CREATED',
        meta: { lockId, industryTenantId: e.industryTenantId,
escrowId: e.escrowId },
        ts: now
      });

      created.push({ lockId, escrowId: e.escrowId, industryTenantId:
e.industryTenantId });
    }

    return { ok: true, scanned: snap.size, created };
  }
}
```

---

9) Permissions sugeridas

- `milk.tankChecks.create` (PRODUCER/TECHNICIAN/OPERATOR)
- `milk.collections.build` (TECHNICIAN/AUDITOR/MANAGER)
- `milk.collections.attachLab` (AUDITOR/TECHNICIAN/INDUSTRY se tiver)
- `milk.collections.price` (AUDITOR/MANAGER)
- `milk.collections.pay` (INDUSTRY/MANAGER)
- internal scans via secret header (igual ao cancel-expired)

---

10) O que faltaria para ficar "100% operacional"

1. **Tela Mobile**: ticket diário (modo acessível, voz/foto)
2. OCR/voz → preenchimento automático (você já pediu isso; encaixa aqui)
3. Integração real do Asaas:
   - criar cobrança com metadata `{ escrowId }`
   - payout para produtores (split nativo ou transfers)
4. Módulo de "qualidade por produtor" (se cada produtor tem amostra separada)

Se você quiser o próximo bloco, eu fecho o que você pediu sobre **voz/foto anotação em papel**:

- endpoint `POST /api/intake/photo-to-movement` (OCR/IA)
- grava evidência e já cria movimentação (estoque/insumo/leite) com validação.

Perfeito — abaixo vai o módulo **"lançamento por foto (papel) e por voz"** do jeito certo para o seu Ciclo+:

- **Não substitui documento fiscal** (NF/CT-e etc.)
- Serve para **operacionalizar**: retirar sal mineral, lançar pesagem, registrar entrega de leite, apontar uso de insumos, etc.
- Gera **evidência**, extrai dados (OCR/voz), cria **rascunho**, passa por **validação + RulesEngine + máquina de estados**, e só então vira movimentação oficial.

Vou te entregar:

1. Modelo de dados
2. Pipeline de intake (Foto/Voz → Extração → Rascunho → Aprovação → Execução)
3. Endpoints NestJS completos (com Firestore tx + idempotência + auditoria)
4. Regras (confiabilidade, bloqueios, evidência Tipo A/B)
5. Integração recomendada para OCR/ASR (sem travar o seu backend)

---

1) Modelo de dados

1.1 Evidência (já existe no seu conceito)

```
evidences/{evidenceId}

{
  "tenantId":"TENANT_FARM",
  "evidenceId":"evg_001",
  "type":"PHOTO|AUDIO|VIDEO|DOC",
  "storage":{
    "provider":"firebase_storage",
    "path":"evidence/TENANT_FARM/evg_001.jpg"
  },
  "meta":{
    "capturedAt":1730000000000,
    "gps":{"lat":-15.7,"lng":-56.1},
    "deviceAttest":"...",
    "hash":"sha256:..."
  },
  "status":"ACTIVE",
  "createdAt":1730000000000
}
```

1.2 Intake (o “caso” criado a partir da foto/voz)

```
intakes/{intakeId}

{
  "tenantId":"TENANT_FARM",
  "intakeId":"in_001",
  "channel":"PHOTO|VOICE",

"purpose":"STOCK_WITHDRAW|WEIGHING|MILK_TICKET|FIELD_OP|NOTE_REFERENCE
",
  "propertyId":"prop_123",
  "evidenceIds":["evg_001","evg_002"],
  "rawText":"retirada sal mineral 3 sacos ...",          // do OCR/ASR
  "structured":{
    "itemId":"item_salt_01",
    "qtyBase": 75,
    "unitBase":"KG",
    "occurredAt":1730000000000,
    "operatorName":"João",
    "noteRef":"caderno_02_p12"
  },
  "confidence":{
    "overall":0.86,
    "fields":{"qtyBase":0.92,"itemId":0.70}
  },

"state":"RECEIVED|EXTRACTED|DRAFT|NEEDS_REVIEW|APPROVED|REJECTED|EXECU
TED",
  "createdBy":{"uid":"u1","role":"OPERATOR"},
  "createdAt":1730000000000,
  "approvedBy":null,
  "approvedAt":null,
  "result":{
    "type":"STOCK_MOVEMENT",
```

```
    "refId":"mv_001"
  }
}
```

1.3 “Draft” de movimentação (rascunho revisável)

```
draft_movements/{draftId}

{
  "tenantId":"TENANT_FARM",
  "draftId":"dmv_001",
  "intakeId":"in_001",
  "type":"WITHDRAW|DEPOSIT|ADJUST",
  "propertyId":"prop_123",
  "itemId":"item_salt_01",
  "qtyBase":75,
  "unitBase":"KG",
  "occurredAt":1730000000000,
  "evidenceIds":["evg_001"],
  "status":"DRAFT|APPROVED|REJECTED"
}
```

---

2) Pipeline (Foto/Voz)

2.1 Foto (papel)

1. App tira foto do caderno/planilha → envia para Storage → cria `evidence`
2. Backend cria `intake` com `state=RECEIVED`
3. Um worker de extração (OCR) lê a imagem e grava `rawText` + `structured` + `confidence`
4. Backend cria `draft_movements`
5. Se confiança alta + regras ok → aprova automático (ou manda para revisão)
6. Ao aprovar → executa: cria `stock_move` append-only, ajusta saldo, auditoria

2.2 Voz

1. App grava áudio (comando) + (opcional) foto evidência
2. ASR transcreve (no device ou cloud)
3. Mesmo pipeline: `intake → draft → aprovação → execução`

✅ Isso resolve “sem digitar”, mantendo lastro e controle.

---

3) Integração OCR/ASR (recomendação prática)

Para “funcionar perfeitamente” você tem 3 opções:

Opção A (mais simples e barata)

- **OCR/voz no celular** (on-device) e manda só o texto + confidences pro backend.
- Backend não depende de serviço externo.

Opção B (robusta para produção)

- Cloud Vision OCR + Speech-to-Text (ou equivalente) num **worker**.
- Backend só "orquestra" e registra.

Opção C (híbrida)

- App tenta on-device. Se confiança baixa → manda imagem/áudio para OCR cloud.

Eu vou te entregar o backend preparado para A/B/C via "Extractor Provider" (pluggable).

---

4) Endpoints NestJS completos

4.1 `POST /api/intake` — cria intake (foto/voz)

Entrada: purpose + propertyId + evidenceIds + (opcional) rawText se o app já transcreveu.

```
// src/intake/dto/intake-create.dto.ts
import { IsArray, IsIn, IsNotEmpty, IsOptional, IsString } from
'class-validator';

export class IntakeCreateDto {
  @IsString() @IsNotEmpty() clientRequestId: string;

  @IsIn(['PHOTO','VOICE'])
  channel: 'PHOTO'|'VOICE';


@IsIn(['STOCK_WITHDRAW','WEIGHING','MILK_TICKET','FIELD_OP','NOTE_REFE
RENCE'])
  purpose: string;

  @IsString() @IsNotEmpty()
  propertyId: string;

  @IsArray()
  evidenceIds: string[];

  @IsOptional() @IsString()
  rawText?: string; // se app já fez OCR/voz
}
// src/intake/intake.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class IntakeService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}
```

```
  async create(actor: any, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'intake.create');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
      if (idem.exists) return idem.data()!.result;

      if (!dto.evidenceIds?.length) throw new
Error('EVIDENCE_REQUIRED');

      // (opcional) validar que property existe e pertence ao tenant
      const propRef =
this.db.collection('properties').doc(dto.propertyId);
      const propSnap = await tx.get(propRef);
      if (!propSnap.exists) throw new Error('PROPERTY_NOT_FOUND');
      if (propSnap.data()!.tenantId !== tenantId) throw new
Error('FORBIDDEN_PROPERTY');

      const intakeId = makeId('in');
      tx.create(this.db.collection('intakes').doc(intakeId), {
        tenantId,
        intakeId,
        channel: dto.channel,
        purpose: dto.purpose,
        propertyId: dto.propertyId,
        evidenceIds: dto.evidenceIds,
        rawText: dto.rawText ?? '',
        structured: null,
        confidence: null,
        state: dto.rawText ? 'EXTRACTED' : 'RECEIVED',
        createdBy: { uid: actor.uid, role: actor.role },
        createdAt: now
      });

      tx.create(this.db.collection('audit_events').doc(makeId('aud')),
{
        auditId: makeId('aud'),
        tenantId,
        type: 'INTAKE_CREATED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { intakeId, channel: dto.channel, purpose: dto.purpose,
propertyId: dto.propertyId, evidenceIdsCount: dto.evidenceIds.length
},
        ts: now
      });

      const out = { intakeId, state: dto.rawText ? 'EXTRACTED' :
'RECEIVED' };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'intake.create', clientRequestId: dto.clientRequestId, createdAt: now,
result: out });
      return out;
    });
  }
}
```

```
// src/intake/intake.controller.ts
import { Body, Controller, Post, UseGuards } from '@nestjs/common';
import { FirebaseAuthGuard } from '../common/guards/firebase-
auth.guard';
import { TenantGuard } from '../common/guards/tenant.guard';
import { RolesGuard } from '../common/guards/roles.guard';
import { Roles } from '../common/decorators/roles.decorator';
import { Permissions } from
'../common/decorators/permissions.decorator';
import { Actor } from '../common/decorators/actor.decorator';
import { IntakeService } from './intake.service';

@Controller('/api/intake')
@UseGuards(FirebaseAuthGuard, TenantGuard, RolesGuard)
export class IntakeController {
  constructor(private readonly svc: IntakeService) {}

  @Post()
  @Roles('ADMIN','MANAGER','OPERATOR','PRODUCER','TECHNICIAN')
  @Permissions('intake.create')
  create(@Actor() actor: any, @Body() dto: any) {
    return this.svc.create(actor, dto);
  }
}
```

4.2 `POST /api/intake/:id/extract` — grava extração (quando worker/app já extraiu)

Isso permite A/B/C: o app manda texto; ou um worker OCR manda resultado.

```
// src/intake/dto/intake-extract.dto.ts
import { IsNotEmpty, IsObject, IsOptional, IsString } from 'class-
validator';

export class IntakeExtractDto {
  @IsString() @IsNotEmpty() clientRequestId: string;

  @IsString() @IsNotEmpty()
  rawText: string;

  @IsObject()
  structured: any;

  @IsObject()
  confidence: any; // { overall, fields: {...} }

  @IsOptional()
  extractorMeta?: any; // provider, model, duration, etc.
}
// src/intake/intake-extract.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class IntakeExtractService {
```

```
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async attachExtract(actor: any, intakeId: string, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'intake.extract.attach');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
      if (idem.exists) return idem.data()!.result;

      const ref = this.db.collection('intakes').doc(intakeId);
      const snap = await tx.get(ref);
      if (!snap.exists) throw new Error('INTAKE_NOT_FOUND');
      const it = snap.data()!;
      if (it.tenantId !== tenantId) throw new Error('FORBIDDEN');
      if (!['RECEIVED','EXTRACTED'].includes(it.state)) throw new
Error('INTAKE_STATE_INVALID');

      tx.set(ref, {
        rawText: dto.rawText,
        structured: dto.structured,
        confidence: dto.confidence,
        extractorMeta: dto.extractorMeta ?? null,
        state: 'EXTRACTED'
      }, { merge: true });

      tx.create(this.db.collection('audit_events').doc(makeId('aud')),
{
        auditId: makeId('aud'),
        tenantId,
        type: 'INTAKE_EXTRACT_ATTACHED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { intakeId, overall: dto.confidence?.overall ?? null },
        ts: now
      });

      const out = { intakeId, state: 'EXTRACTED' };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'intake.extract.attach', clientRequestId: dto.clientRequestId,
createdAt: now, result: out });
      return out;
    });
  }
}
```

4.3 `POST /api/intake/:id/to-draft` — cria draft de movimentação (com RulesEngine)

Aqui acontece o "entendimento": transformar `structured` em `draft_movements`.

Regras mínimas:

- itemId obrigatório (ou catálogo fuzzy → precisa revisão)

- qtyBase > 0
- occurredAt definido (senão usa now e marca “NEEDS_REVIEW”)
- Locks (CAR/IE/property) podem impedir execução final, mas **draft pode existir**.

```
// src/intake/intake-to-draft.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';

@Injectable()
export class IntakeToDraftService {
  constructor(@Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore) {}

  async toDraft(actor: any, intakeId: string, clientRequestId: string)
{
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, clientRequestId,
'intake.toDraft');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
      if (idem.exists) return idem.data()!.result;

      const itRef = this.db.collection('intakes').doc(intakeId);
      const itSnap = await tx.get(itRef);
      if (!itSnap.exists) throw new Error('INTAKE_NOT_FOUND');
      const it = itSnap.data()!;
      if (it.tenantId !== tenantId) throw new Error('FORBIDDEN');
      if (it.state !== 'EXTRACTED') throw new
Error('INTAKE_NOT_EXTRACTED');

      const s = it.structured || {};
      const conf = it.confidence || { overall: 0, fields: {} };

      // interpreta por propósito
      if (it.purpose !== 'STOCK_WITHDRAW') {
        // você pode expandir aqui para WEIGHING, MILK_TICKET,
FIELD_OP etc.
        throw new Error('PURPOSE_NOT_IMPLEMENTED_YET');
      }

      const itemId = s.itemId;
      const qtyBase = Number(s.qtyBase || 0);
      const unitBase = (s.unitBase || 'UN').toUpperCase();
      const occurredAt = Number(s.occurredAt || now);

      if (!itemId || !(qtyBase > 0)) {
        // cria intake como NEEDS_REVIEW
        tx.set(itRef, { state: 'NEEDS_REVIEW' }, { merge: true });
        const out = { intakeId, state: 'NEEDS_REVIEW', reason:
'MISSING_ITEM_OR_QTY' };
        tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'intake.toDraft', clientRequestId, createdAt: now, result: out });
        return out;
```

```
      }

      const draftId = makeId('dmv');
      tx.create(this.db.collection('draft_movements').doc(draftId), {
        tenantId,
        draftId,
        intakeId,
        type: 'WITHDRAW',
        propertyId: it.propertyId,
        itemId,
        qtyBase,
        unitBase,
        occurredAt,
        evidenceIds: it.evidenceIds || [],
        status: conf.overall >= 0.85 ? 'APPROVED' : 'DRAFT'
      });

      tx.set(itRef, {
        state: conf.overall >= 0.85 ? 'APPROVED' : 'DRAFT',
        result: { type: 'DRAFT_MOVEMENT', refId: draftId }
      }, { merge: true });

      tx.create(this.db.collection('audit_events').doc(makeId('aud')),
{
        auditId: makeId('aud'),
        tenantId,
        type: 'INTAKE_TO_DRAFT_MOVEMENT_CREATED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { intakeId, draftId, itemId, qtyBase, unitBase,
autoApproved: conf.overall >= 0.85 },
        ts: now
      });

      const out = { intakeId, draftId, state: conf.overall >= 0.85 ?
'APPROVED' : 'DRAFT' };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'intake.toDraft', clientRequestId, createdAt: now, result: out });
      return out;
    });
  }
}
```

4.4 `POST /api/draft-movements/:id/approve` — aprovação humana

```
// src/movements/dto/draft-approve.dto.ts
import { IsNotEmpty, IsOptional, IsString } from 'class-validator';

export class DraftApproveDto {
  @IsString() @IsNotEmpty() clientRequestId: string;
  @IsOptional() @IsString() notes?: string;
}
```

4.5 `POST /api/draft-movements/:id/execute` — executa movimentação oficial (append-only)

Aqui entra o “coração”: valida locks + saldo + regras, e cria `stock_moves` + ajusta `stock_balance`.

Você já tinha a ideia de:

- `inventory/{itemId}/moves` ou `stock_moves` global
- registrar balanceBefore/balanceAfter
- evidência anexada

Exemplo robusto (com saldo simples por item+property):

```
// src/movements/draft-execute.service.ts
import { Inject, Injectable } from '@nestjs/common';
import { FIRESTORE } from '../common/firestore.provider';
import { idemKey } from '../common/idempotency/idempotency.util';
import { makeId } from '../common/utils/id.util';
import { LockCheckService } from '../locks/lock-check.service';

@Injectable()
export class DraftExecuteService {
  constructor(
    @Inject(FIRESTORE) private readonly db:
FirebaseFirestore.Firestore,
    private readonly lockCheck: LockCheckService
  ) {}

  private balanceDocId(tenantId: string, propertyId: string, itemId:
string) {
    return `bal_${tenantId}_${propertyId}_${itemId}`;
  }

  async execute(actor: any, draftId: string, dto: any) {
    const now = Date.now();
    const tenantId = actor.tenantId;

    const key = idemKey(tenantId, actor.uid, dto.clientRequestId,
'draft.execute');
    const idemRef = this.db.collection('idempotency').doc(key);

    return this.db.runTransaction(async (tx) => {
      const idem = await tx.get(idemRef);
      if (idem.exists) return idem.data()!.result;

      const dRef = this.db.collection('draft_movements').doc(draftId);
      const dSnap = await tx.get(dRef);
      if (!dSnap.exists) throw new Error('DRAFT_NOT_FOUND');
      const d = dSnap.data()!;
      if (d.tenantId !== tenantId) throw new Error('FORBIDDEN');
      if (!['APPROVED','DRAFT'].includes(d.status)) throw new
Error('DRAFT_NOT_EXECUTABLE');

      // ✅ locks: se travar venda/nota, aqui trava “movimentação
crítica” se você quiser.
      // Para estoque interno da fazenda, você pediu que isso funcione
— então por padrão:
      // - NÃO bloqueia withdraw interno por NON_PAYMENT
      // - MAS bloqueia se houver lock ambiental/contratual
configurado para operações internas também (opcional).
```

```
      const { locks } = await
this.lockCheck.getActiveLocksForProperty(tenantId, d.propertyId);
      // Se você quiser bloquear retirada de insumo quando houver
ENV_LIMIT:
      // this.lockCheck.enforceEffects(locks, ['blockSale']); // ou um
novo effect: blockInternalOps

      // saldo
      const balId = this.balanceDocId(tenantId, d.propertyId,
d.itemId);
      const balRef = this.db.collection('stock_balance').doc(balId);
      const balSnap = await tx.get(balRef);
      const bal = balSnap.exists ? balSnap.data()! : { qtyBase: 0 };

      const before = Number(bal.qtyBase || 0);
      const qty = Number(d.qtyBase || 0);

      if (d.type === 'WITHDRAW') {
        if (before < qty) throw new
Error('INSUFFICIENT_STOCK_BALANCE');
      }

      const after = d.type === 'WITHDRAW' ? before - qty : before +
qty;

      // append-only move
      const mvId = makeId('mv');
      tx.create(this.db.collection('stock_moves').doc(mvId), {
        tenantId,
        moveId: mvId,
        draftId,
        intakeId: d.intakeId ?? null,
        type: d.type,
        propertyId: d.propertyId,
        itemId: d.itemId,
        qtyBase: qty,
        unitBase: d.unitBase,
        balanceBefore: before,
        balanceAfter: after,
        evidenceIds: d.evidenceIds || [],
        performedBy: { uid: actor.uid, role: actor.role },
        occurredAt: d.occurredAt ?? now,
        createdAt: now
      });

      // update balance
      tx.set(balRef, {
        tenantId,
        propertyId: d.propertyId,
        itemId: d.itemId,
        qtyBase: after,
        updatedAt: now
      }, { merge: true });

      // mark draft executed
      tx.set(dRef, { status: 'EXECUTED', executedAt: now, executedBy:
{ uid: actor.uid, role: actor.role }, resultMoveId: mvId }, { merge:
true });

      // audit
```

```
      tx.create(this.db.collection('audit_events').doc(makeId('aud')),
{
        auditId: makeId('aud'),
        tenantId,
        type: 'DRAFT_EXECUTED_STOCK_MOVE_CREATED',
        actor: { uid: actor.uid, role: actor.role },
        meta: { draftId, moveId: mvId, itemId: d.itemId, qtyBase: qty,
balanceBefore: before, balanceAfter: after },
        ts: now
      });

      const out = { draftId, moveId: mvId, status: 'EXECUTED',
balanceAfter: after };
      tx.create(idemRef, { tenantId, uid: actor.uid, scope:
'draft.execute', clientRequestId: dto.clientRequestId, createdAt: now,
result: out });
      return out;
    });
  }
}
```

---

5) Regras importantes (para ficar "à prova")

5.1 Evidência obrigatória por risco

- Retirada simples (sal mineral): evidência Tipo A ou foto do caderno (Tipo B "leve")
- Operação crítica (venda, transferência, nota): evidência Tipo B completa + validação técnico/auditor

5.2 Confiança baixa vira "NEEDS_REVIEW"

- `overall < 0.85` → rascunho exige revisão
- `itemId` não identificado → revisão

5.3 Idempotência em tudo

- Evita duplicar lançamento se o operador enviar 2x.

5.4 Auditoria append-only

- Qualquer execução gera evento e hash (no seu módulo de auditoria imutável).

---

6) Como encaixa com "estoque do fornecedor em tempo real" e marketplace

O mesmo "intake" serve para:

- operador na loja lançar entrada/saída por foto
- fornecedor lançar saldo por feed/API
- e o marketplace só mostra estoque "reservável" após escrow

---

7) Permissions sugeridas

- `intake.create`
- `intake.extract.attach` (app/worker)
- `intake.toDraft`
- `draft.approve`
- `draft.execute`

Se você quiser, eu fecho o último "pedaço" para ficar redondo no campo:

- **modo acessível** (UI com letras grandes + voz guiada + botões enormes),
- e o "fallback" quando OCR falhar: app pergunta 3 coisas por voz ("qual item?", "quantidade?", "data/hora?") e completa o draft automaticamente com evidência da foto.

Perfeito — vou fechar agora o **Modo Acessível (mobile)** + **Voz guiada (fallback quando OCR falhar)** + **fluxo de 3 perguntas** e como isso integra com o backend (`intake → draft → approve/execute`) sem digitação.

Vou te entregar:

1. UX (telas e comportamento)
2. Contratos de API (requests/responses)
3. Lógica no app (Expo + RN) com Voice + Photo + offline queue
4. Regras de acessibilidade (WCAG pragmático)
5. "Assistente de campo" (máquina de estados no app)

---

1) UX: Modo Acessível (para leitura difícil)

1.1 Princípios

- **3 ações principais** na home:
    1. **Retirar/Usar Insumo**
    2. **Registrar Leite / Pesagem**
    3. **Comprar/Vender (Marketplace)**
- Texto grande, pouco conteúdo por tela
- Ícones + cor + voz confirmando
- Sempre ter: **"Falar"**, **"Foto"**, **"Confirmar"**

1.2 Telas mínimas (mobile)

Tela A — Home (Acessível)

- Botão gigante: 📦 **ESTOQUE (RETIRAR / ENTRAR)**

- Botão gigante: 🥛 **LEITE (TICKET / ENTREGA)**
- Botão gigante: 🛒 **MARKETPLACE (COMPRAR / VENDER)**
- Botão gigante: 🎙️ **REGISTRAR POR VOZ**
- Botão gigante: 📷 **REGISTRAR POR FOTO**

Tela B — Registro por Foto (papel)

- Botão gigante: **TIRAR FOTO**
- Mostra miniatura
- Botão: **ENVIAR**
- Resultado:
  - "Entendi assim: ITEM = … QTD = … DATA = …"
  - **CONFIRMAR** ou **CORRIGIR POR VOZ**

Tela C — Voz guiada (3 perguntas)

- 1/3: "Qual item?" (fala ou escolhe lista)
- 2/3: "Qual quantidade?" (fala número)
- 3/3: "Foi hoje?" (sim/não; se não, pergunta data)
- Final: "Confirmar?" → cria draft e executa (se permitido)

---

2) Contratos de API (sem ambiguidade)

2.1 Criar intake

```
POST /api/intake

{
  "clientRequestId":"uuid",
  "channel":"PHOTO",
  "purpose":"STOCK_WITHDRAW",
  "propertyId":"prop_123",
  "evidenceIds":["evg_001"],
  "rawText":"opcional"
}
```

Resp:

```
{ "intakeId":"in_001", "state":"RECEIVED" }
```

2.2 Anexar extração (OCR/voz)

```
POST /api/intake/:id/extract

{
  "clientRequestId":"uuid",
  "rawText":"retirada sal mineral 3 sacos",

"structured":{"itemId":"item_salt","qtyBase":75,"unitBase":"KG","occur
redAt":173...},
  "confidence":{"overall":0.72,"fields":{"itemId":0.6,"qtyBase":0.95}}
```

```
}
```

2.3 Criar draft

```
POST /api/intake/:id/to-draft
```

```
{ "clientRequestId":"uuid" }
```

Resp:

```
{ "intakeId":"in_001", "draftId":"dmv_001", "state":"DRAFT" }
```

2.4 Executar draft

```
POST /api/draft-movements/:id/execute
```

```
{ "clientRequestId":"uuid" }
```

Resp:

```
{ "draftId":"dmv_001", "moveId":"mv_001", "status":"EXECUTED" }
```

---

3) App (Expo + React Native): fluxo com Foto + Voz + Offline

3.1 Dependências (sugestão)

- `expo-av` (áudio)
- `expo-speech` (falar/confirmar)
- `expo-image-picker` (foto)
- `expo-location` (GPS)
- `@react-native-voice/voice` (ASR) **ou** ASR do próprio teclado/OS
- `react-query` + persistência (cache offline)
- fila local (AsyncStorage) para "pending actions"

Para simplificar, você pode começar sem `Voice` e usar "ditado do teclado", mas deixo o modelo completo.

3.2 Máquina de estados no app (Assistente de Campo)

Estados:

- `IDLE`
- `CAPTURE_PHOTO`
- `UPLOAD_EVIDENCE`
- `OCR_PENDING`
- `VOICE_FALLBACK_Q1_ITEM`
- `VOICE_FALLBACK_Q2_QTY`
- `VOICE_FALLBACK_Q3_DATE`
- `CONFIRM_SUMMARY`
- `SUBMIT_DRAFT`

- `EXECUTE_OR_REVIEW`
- `DONE` / `ERROR`

O app sempre consegue terminar, mesmo sem OCR.

3.3 Estrutura de "Action Queue" (offline)

`local_queue[]` item:

```
{
  "id":"q_001",
  "type":"INTAKE_PHOTO_STOCK_WITHDRAW",
  "payload":{...},
  "status":"PENDING|SENT|DONE|FAILED",
  "createdAt":173...
}
```

Regra:

- Se sem internet: salva na fila e mostra "✅ Salvo. Enviar depois."
- Quando volta internet: worker no app reenvia.

---

4) Fallback quando OCR falhar (3 perguntas)

4.1 Quando considerar "falhou"

- confidence overall < 0.85 **ou**
- faltou `itemId` **ou**
- faltou `qtyBase`

Aí o app ativa **"Corrigir por voz"** e pergunta só o que falta.

---

5) Implementação (pseudo-código RN) — fluxo resumido

5.1 Captura + envio

```
async function createStockWithdrawByPhoto(propertyId: string) {
  // 1) tirar foto
  const photo = await pickCameraPhoto();

  // 2) GPS
  const gps = await getGps();

  // 3) upload evidence (storage) e criar evidenceId
  const evidenceId = await uploadEvidence(photo.uri, gps);

  // 4) cria intake
```

```
  const intake = await api.post('/api/intake', {
    clientRequestId: uuid(),
    channel: 'PHOTO',
    purpose: 'STOCK_WITHDRAW',
    propertyId,
    evidenceIds: [evidenceId]
  });

  speak(`Foto enviada. Vou tentar ler sua anotação.`);
  return intake.intakeId;
}
```

5.2 OCR/Extração

- Se OCR rodar no app → chama `/extract` direto.
- Se OCR rodar em worker → o app consulta status e recebe update.

5.3 Se OCR falhar → 3 perguntas por voz

```
async function voiceFallback(intakeId: string, missing: {
item?:boolean; qty?:boolean; date?:boolean }) {
  const item = missing.item ? await askVoice("Qual item foi retirado?
Exemplo: sal mineral.") : null;
  const qty  = missing.qty  ? await askVoice("Qual quantidade?
Exemplo: setenta e cinco quilos.") : null;
  const date = missing.date ? await askVoice("Foi hoje? Diga sim ou
não.") : "today";

  const structured = normalizeToStructured({ item, qty, date });
  await api.post(`/api/intake/${intakeId}/extract`, {
    clientRequestId: uuid(),
    rawText: `voz: ${item} ${qty} ${date}`,
    structured,
    confidence: { overall: 0.99, fields: { itemId: 0.95, qtyBase: 0.95
} },
    extractorMeta: { provider: 'voice_fallback' }
  });

  // cria draft
  const draft = await api.post(`/api/intake/${intakeId}/to-draft`, {
clientRequestId: uuid() });

  // tenta executar direto (se permitido)
  try {
    const exec = await api.post(`/api/draft-
movements/${draft.draftId}/execute`, { clientRequestId: uuid() });
    speak("Registrado com sucesso.");
    return exec;
  } catch {
    speak("Salvei para revisão.");
    return draft;
  }
}
```

---

6) Normalização (item por catálogo + sinônimos)

Para “qual item”, você pode:

- sugerir lista de top 10 itens mais usados
- aceitar voz livre e aplicar "match" em `catalog` com sinônimos (ex.: "sal mineral", "núcleo", "ração", "ureia")

No backend: se não bater com confiança, `NEEDS_REVIEW`.

---

7) Regras de acessibilidade (práticas)

- Fonte mínima 18–22
- Botões 56px+ (touch)
- Contraste alto (texto preto fundo claro)
- Uma ação por tela
- Confirmação por voz: "Entendi X. Confirmar?"
- Vibração curta em sucesso/erro (opcional)

Próximo passo (se você disser "sim")

Eu gero:

- **wireframes das telas** (Home, Foto, Voz 3 passos, Confirmar, Offline)
- e o **código React Native (Expo)** dos componentes principais (`AccessibleHome`, `CapturePhotoFlow`, `VoiceWizard`, `OfflineQueueWorker`) pronto para o seu MVP.

Perfeito — vou fechar agora o **Modo Acessível (mobile)** + **Voz guiada (fallback quando OCR falhar)** + **fluxo de 3 perguntas** e como isso integra com o backend (`intake → draft → approve/execute`) sem digitação.

Vou te entregar:

1. UX (telas e comportamento)
2. Contratos de API (requests/responses)
3. Lógica no app (Expo + RN) com Voice + Photo + offline queue
4. Regras de acessibilidade (WCAG pragmático)
5. "Assistente de campo" (máquina de estados no app)

---

1) UX: Modo Acessível (para leitura difícil)

1.1 Princípios

- **3 ações principais** na home:
    1. **Retirar/Usar Insumo**
    2. **Registrar Leite / Pesagem**
    3. **Comprar/Vender (Marketplace)**

- Texto grande, pouco conteúdo por tela
- Ícones + cor + voz confirmando
- Sempre ter: **“Falar”**, **“Foto”**, **“Confirmar”**

1.2 Telas mínimas (mobile)

Tela A — Home (Acessível)

- Botão gigante: 📦 **ESTOQUE (RETIRAR / ENTRAR)**
- Botão gigante: 🥛 **LEITE (TICKET / ENTREGA)**
- Botão gigante: 🛒 **MARKETPLACE (COMPRAR / VENDER)**
- Botão gigante: 🎙️ **REGISTRAR POR VOZ**
- Botão gigante: 📷 **REGISTRAR POR FOTO**

Tela B — Registro por Foto (papel)

- Botão gigante: **TIRAR FOTO**
- Mostra miniatura
- Botão: **ENVIAR**
- Resultado:
  - “Entendi assim: ITEM = … QTD = … DATA = …”
  - **CONFIRMAR** ou **CORRIGIR POR VOZ**

Tela C — Voz guiada (3 perguntas)

- 1/3: “Qual item?” (fala ou escolhe lista)
- 2/3: “Qual quantidade?” (fala número)
- 3/3: “Foi hoje?” (sim/não; se não, pergunta data)
- Final: “Confirmar?” → cria draft e executa (se permitido)

---

2) Contratos de API (sem ambiguidade)

2.1 Criar intake

```
POST /api/intake

{
  "clientRequestId":"uuid",
  "channel":"PHOTO",
  "purpose":"STOCK_WITHDRAW",
  "propertyId":"prop_123",
  "evidenceIds":["evg_001"],
  "rawText":"opcional"
}
```

Resp:

```
{ "intakeId":"in_001", "state":"RECEIVED" }
```

2.2 Anexar extração (OCR/voz)

```
POST /api/intake/:id/extract
```

```
{
  "clientRequestId":"uuid",
  "rawText":"retirada sal mineral 3 sacos",

"structured":{"itemId":"item_salt","qtyBase":75,"unitBase":"KG","occur
redAt":173...},
  "confidence":{"overall":0.72,"fields":{"itemId":0.6,"qtyBase":0.95}}
}
```

2.3 Criar draft

```
POST /api/intake/:id/to-draft
```

```
{ "clientRequestId":"uuid" }
```

Resp:

```
{ "intakeId":"in_001", "draftId":"dmv_001", "state":"DRAFT" }
```

2.4 Executar draft

```
POST /api/draft-movements/:id/execute
```

```
{ "clientRequestId":"uuid" }
```

Resp:

```
{ "draftId":"dmv_001", "moveId":"mv_001", "status":"EXECUTED" }
```

---

3) App (Expo + React Native): fluxo com Foto + Voz + Offline

3.1 Dependências (sugestão)

- `expo-av` (áudio)
- `expo-speech` (falar/confirmar)
- `expo-image-picker` (foto)
- `expo-location` (GPS)
- `@react-native-voice/voice` (ASR) **ou** ASR do próprio teclado/OS
- `react-query` + persistência (cache offline)
- fila local (AsyncStorage) para “pending actions”

Para simplificar, você pode começar sem `Voice` e usar “ditado do teclado”, mas deixo o modelo completo.

3.2 Máquina de estados no app (Assistente de Campo)

Estados:

- `IDLE`
- `CAPTURE_PHOTO`
- `UPLOAD_EVIDENCE`
- `OCR_PENDING`
- `VOICE_FALLBACK_Q1_ITEM`
- `VOICE_FALLBACK_Q2_QTY`
- `VOICE_FALLBACK_Q3_DATE`
- `CONFIRM_SUMMARY`
- `SUBMIT_DRAFT`
- `EXECUTE_OR_REVIEW`
- `DONE` / `ERROR`

O app sempre consegue terminar, mesmo sem OCR.

3.3 Estrutura de "Action Queue" (offline)

`local_queue[]` item:

```
{
  "id":"q_001",
  "type":"INTAKE_PHOTO_STOCK_WITHDRAW",
  "payload":{...},
  "status":"PENDING|SENT|DONE|FAILED",
  "createdAt":173...
}
```

Regra:

- Se sem internet: salva na fila e mostra "✅ Salvo. Enviar depois."
- Quando volta internet: worker no app reenvia.

---

4) Fallback quando OCR falhar (3 perguntas)

4.1 Quando considerar "falhou"

- confidence overall < 0.85 **ou**
- faltou `itemId` **ou**
- faltou `qtyBase`

Aí o app ativa **"Corrigir por voz"** e pergunta só o que falta.

---

5) Implementação (pseudo-código RN) — fluxo resumido

5.1 Captura + envio

```
async function createStockWithdrawByPhoto(propertyId: string) {
  // 1) tirar foto
  const photo = await pickCameraPhoto();

  // 2) GPS
  const gps = await getGps();

  // 3) upload evidence (storage) e criar evidenceId
  const evidenceId = await uploadEvidence(photo.uri, gps);

  // 4) cria intake
  const intake = await api.post('/api/intake', {
    clientRequestId: uuid(),
    channel: 'PHOTO',
    purpose: 'STOCK_WITHDRAW',
    propertyId,
    evidenceIds: [evidenceId]
  });

  speak(`Foto enviada. Vou tentar ler sua anotação.`);
  return intake.intakeId;
}
```

5.2 OCR/Extração

- Se OCR rodar no app → chama `/extract` direto.
- Se OCR rodar em worker → o app consulta status e recebe update.

5.3 Se OCR falhar → 3 perguntas por voz

```
async function voiceFallback(intakeId: string, missing: {
item?:boolean; qty?:boolean; date?:boolean }) {
  const item = missing.item ? await askVoice("Qual item foi retirado?
Exemplo: sal mineral.") : null;
  const qty  = missing.qty  ? await askVoice("Qual quantidade?
Exemplo: setenta e cinco quilos.") : null;
  const date = missing.date ? await askVoice("Foi hoje? Diga sim ou
não.") : "today";

  const structured = normalizeToStructured({ item, qty, date });
  await api.post(`/api/intake/${intakeId}/extract`, {
    clientRequestId: uuid(),
    rawText: `voz: ${item} ${qty} ${date}`,
    structured,
    confidence: { overall: 0.99, fields: { itemId: 0.95, qtyBase: 0.95
} },
    extractorMeta: { provider: 'voice_fallback' }
  });

  // cria draft
  const draft = await api.post(`/api/intake/${intakeId}/to-draft`, {
clientRequestId: uuid() });

  // tenta executar direto (se permitido)
  try {
    const exec = await api.post(`/api/draft-
movements/${draft.draftId}/execute`, { clientRequestId: uuid() });
    speak("Registrado com sucesso.");
    return exec;
```

```
  } catch {
    speak("Salvei para revisão.");
    return draft;
  }
}
```

---

6) Normalização (item por catálogo + sinônimos)

Para “qual item”, você pode:

- sugerir lista de top 10 itens mais usados
- aceitar voz livre e aplicar “match” em `catalog` com sinônimos (ex.: “sal mineral”, “núcleo”, “ração”, “ureia”)

No backend: se não bater com confiança, `NEEDS_REVIEW`.

---

7) Regras de acessibilidade (práticas)

- Fonte mínima 18–22
- Botões 56px+ (touch)
- Contraste alto (texto preto fundo claro)
- Uma ação por tela
- Confirmação por voz: “Entendi X. Confirmar?”
- Vibração curta em sucesso/erro (opcional)

Próximo passo (se você disser “sim”)

Eu gero:

- **wireframes das telas** (Home, Foto, Voz 3 passos, Confirmar, Offline)
- e o **código React Native (Expo)** dos componentes principais (`AccessibleHome`, `CapturePhotoFlow`, `VoiceWizard`, `OfflineQueueWorker`) pronto para o seu MVP.

Abaixo vai uma análise **do sistema inteiro** (central + módulos), com foco em **não duplicar funções**, definir uma **lógica única de operação** e listar **o que falta** para rodar “100%” em produção.

---

1) Central do Ciclo+ (núcleo único, tudo passa por aqui)

Para evitar duplicação, **nenhum módulo implementa segurança, regras, auditoria, evidência ou escrow por conta própria**. Tudo é serviço central.

1.1 Serviços centrais obrigatórios

1. **Identity Service (ID Único)**
   - Gera e valida identidade de: produto, ativo, contrato, projeto, lote, pool, evento, evidência.
2. **Tenant Isolation Service**
   - Aplica `tenantId` e `orgId` em tudo e valida no backend + Firestore Rules.
3. **Auth + RBAC + Permissions**
   - Roles (ADMIN, MANAGER, OPERATOR, PRODUCER, TECHNICIAN, SUPPLIER, AUDITOR, LOGISTICS, INVESTOR)
   - Permission matrix única.
4. **RulesEngine (pré-execução)**
   - Regras ambientais, contratuais, limites, inadimplência, custódia, estoque, geofence.
5. **State Machine**
   - Estados formais para: solicitações, contratos, entregas, leilão, escrow, compliance.
6. **Evidence Service**
   - Evidência Tipo A/B, hash, device attest, GPS, anexos.
7. **Immutable Audit Ledger**
   - Append-only + hash encadeado entre eventos.
8. **Lock Service (CAR/IE/Property/Tenant)**
   - Bloqueia NF, venda, transferência, novos contratos, etc.
9. **Escrow + Split Settlement**
   - Custódia lógica, tranches, split por fornecedor/produtor/investidor/boitel.
10. **Event Orchestrator**

- Um "dispatcher" que executa qualquer operação como **evento padronizado**.

Regra de ouro: qualquer módulo cria "intenções" (requests), mas execução real é sempre via **Event Orchestrator**.

---

2) Padrão único de operação: "Tudo é Evento"

2.1 Fluxo padrão de qualquer ação

UI → API → EventOrchestrator → (Guards/RBAC) → TenantGuard → LockCheck → RulesEngine → StateMachine → EvidenceCheck → Execute → Audit → (Escrow/Split se aplicável)

Isso impede duplicação e garante coerência.

2.2 Estrutura de evento (modelo único)

```
events/{eventId}
```

- `eventType` (ex.: STOCK_WITHDRAW, ASSET_TRANSFER, MILK_TICKET, SALE_PDV, INVOICE_ISSUE, CONTRACT_MILESTONE)
- `subject` (o que afeta: itemId, assetId, contractId, poolId)
- `inputs` (dados)
- `requiredEvidence` (A/B)
- `ruleset` (regras aplicáveis)
- `stateTransition` (se houver)
- `escrowPlan` (se houver)
- `result` (ok/blocked + motivo)

✅ O mesmo padrão vale para estoque, leite, marketplace, custódia, boitel.

---

3) Módulos (plugáveis) e o que NÃO duplicam

3.1 Módulos que você já definiu (agrupados)

1. **Propriedades & Compliance**
   - CAR, Inscrição Estadual, geofence, limites ambientais
2. **Catálogo / Itens / Identidade**
   - produtos/insumos com identidade e rastreio
3. **Estoque (Fazenda/Loja/Fornecedor)**
   - saldo, movimentos append-only, aplicação de insumo
4. **Marketplace (Insumos + Produções + Leilão opcional)**
   - B2C restrito a cadastrados, raio por proximidade expansível
5. **Contratos (produção sob demanda / ciclos)**
   - etapas, marcos, garantias, assinatura digital
6. **Escrow & Split**
   - reserva de estoque só após pagamento; cancelamento com multa
7. **Custódia / Terceiros / Boitel**
   - dono seguro; travas contra NF/venda sem split
8. **Ativos Identificáveis**
   - bovinos com brinco/RFID; “filho segue mãe”
9. **Pools Não Separáveis**
   - leite no tanque, grãos em silo; quotas e split proporcional
10. **Leite**

- ticket diário, coleta, laudo, fórmula, escrow por coleta

11. **Intake OCR/Voz**

- foto caderno/voz → draft → revisão/execução

12. **Auditoria/Fiscalização**

- read-only + validação + parecer

3.2 Onde costuma duplicar (e como evitar)

- **Estoque**: módulo não pode recalcular saldo por si → só chama `StockMovementService`
- **Marketplace**: não cria escrow próprio → só chama `EscrowService.createPlan()`
- **Contratos**: não valida etapas "na mão" → usa `StateMachineService.transition()`
- **Leite**: não "paga direto" → sempre `EscrowMilkService` + split
- **Compliance**: módulo não decide travas sozinho → cria lock via `LockService`

---

4) O que ainda falta para operar 100% (gap list)

Vou listar em prioridade (P0 = sem isso não roda em produção).

P0 — indispensável para produção

1. **Event Orchestrator central**
   - Um executor único de eventos com pipeline completo.
2. **Modelo único de identidade**
   - `IdentityService` + regras "todo objeto tem ID único" (produto, NF, contrato, pool, etc.)
3. **Ledger de auditoria com hash encadeado (implementação real)**
   - Hoje temos "audit_events"; falta o encadeamento + verificador de integridade.
4. **Escrow real com PSP (Asaas)**
   - criar cobrança, receber webhooks, confirmar pagamento, executar split/payout.
5. **Locks integrados aos fluxos críticos**
   - NF, venda, transferência, novos contratos: garantido em todos endpoints.
6. **Política clara de estados (state machine)**
   - Mapear estados oficiais para: contrato, leilão, custódia, milk collection, delivery.
7. **Regras ambientais com fonte de dados**
   - Como o sistema "sabe" o limite? precisa: cadastro de limite/regra por CAR/UF/bioma, ou input de auditor/relatório.
8. **Controle de concorrência / idempotência universal**
   - Já existe em partes; falta padronizar para todos eventos.

P1 — essencial para escala e menos erro

9. **Sistema de permissões completo + seeds**
   - você pediu JSON de permissions; precisa virar "fonte única" e middleware de permissão.
10. **Módulo de "Documentos"**

- armazenar contratos assinados, laudos, NF anexas, termos de custódia.

11. **Notificações (push/SMS/WhatsApp/email)**

- avisos de bloqueios, marcos, pagamento, falta de ticket, etc.

12. **Monitoramento + trilhas de erro**

- logs, alertas de falhas de webhook, reconciliação de split.

P2 — fechamento do agro completo

13. **Integração NF-e/SEFAZ (ou conector)**

- mesmo que não emita, precisa "bloquear fluxo" antes da emissão.

14. **Logística**

- CT-e/manifesto, transportador, evidência de entrega, geofence.

15. **Módulo de qualidade e classificação (além do leite)**

- grãos, carcaça, pesagem, laudos.

16. **Conciliação financeira**

- relatório "escrow vs pago vs split vs pendente".

---

5) Checklist para dizer "está 100% operando"

O sistema está "100%" quando consegue fechar 3 jornadas sem buracos:

Jornada A — Insumo/Estoque (fazenda)

- lançar por foto/voz → draft → executar → saldo correto → auditoria imutável

Jornada B — Marketplace insumos (com reserva)

- oferta + estoque fornecedor real-time → pagamento escrow → reserva → entrega evidenciada → split pago → cancelamento com multa

Jornada C — Leite (anti-calote)

- ticket diário → coleta → laudo → fórmula preço → escrow por coleta → split por produtor → lock por inadimplência se não pagar

Se essas 3 rodam ponta a ponta, o "core" está pronto.

---

6) Recomendo a lógica final (sem duplicar): “Kernel + Adapters”

- **Kernel (Central)**: Orchestrator + Rules + State + Evidence + Audit + Locks + Escrow + Identity
- **Adapters (Módulos)**: Marketplace/Leite/Estoque/Contratos só chamam kernel via APIs internas

Isso reduz bug, retrabalho e garante prova jurídica.

Se você disser “sim”, eu te devolvo um **mapa oficial** (em formato bem prático para dev) com:

- lista de **eventTypes** padrão (estoque, venda, leite, custódia, contrato, NF)
- quais **regras** e **locks** se aplicam em cada um
- quais **evidências** são exigidas
- e o diagrama de dependências “Kernel → Módulos” (sem duplicação).

Abaixo está uma **estrutura fechada** do Ciclo+ (Kernel + Módulos), com **funções separadas sem duplicação**, e um **mapa de telas ↔ operações ↔ eventos** para tudo ficar correspondente no app e no backend.

---

1) Arquitetura final: Kernel + Módulos (sem duplicar)

1.1 Kernel (Central) — único e obrigatório

Tudo que é regra/segurança/lastro fica aqui, e os módulos só “pedem” operações.

Serviços do Kernel

1. **Identity** (IDs únicos para tudo)
2. **Auth + RBAC + Permissions**
3. **Tenant Isolation** (tenantId em tudo + rules Firestore)
4. **Event Orchestrator** (executor padrão de eventos)
5. **RulesEngine** (pré-execução)
6. **StateMachine** (transições formais)
7. **Evidence** (Tipo A/B + anexos + hash)
8. **Audit Ledger** (append-only + hash encadeado)
9. **Lock Service** (CAR/IE/contrato/inadimplência/custódia)
10. **Escrow + Split Settlement** (tranches/split/multa/cancelamento)
11. **Notification** (push/WhatsApp/e-mail)
12. **Geo/Proximity** (raio expansível, geofence)

Regra: **nenhum módulo** pode implementar “escrow”, “auditoria”, “regras”, “locks” por conta própria.

---

2) Módulos (Adapters) e responsabilidades

M1 — Propriedades & Compliance (CAR/IE/Ambiental)

- Cadastro da propriedade
- Vincular CAR e Inscrição Estadual
- Limites ambientais (regras por área)
- Gera locks: `ENV_LIMIT`, `CAR_BLOCK`, `IE_BLOCK`

M2 — Catálogo & Identidades (Produtos/Ativos/Contratos)

- Cadastro de itens com **identidade obrigatória**
- Sinônimos (para voz)
- Mapeamento NF/entradas → rastreio

M3 — Estoque (Fazenda / Loja / Fornecedor)

- Saldo por item/local
- Movimentação append-only
- Aplicação/baixa por operação (inclusive por foto/voz via Intake)

M4 — Intake (Foto/Voz/OCR) — Operação sem digitar

- Captura evidência + extração + draft + revisão + execução
- Nunca substitui NF; só operacionaliza

M5 — Marketplace (Insumos + Produções)

- Buscar por proximidade (raio expandível)
- B2C restrito a cadastrados
- Reserva de estoque **somente após escrow**
- Cancelamento com multa
- Oferta: “Compra”, “Venda”, “Leilão (opcional)”

M6 — Contratos (Produção sob demanda / Ciclos)

- Negociação → contrato gerado → assinatura digital
- Ciclos: bovinos (cria/recria/engorda), aves, peixes, plantações, silvicultura etc.
- Marcos verificáveis → liberam tranches

M7 — Custódia / Terceiros / Boitel / Ativos

- Ativo unitário (bovinos com brinco/RFID)
- “Filho segue mãe”
- Custódia com locks anti-fraude (NF/venda/transferência)

M8 — Leite (anti-calote)

- Ticket diário do tanque

- Coleta consolidada
- Laudo e fórmula de preço
- Escrow por coleta + split por produtor
- Lock `NON_PAYMENT` para indústria inadimplente

M9 — PDV / Loja (split por fornecedor)

- Venda balcão: itens de vários fornecedores → split automático
- Rastreamento por identidade (NF/código/registro)

M10 — Auditoria/Fiscalização (read-only)

- Painel read-only
- Validação de evidências
- Parecer e compliance

---

3) Modelo operacional único: "Eventos" (para todos os módulos)

3.1 Eventos padrão (exemplos)

- `STOCK_DEPOSIT`, `STOCK_WITHDRAW`, `STOCK_APPLY`
- `MARKETPLACE_RESERVE`, `MARKETPLACE_CANCEL`, `MARKETPLACE_FULFILL`
- `CONTRACT_CREATE`, `CONTRACT_SIGN`, `CONTRACT_MILESTONE_VALIDATE`
- `ESCROW_CREATE`, `ESCROW_PAID`, `ESCROW_RELEASE_TRANCHE`, `SPLIT_EXECUTE`
- `ASSET_BIRTH`, `ASSET_TRANSFER`, `CUSTODY_LOCK`
- `MILK_TANK_CHECK`, `MILK_COLLECTION_BUILD`, `MILK_PRICE`, `MILK_ESCROW_PAY`
- `PDV_SALE`, `PDV_SPLIT`

Todos passam por:
RBAC → Tenant → Locks → Rules → State → Evidence → Execute → Audit → Settlement

---

4) Telas ideais (Mobile) e operações correspondentes

4.1 Mobile — Home (Acessível)

Botões grandes (4 a 6)

1. **Estoque**
2. **Leite**
3. **Marketplace**
4. **Contratos**
5. **Registrar por Foto**
6. **Registrar por Voz**

Operações

- Navega sempre para “fluxo guiado”
- Cada fluxo termina criando um evento via kernel

4.2 Estoque (Fazenda/Loja)

Telas

- **Saldo Atual** (por item/local)
- **Retirar/Usar** (fluxo rápido)
- **Entrada** (compra/chegada)
- **Aplicação no campo** (talhão/lote/animal)
- **Histórico (auditoria)**

Operações ↔ Eventos

- Retirar → `STOCK_WITHDRAW`
- Entrada → `STOCK_DEPOSIT`
- Aplicar → `STOCK_APPLY` (pode exigir evidência Tipo B em operações críticas)

Entrada por foto/voz

- `intake.create` → `intake.extract` → `to-draft` → `draft.execute`

4.3 Marketplace (Insumos / Produções)

Telas

- **Explorar** (raio expansível + filtro “somente cadastrados”)
- **Oferta** (produto/insumo/produção)
- **Detalhe da oferta**
- **Comprar/Vender**
- **Leilão** (se ativo livre)
- **Pagamento (Escrow)**
- **Entrega / Evidência**

Operações ↔ Eventos

- Reservar estoque → `MARKETPLACE_RESERVE` (só após `ESCROW_PAID`)
- Cancelar → `MARKETPLACE_CANCEL` (multa via escrow)
- Entrega → `MARKETPLACE_FULFILL` (evidência)
- Split pagamento → `SPLIT_EXECUTE`

Regra de ouro

- “Reservar” só ocorre quando escrow confirmado (anti-fraude).

4.4 Contratos (produção sob demanda)

Telas

- **Criar negociação** (IA sugere cláusulas e padrões)
- **Gerar contrato** (template + variáveis)
- **Assinar digitalmente**
- **Marcos/Etapas** (checklist)
- **Evidências por etapa**
- **Pagamentos por tranches**
- **Bloqueios e penalidades**

Operações ↔ Eventos

- Criar contrato → `CONTRACT_CREATE`
- Assinar → `CONTRACT_SIGN`
- Validar marco → `CONTRACT_MILESTONE_VALIDATE`
- Liberar tranche → `ESCROW_RELEASE_TRANCHE`

4.5 Leite (anti-calote)

Telas

- **Ticket diário (tanque)** (1 clique + foto + GPS)
- **Histórico tickets**
- **Coleta do dia** (consolidado)
- **Preço (fórmula)**
- **Escrow / Pagamento**
- **Split por produtor**
- **Alertas de inadimplência**

Operações ↔ Eventos

- Ticket → `MILK_TANK_CHECK`
- Coleta → `MILK_COLLECTION_BUILD`
- Laudo → `LAB_ATTACH`
- Precificar → `MILK_PRICE`
- Cobrar → `MILK_ESCROW_PAY`
- Pagar → `ESCROW_PAID` → `PAYOUTS_QUEUED`

4.6 Custódia/Ativos (boitel, terceiros)

Telas

- **Ativos sob custódia (lista)**
- **Detalhe do ativo** (owner/custodian)
- **Movimentar / Transferir** (exige evidência)
- **Bloqueios** (locks)
- **Pagamento / split (dono/boitel)**

Operações ↔ Eventos

- Nascimento → `ASSET_BIRTH` (filho segue mãe)
- Transferência → `ASSET_TRANSFER`
- Lock automático terceiros → `CUSTODY_LOCK`

4.7 Auditoria (read-only)

Telas

- **Painel de eventos**
- **Ver evidências**
- **Ver hashes/integridade**
- **Parecer**
- **Conformidade ambiental e contratual**

---

5) Tabela de correspondência (Telas ↔ Eventos ↔ Evidência)

Tela	Evento(s)	Evidência mínima	Bloqueios (Locks) comuns
Retirar Insumo (rápido)	STOCK_WITHDRAW	Tipo A ou Foto caderno	ENV_LIMIT opcional, saldo
Aplicar Insumo	STOCK_APPLY	Tipo B (foto + GPS) se crítico	ENV_LIMIT, contrato
Comprar no Marketplace	MARKETPLACE_RESERVE + ESCROW_PAID	Pagamento + evidência entrega	NON_PAYMENT, CAR/IE
Venda PDV (multi fornecedor)	PDV_SALE + SPLIT_EXECUTE	NF/código/identidade	Locks de custódia/terceiros
Ticket Leite Diário	MILK_TANK_CHECK	Tipo B (foto tanque + GPS)	Falta ticket gera alerta
Coleta Leite	MILK_COLLECTION_BUILD + MILK_ESCROW_PAY	Tickets + laudo	NON_PAYMENT indústria

Tela	Evento(s)	Evidência mínima	Bloqueios (Locks) comuns
Transferir Bovino	ASSET_TRANSFER	Tipo B + validação	THIRD_PARTY_CUSTODY, contrato
Emissão NF (quando integrar)	INVOICE_ISSUE	Doc fiscal real	CAR/IE/terceiros/inadimplência

---

6) O que falta para fechar "telas e operações 100% correspondentes"

1. **Mapa oficial de estados** (state machine) por módulo (contrato, leilão, escrow, custódia, leite).
2. **Catálogo de eventTypes** final (lista fechada) + "requiredEvidence" por evento.
3. **Design System Acessível** (componentes padrão: Botão Gigante, VoiceConfirm, StepWizard).
4. **Conector PSP (Asaas)** completo (criar cobrança + webhook + payout).
5. **Reconciliação**: tela/serviço que compara `escrow` vs `payouts` vs `split` e aponta pendências.

Se você quiser, eu já devolvo o **"Mapa Oficial do Produto"** em formato prático para dev (pode colar no seu repositório):

- `eventTypes.json` (todos os eventos + evidência + locks + rulesets)
- `stateMachines.json` (tabelas de transição por domínio)
- `permissions.json` (roles x permissions)
- e um **wireframe** do app com navegação completa (Home → módulos → fluxos).