Technical Documentation for DobDex x Stellar

Score-Aware, LP-Priced Exit Liquidity for Tokenized Real Assets on Soroban

Stage 3: Exit Liquidity Layer (6-Month Roadmap) — SCF Build, Integration Track

1. Executive Summary

This document outlines a realistic 6-month development plan for DobDex, the exit-liquidity layer of the Dob Protocol stack on Stellar. DobDex closes the one gap that keeps institutional capital out of tokenized real-world assets (RWAs): investors holding tokenized positions currently have no mechanism to exit before maturity. Automated market makers price on pool depth, not asset performance, so a high-performing asset is indistinguishable from a failing one. DobDex prices exits against Dob Validator's on-chain risk scores and executes them through three sequential layers — LP auction, pool reserve, and operator co-liquidity — settling to USDC or fiat through Stellar anchors.

The positioning is deliberate and consistent across our stack: score-aware, LP-priced exit liquidity for validated Stellar RWAs — not fixed-price or zero-slippage exits. Exit price is set by competing liquidity providers against a live risk score, never by a static oracle peg.

The design is already proven. A full reference implementation of this mechanism runs on EVM testnets (Unichain, Arbitrum, Base) as a Uniswap V4 Custom Accounting hook with a permissionless LP registry and a pooled liquidity node. This grant ports that proven architecture to six native Soroban contracts and wires it into the live Stellar stack (Dob Validator, DeFindex, Blend, and Stellar anchors).

Deliverable Roadmap

Six new Soroban contracts plus the exit interface and LP dashboard, delivered across three tranches.

Budget Allocation Note

Funding under the Stellar Build Award is distributed in three equal parts (⅓ at approval, ⅓ upon MVP delivery, ⅓ upon mainnet launch). We specify our budget per tranche based on the actual engineering work and hours committed at each stage, to keep resource allocation transparent and show how each milestone is supported by a proportionate investment of effort. DobDex is

submitted under the Integration Track: it extends an existing, live Stellar stack rather than building a platform from zero, so the total is scoped accordingly. All figures below are adjustable.

Tranche 1 – MVP

- Score Reader contract: read Validator approval status, TRUFA score, certificate hash, and freshness (Weeks 1–2)
- Risk Classifier contract: map validator score to liquidity tiers, reserve ratios, and discount ranges (Weeks 3–4)
- Liquidity Vault contract: LP USDC deposits, hot reserve accounting, share tracking (Weeks 5–6)
- Internal testing, unit coverage, and iterative improvements (Weeks 7–8)

Budget Allocation: **$38,270 (~26.7% of total)**

Tranche 2 – Testnet

- Exit Auction Engine: escrow, funded LP bids, best-net-payout selection (Weeks 9–10)
- Settlement Router: RWA transfer to winning LP, USDC to seller or managed wallet (Weeks 11–12)
- FIFO Queue Manager for underfilled exits + end-to-end integration testing on testnet (Weeks 13–14)

Budget Allocation: **$45,006 (~31.4% of total)**

Tranche 3 – Mainnet

- DeFindex + Blend integration: route idle LP capital to yield while uninvested (Week 15)
- Anchor off-ramp integration (SEP-24) for LATAM fiat settlement (Week 16)
- Exit interface + LP dashboard with contract event indexing (Weeks 17–18)
- Deploy all six contracts to Stellar mainnet
- Onboard a minimum of 5 validated assets with live exit liquidity
- Begin live monitoring, optimization, and structured feedback collection

Budget Allocation: **$60,057 (~41.9% of total)**

Total: $143,333 over 6 months.

2. Scope & Priorities

2.1 What We're Building

- **Score Reader:** on-chain bridge to Dob Validator for approval status, TRUFA score, certificate hash, and freshness.
- **Risk Classifier:** deterministic mapping from validator score to liquidity tiers (reserve ratio, discount range, max exit size, eligible LPs).

- **LP Liquidity Vault:** permissionless USDC deposits with a hot reserve for immediate exits and idle capital routed to yield.
- **Exit Auction Engine:** sealed escrow of seller RWA, funded LP bids in a fixed window, best-net-payout wins.
- **Settlement Router:** atomic RWA-to-LP and USDC-to-seller settlement, with managed-wallet and anchor off-ramp paths.
- **FIFO Queue Manager:** underfilled exits queued and filled as new liquidity or bids arrive.
- **Yield on Idle Capital:** DeFindex strategies (potentially Blend underneath) so uninvested LP USDC accrues yield.
- **Fiat Off-Ramp:** single Stellar anchor integration (SEP-24) for LATAM settlement.
- **Exit Interface & LP Dashboard:** request exits, submit bids, and index exit/bid/settlement/queue events.

Out of scope for this stage: cross-chain messaging, secondary RWA order books, and ML-based dynamic discounting. These are noted in Section 9.

3. Technical Architecture

DobDex is an integration layer on top of the existing Stellar/Soroban Dob Protocol stack. It adds no new trust assumptions: Dob Validator remains the single source of truth for asset approval and risk data, and all pricing authority sits with competing LPs. The architecture is composed of the following elements:

Existing live dependencies (mainnet): Dob Validator (approval + TRUFA risk score), DobTokenizer (RWA token contracts), DobPay (yield distribution), managed wallet infrastructure, DeFindex + Blend (yield), and Stellar anchors (fiat on/off ramp).

New Soroban contracts: Score Reader, Risk Classifier, Liquidity Vault, Exit Auction Engine, Settlement Router, FIFO Queue Manager.

Frontend: React + TypeScript + TailwindCSS exit interface and LP dashboard.

Backend / indexer: Node.js + TypeScript event indexer over Soroban RPC; PostgreSQL for exit/bid/settlement history; Redis cache.

Settlement rails: USDC on Stellar, plus Alchemy Pay (or equivalent) anchor via SEP-24 for fiat off-ramp.

Component Interaction

A token holder requests an exit from the React interface. The backend checks KYC/identity status, asset eligibility, and score freshness via the Score Reader, then the Risk Classifier resolves the asset's liquidity tier. The Exit Auction Engine locks the seller's RWA in escrow and opens a fixed bidding window; approved LPs submit funded USDC bids from the Liquidity Vault, and the best valid bid — ranked by net USDC payout to the seller — wins. The Settlement Router

atomically transfers RWA tokens to the winning LP and sends USDC to the seller, a managed dob.capital wallet, or an anchor off-ramp. If an exit is underfilled, the remainder enters the FIFO Queue Manager and is processed when new LP liquidity arrives (including capital withdrawn from DeFindex/Blend) or new bids appear. The indexer subscribes to contract events for exits, bids, settlements, queue status, and liquidity movements, feeding the dashboard.

Lineage from the EVM reference implementation

Each Soroban contract maps to a proven component already running on EVM testnets, de-risking the build:

Soroban contract (new)	EVM reference (existing)
Score Reader	DobValidatorRegistry oracle reads
Risk Classifier	LP condition checks (minOraclePrice, tiers)
Liquidity Vault	DobPooledLN + DobLPRegistry USDC accounting
Exit Auction Engine	DobLPRegistry.queryAndFillAtMarket (FIFO LP fills)
Settlement Router	DobPegHook custom-accounting settlement
FIFO Queue Manager	LPRegistry FIFO fallback fill ordering

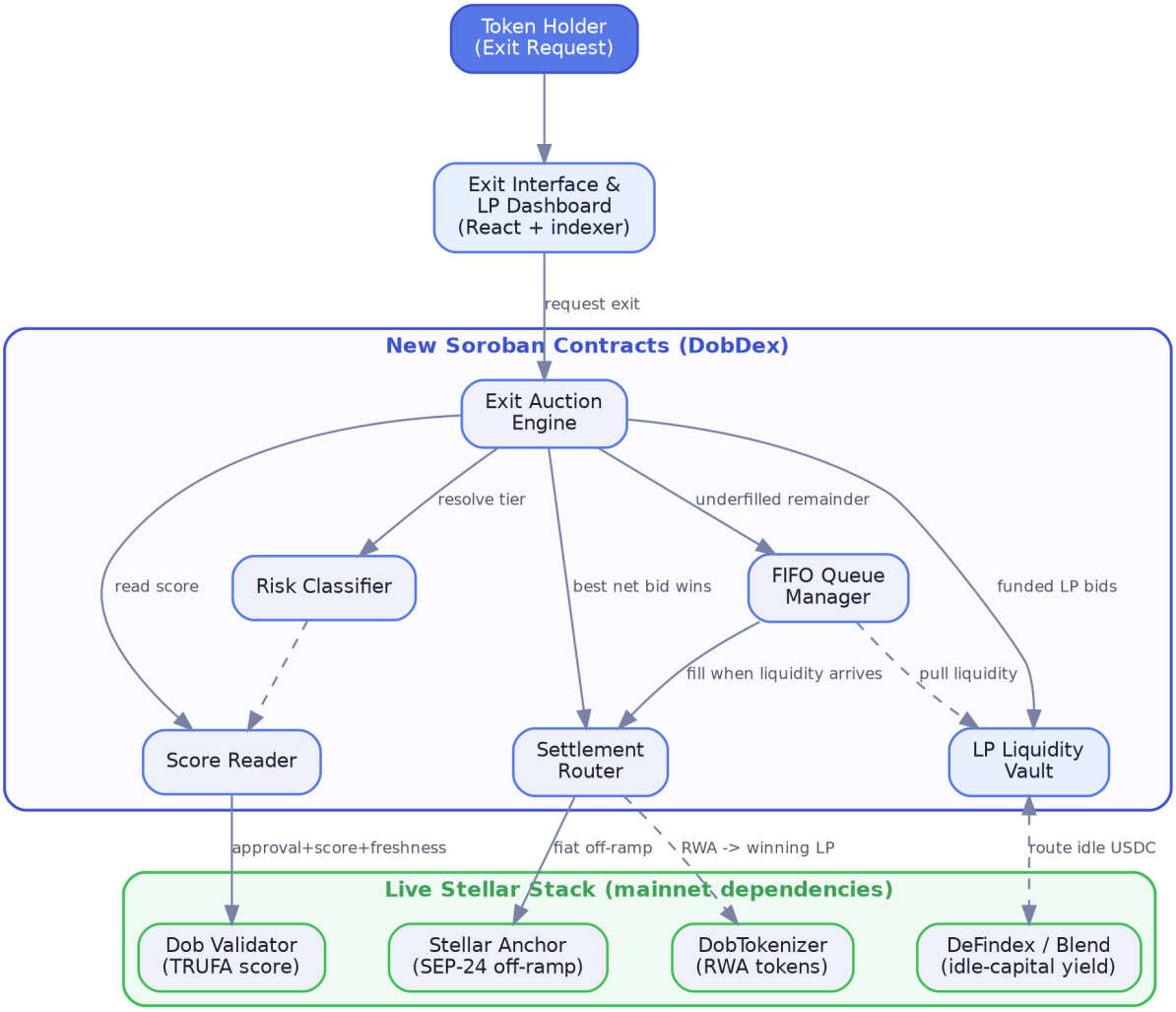

Fig 1. System architecture — exit request flows through the six DobDex Soroban contracts (blue) and settles against the live Stellar stack (green). Dashed edges denote asset/liquidity movement.

3.1 Contract Interfaces

Each contract exposes a narrow, auditable surface. The public entry points are:

```
// ---- Score Reader ----
pub trait ScoreReader {
    fn read(env: Env, asset: Address) -> AssetScore;      // approval, trufa, cert_hash,
updated_at
    fn is_fresh(env: Env, asset: Address) -> bool;        // updated_at within staleness
window
}

// ---- Risk Classifier ----
pub trait RiskClassifier {
    fn classify(env: Env, asset: Address) -> Tier;        // reads ScoreReader, returns
tier params
    fn set_tier_params(env: Env, tier: Symbol, p: TierParams);  // governance-gated
    fn is_eligible(env: Env, lp: Address, tier: Symbol) -> bool;
```

```
}

// ---- LP Liquidity Vault ----
pub trait LiquidityVault {
    fn deposit(env: Env, lp: Address, amount: i128) -> i128;       // -> shares
    fn withdraw(env: Env, lp: Address, shares: i128) -> i128;      // -> usdc (pulls from
DeFindex if short)
    fn hot_reserve(env: Env) -> i128;
    fn reserve_for_bid(env: Env, auction_id: u64, amount: i128);   // auth: Auction
Engine
    fn release_reserved(env: Env, auction_id: u64);
}

// ---- Exit Auction Engine ----
pub trait AuctionEngine {
    fn request_exit(env: Env, seller: Address, asset: Address, amount: i128) -> u64;  //
-> auction_id
    fn submit_bid(env: Env, lp: Address, auction_id: u64, funded_usdc: i128,
discount_bps: u32);
    fn close(env: Env, auction_id: u64) -> ExitOutcome;   // selects best net bid,
triggers settle/queue
    fn cancel(env: Env, seller: Address, auction_id: u64);
}

// ---- Settlement Router ----
pub trait SettlementRouter {
    fn settle(env: Env, auction_id: u64, dest: Destination);  // atomic RWA->LP,
USDC->seller/wallet/anchor
}

// ---- FIFO Queue Manager ----
pub trait QueueManager {
    fn enqueue(env: Env, auction_id: u64, remainder: i128);   // auth: Auction Engine
    fn process(env: Env, max_items: u32);                     // fills head-of-queue
while liquidity exists
    fn cancel(env: Env, seller: Address, queue_id: u64);
}
```

3.2 Storage Layout & Types

Contracts use Soroban persistent storage keyed by enum discriminants; auction/queue records live in instance storage with TTL bumps on write. Core types:

```
#[contracttype]
pub struct AssetScore { pub approved: bool, pub trufa: u32, pub cert_hash: BytesN<32>,
pub updated_at: u64 }

#[contracttype]
pub struct TierParams {
    pub reserve_ratio_bps: u32,   // hot reserve required to open an auction
    pub min_discount_bps: u32,    // narrowest LP discount allowed
    pub max_discount_bps: u32,    // widest LP discount allowed
    pub max_exit_size: i128,      // per-auction cap for this tier
}
```

```
#[contracttype]
pub struct Auction {
    pub seller: Address, pub asset: Address, pub amount: i128,
    pub tier: Symbol, pub best_bid: Option<Bid>, pub state: ExitState, pub closes_at:
u64,
}

#[contracttype] pub struct Bid { pub lp: Address, pub funded_usdc: i128, pub
discount_bps: u32 }

#[contracttype] pub enum ExitState { Requested, Escrowed, Auctioning, Settled, Queued,
Cancelled }

#[contracttype] pub enum DataKey {
    Auction(u64), Queue(u64), Shares(Address), TierCfg(Symbol), Reserved(u64), Config,
}
```

3.3 Error Handling

```
#[contracterror]
pub enum Error {
    NotApproved = 1,      // asset rejected by Dob Validator
    StaleScore = 2,       // score older than staleness window
    NotEligible = 3,      // LP not permitted for this tier
    BidOutOfRange = 4,    // discount outside tier band
    InsufficientReserve = 5,
    WindowClosed = 6,
    NotSeller = 7,
    AlreadySettled = 8,
}
```

4. Deliverables

Fig 2. Exit lifecycle — an exit passes through eight stages; a partial fill branches to the FIFO queue and settles later when liquidity arrives.

4.1 Score Reader Contract

Brief Description. On-chain bridge that reads Dob Validator state so every exit decision is anchored to verified asset data, not off-chain assumptions.

Simplified Implementation

- Reads approval status, TRUFA score, certificate hash, and score timestamp for a given asset
- Freshness guard: exits are blocked when the score is older than a configurable staleness window
- Read-only, no privileged writes; callable by the Auction Engine and the backend

User Stories

User Story 1: "As the protocol, I want every exit priced against a fresh validator score, so that liquidity reflects real asset risk"

Acceptance Criteria:

Given an asset with a Dob Validator record

When an exit is requested

Then the Score Reader returns approval status, TRUFA score, and freshness

And stale or rejected assets are excluded from auctions

User Story 2: "As an LP, I want to trust the score behind an asset, so that I can bid confidently"

Acceptance Criteria:

Given a validated asset

When reviewing an exit request

Then see the on-chain certificate hash and score timestamp

And verify it against the Validator record

Smart Contract Code (Simplified)

```
// Score Reader (Soroban) — read-only bridge to Dob Validator
const STALENESS_SECS: u64 = 86_400; // 24h; governance-updatable

#[contractimpl]
impl ScoreReader {
    pub fn read(env: Env, asset: Address) -> AssetScore {
        // cross-contract call into the live Validator registry
        let v = validator_client(&env);
        AssetScore {
            approved:   v.is_approved(&asset),
            trufa:      v.score_of(&asset),
            cert_hash:  v.cert_hash(&asset),
            updated_at: v.updated_at(&asset),
        }
    }
    pub fn is_fresh(env: Env, asset: Address) -> bool {
        let s = Self::read(env.clone(), asset);
        s.approved && (env.ledger().timestamp() - s.updated_at) <= STALENESS_SECS
    }
}
```

4.2 Risk Classifier Contract

Brief Description. Deterministic mapping from a validator score to DobDex liquidity tiers that define how much liquidity an asset receives and on what terms.

Simplified Implementation

- Maps TRUFA score ranges to tiers (e.g. A–E), each with a reserve ratio, discount range, and max exit size
- Defines which LPs are eligible to bid on each tier
- Rule-based and transparent (not ML); parameters are governance-updatable
- High-risk, stale, or rejected assets can be paused, manually reviewed, or excluded

User Stories

User Story 1: "As an operator, I want my asset's liquidity terms derived from its validated score"

Acceptance Criteria:

Given a completed validation with a TRUFA score

When the classifier runs

Then the asset receives a tier with defined reserve ratio and discount range

And the operator sees the tier breakdown

User Story 2: "As an investor, I want risky assets clearly gated, so that liquidity is safe"

Acceptance Criteria:

Given a low-score or stale asset

When classification runs

Then the asset is capped, paused, or excluded from auctions

And the reason is visible on-chain

Reference Tier Mapping (governance-tunable)

Tier	TRUFA	Reserve ratio	Discount band	Max exit	Eligible LPs
A	85–100	10%	1.0–2.5%	$250k	All
B	70–84	20%	1.5–4.0%	$150k	All
C	55–69	35%	3.0–6.0%	$75k	Verified
D	40–54	50%	5.0–9.0%	$25k	Whitelisted
E	0–39	—	—	paused	—

Illustrative values. Reserve ratio is the share of an exit that must be sittable in the hot reserve before an auction opens; the discount band bounds valid LP bids; max exit caps per-auction size. E-tier assets are excluded pending manual review.

Smart Contract Code (Simplified)

```
// Risk Classifier (Soroban)
#[contractimpl]
impl RiskClassifier {
    pub fn classify(env: Env, asset: Address) -> Tier {
        let s = score_reader(&env).read(&asset);
        if !s.approved { return Tier::excluded(Symbol::new(&env, "E")); }
        let sym = match s.trufa {
            85..=100 => "A", 70..=84 => "B", 55..=69 => "C", 40..=54 => "D", _ => "E",
        };
        let key = Symbol::new(&env, sym);
        let params: TierParams =
env.storage().persistent().get(&DataKey::TierCfg(key.clone()))
            .unwrap_or_else(|| panic_with_error!(&env, Error::NotApproved));
        Tier { symbol: key, params }
    }
}
```

4.3 LP Liquidity Vault Contract

Brief Description. Permissionless USDC vault where LPs deposit capital to fund exits, keeping a hot reserve for immediate fills and routing idle capital to yield.

Simplified Implementation

- LPs deposit USDC and receive proportional shares
- Hot reserve kept liquid for immediate exits; surplus routed to DeFindex strategies (potentially Blend)
- Idle capital withdrawn on demand when the FIFO queue needs liquidity
- Ports the proven DobPooledLN / DobLPRegistry accounting from the EVM implementation

User Stories

User Story 1: "As an LP, I want to deposit USDC and earn yield while idle"

Acceptance Criteria:

Given an approved LP

When depositing USDC

Then receive vault shares

And idle capital accrues yield via DeFindex/Blend until needed for a fill

User Story 2: "As an LP, I want to withdraw my capital and accrued returns"

Acceptance Criteria:

- **Given** vault shares
- **When** requesting withdrawal
- **Then** idle capital is pulled back from DeFindex if required
- **And** USDC plus proportional yield is returned

Smart Contract Code (Simplified)

```
// LP Liquidity Vault (Soroban)
#[contract]
pub struct LiquidityVault;

#[contractimpl]
impl LiquidityVault {
    pub fn deposit(env: Env, lp: Address, amount: i128) -> Result<i128, Error> {
        lp.require_auth();
        // pull USDC from the LP into the vault
        token::transfer(&env, &lp, &env.current_contract_address(), amount);

        // mint proportional shares
        let shares = shares_for(&env, amount);
        credit_shares(&env, &lp, shares);

        // keep hot reserve, route surplus to DeFindex/Blend for yield
        route_idle_to_yield(&env);
        Ok(shares)
    }

    pub fn fund_bid(env: Env, auction_id: u64, amount: i128) -> Result<(), Error> {
        // only the Auction Engine may draw funded bids
        require_auction_engine(&env);
        ensure_hot_reserve(&env, amount); // pull from DeFindex if short
        reserve_for_bid(&env, auction_id, amount);
        Ok(())
    }
}
```

4.4 Exit Auction Engine

Brief Description. The core exit mechanism: escrow the seller's RWA, collect funded LP bids in a fixed window, and settle to the bid that pays the seller the most net USDC.

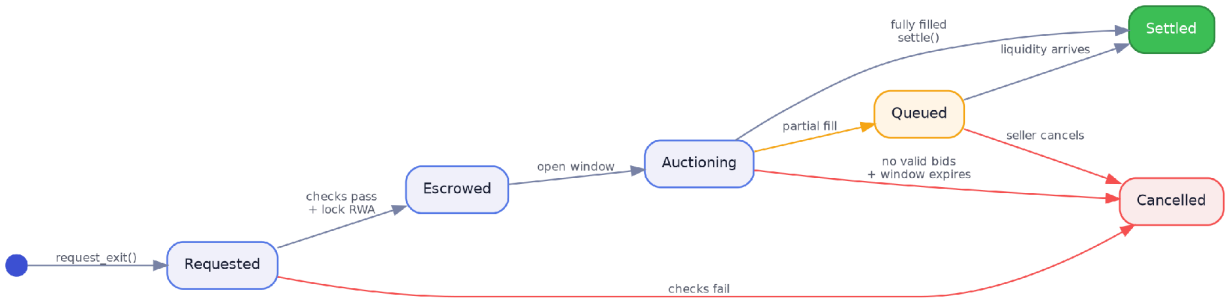

Fig 3. Exit auction state machine — an exit is Requested, Escrowed, then Auctioned; it resolves to Settled, Queued (then Settled), or Cancelled.

Simplified Implementation

- On exit request, verifies KYC, asset eligibility, score freshness, and liquidity conditions
- Locks the seller's RWA tokens in escrow for the auction window
- Approved LPs submit funded bids drawn from the Liquidity Vault
- Best valid bid wins by net USDC payout to the seller; ties broken FIFO
- Underfilled amounts are handed to the FIFO Queue Manager

User Stories

User Story 1: "As a token holder, I want to exit my position before maturity"

Acceptance Criteria:

Given a validated asset and passing KYC

When requesting an exit for an amount

Then my RWA is escrowed and an auction window opens

And I see the winning net USDC payout before confirming

User Story 2: "As an LP, I want to bid on exits at my own price"

Acceptance Criteria:

Given eligibility for the asset's tier

When the auction window is open

Then submit a funded USDC bid within the tier discount range

And win if my net payout to the seller is the highest valid bid

User Story 3: "As a token holder, I want a fair result if liquidity is thin"

Acceptance Criteria:

Given an exit only partially covered by bids

When the window closes

Then the filled portion settles immediately

And the remainder enters the FIFO queue

Smart Contract Code (Simplified)

```
// Exit Auction Engine (Soroban)
#[contractimpl]
impl AuctionEngine {
    pub fn request_exit(env: Env, seller: Address, asset: Address, amount: i128) -> u64 {
        seller.require_auth();
        // gate: fresh, approved score + KYC + tier cap
        if !score_reader(&env).is_fresh(&asset) { panic_with_error!(&env,
Error::StaleScore); }
        require_kyc(&env, &seller);
        let tier = classifier(&env).classify(&asset);
        if amount > tier.params.max_exit_size { panic_with_error!(&env,
Error::BidOutOfRange); }

        // escrow the seller's RWA for the auction window
        token::transfer(&env, &seller, &env.current_contract_address(), &asset, amount);
        let id = next_id(&env);
        let closes = env.ledger().timestamp() + AUCTION_WINDOW;
        put_auction(&env, id, Auction {
            seller, asset, amount, tier: tier.symbol,
            best_bid: None, state: ExitState::Auctioning, closes_at: closes,
        });
        env.events().publish((symbol!("exit"), symbol!("opened")), (id, amount));
        id
    }

    pub fn submit_bid(env: Env, lp: Address, id: u64, funded_usdc: i128, discount_bps:
u32) {
        lp.require_auth();
        let mut a = get_auction(&env, id);
        if env.ledger().timestamp() > a.closes_at { panic_with_error!(&env,
Error::WindowClosed); }
        let p = tier_params(&env, &a.tier);
        if discount_bps < p.min_discount_bps || discount_bps > p.max_discount_bps {
            panic_with_error!(&env, Error::BidOutOfRange);
        }
        if !classifier(&env).is_eligible(&lp, &a.tier) { panic_with_error!(&env,
Error::NotEligible); }

        // fund the bid from the vault's hot reserve and keep the highest net payout
        vault(&env).reserve_for_bid(id, funded_usdc);
        if net_payout(funded_usdc, discount_bps) > a.best_net() {
            if let Some(prev) = &a.best_bid {
vault(&env).release_reserved(prev_auction(prev)); }
            a.best_bid = Some(Bid { lp, funded_usdc, discount_bps });
            put_auction(&env, id, a);
        }
```

```
    }

    pub fn close(env: Env, id: u64) -> ExitOutcome {
        let a = get_auction(&env, id);
        match a.best_bid {
            Some(b) if b.funded_usdc >= a.amount_value() => {
                settlement(&env).settle(id, Destination::Seller); ExitOutcome::Settled
            }
            Some(_) => { queue(&env).enqueue(id, a.unfilled()); ExitOutcome::Queued }
            None    => { refund_escrow(&env, &a); ExitOutcome::Cancelled }
        }
    }
}
```

4.5 Settlement Router

Brief Description. Atomically settles a won auction — RWA to the LP, USDC to the seller — with optional routing to a managed wallet or fiat off-ramp.

Simplified Implementation

- Transfers escrowed RWA to the winning LP and USDC to the seller in one atomic operation
- Destination options: seller wallet, managed dob.capital wallet, or anchor off-ramp (SEP-24)
- Emits settlement events for the indexer and dashboard
- Reverts cleanly if any leg fails, returning escrow to the seller

User Stories

User Story 1: "As a seller, I want to receive USDC or fiat when my exit settles"

Acceptance Criteria:

Given a winning bid

When settlement executes

Then the LP receives the RWA and I receive USDC

And I may route the USDC to my bank via the anchor off-ramp

Smart Contract Code (Simplified)

```
// Settlement Router (Soroban) — atomic, all-or-nothing
#[contractimpl]
impl SettlementRouter {
    pub fn settle(env: Env, id: u64, dest: Destination) {
        require_auction_engine(&env);
        let a = get_auction(&env, id);
        let bid = a.best_bid.unwrap();

        // leg 1: escrowed RWA -> winning LP
        token::transfer(&env, &env.current_contract_address(), &bid.lp, &a.asset,
a.amount);
```

```
        // leg 2: funded USDC -> destination (both legs in one host transaction =>
atomic)
        match dest {
            Destination::Seller       => usdc_transfer(&env, &a.seller, bid.funded_usdc),
            Destination::Managed(w)    => usdc_transfer(&env, &w, bid.funded_usdc),
            Destination::Anchor(memo)  => anchor_offramp(&env, memo, bid.funded_usdc), //
SEP-24
        }
        mark_settled(&env, id);
        env.events().publish((symbol!("exit"), symbol!("settled")), (id,
bid.funded_usdc));
    }
}
```

4.6 FIFO Queue Manager

Brief Description. Handles exits that could not be fully filled, processing them fairly in first-in first-out order as liquidity becomes available.

Simplified Implementation

- Queues underfilled exit remainders with their tier and requested terms
- Triggers processing when new LP liquidity arrives, idle capital is pulled from DeFindex/Blend, or new bids appear
- Strict FIFO ordering; sellers can cancel a queued remainder and reclaim escrow
- Emits queue-status events for transparency

User Stories

User Story 1: "As a seller in the queue, I want to be filled fairly and be able to exit the queue"

Acceptance Criteria:

> **Given** a queued exit remainder
>
> **When** new liquidity arrives
>
> **Then** my position fills in FIFO order
>
> **And** I may cancel and reclaim my escrowed RWA at any time

4.7 Exit Interface & LP Dashboard

Brief Description. React interface for token holders to request exits and for LPs to manage liquidity and bid, backed by a Soroban event indexer.

Simplified Implementation

- Exit flow: select asset + amount, see live score/tier, preview net payout, confirm
- LP flow: deposit/withdraw, view eligible auctions, submit funded bids, track earned RWA
- Indexer over Soroban RPC for exits, bids, settlements, queue status, and liquidity movements

- Reuses the design system and component patterns from the existing DobDex web app

User Stories

User Story 1: "As a token holder, I want a clear exit experience"

Acceptance Criteria:

Given a validated holding

When requesting an exit

Then see the asset score, tier, discount range, and expected net USDC

And confirm in a single guided flow

5. Implementation Timeline

Month 1–2: Foundation (MVP)

- Week 1–2: Score Reader contract + Validator integration
- Week 3–4: Risk Classifier + liquidity tier parameters
- Week 5–6: Liquidity Vault (deposits, shares, hot reserve)
- Week 7–8: Unit testing & iterations

Month 2.5–3.5: Core Mechanism (Testnet)

- Week 9–10: Exit Auction Engine (escrow, funded bids, selection)
- Week 11–12: Settlement Router
- Week 13–14: FIFO Queue Manager + end-to-end integration testing

Month 4: Launch Preparation

- Week 15: DeFindex + Blend yield integration
- Week 16: Anchor off-ramp (SEP-24) integration
- Week 17–18: Exit interface + LP dashboard + event indexer; mainnet deployment

Months 5–6: Operations & Iteration

- Onboard 5+ validated assets with live exit liquidity
- Seed LP vault and run real exits
- Monitor, optimize, and gather feedback for the next stage

6. Success Metrics Tracking

For SCF Requirements:

Six Soroban contracts deployed to mainnet

- Score Reader, Risk Classifier, Liquidity Vault, Auction Engine, Settlement Router, FIFO Queue Manager
- Full test coverage; one external audit before mainnet (outside this budget)

5+ validated assets with live exit liquidity

- Drawn from the existing pipeline (EV charging, solar, mining, hydro)
- Each classified into a liquidity tier from its TRUFA score

Live exits executed end-to-end

- Target: 10+ real exits settled to USDC on mainnet
- At least one exit routed to fiat via the anchor off-ramp
- Idle LP capital earning yield via DeFindex/Blend

7. Technical Specifications

7.1 Simplified API Structure

```
/api/auth
  POST /login

/api/exit
  POST /request          // open an exit auction
  GET  /:id              // auction status + best bid
  POST /:id/confirm      // accept winning bid / settle

/api/lp
  POST /deposit
  POST /withdraw
  GET  /auctions         // open auctions the LP is eligible for
  POST /auctions/:id/bid // submit a funded bid

/api/assets
  GET  /:id/score        // Score Reader passthrough
  GET  /:id/tier         // Risk Classifier result

/api/queue
  GET  /                 // FIFO queue status
  POST /:id/cancel       // reclaim escrowed RWA
```

7.2 Simplified Data Models

Exit Auction

```
{
  "id": "uuid",
  "sellerId": "uuid",
  "assetAddress": "CBEOJ....",
  "amount": 5000,
  "trufaScore": 82,
```

```
  "tier": "B",
  "discountRangeBps": [150, 400],
  "status": "open",          // open | settled | queued | cancelled
  "bestBidNetUsdc": 4820,
  "windowClosesAt": "2026-07-05T18:00:00Z"
}
```

LP Bid

```
{
  "id": "uuid",
  "auctionId": "uuid",
  "lpId": "uuid",
  "fundedUsdc": 4820,
  "discountBps": 360,
  "status": "winning"        // pending | winning | lost | settled
}
```

7.3 Contract Events (indexed)

The dashboard is built entirely off Soroban contract events; no privileged backend state is authoritative. Topics and payloads:

Topic	Payload	Emitted by
`exit.opened`	(auction_id, seller, asset, amount, tier)	Auction Engine
`exit.bid`	(auction_id, lp, funded_usdc, discount_bps)	Auction Engine
`exit.settled`	(auction_id, lp, usdc_to_seller)	Settlement Router
`exit.queued`	(auction_id, remainder)	Queue Manager
`exit.cancelled`	(auction_id, reason)	Auction Engine
`vault.deposit / vault.withdraw`	(lp, amount, shares)	Liquidity Vault
`vault.yield`	(strategy, delta_usdc)	Liquidity Vault

7.4 Configurable Parameters

- **AUCTION_WINDOW:** bidding window per exit (default 30 min).
- **STALENESS_SECS:** max validator-score age before exits are blocked (default 24h).
- **Tier params:** reserve ratio, discount band, and max exit size per tier (see §4.2).
- **protocol_fee_bps / swap_fee_bps:** protocol take on settled exits, ported from the EVM config.
- **hot_reserve_floor:** minimum USDC kept liquid in the vault before routing to DeFindex/Blend.

All parameters are governance-gated via a multisig admin, mirroring the emergency-pause and price-bound controls already present in the EVM contracts.

8. Risk Mitigation

Technical Risks:

- **Smart contract bugs** → port from an audited, test-covered EVM reference; single external audit before mainnet (outside budget)
- **Thin LP liquidity** → hot reserve + FIFO queue guarantee fair processing; Pooled LN seeds initial depth
- **Stale or gamed scores** → freshness guard in Score Reader; rejected/stale assets excluded from auctions
- **DeFindex/Blend integration risk** → idle-capital-only exposure with on-demand withdrawal; hot reserve never leaves the vault

Business Risks:

- **Exit price disputes** → transparent, LP-set, score-anchored pricing; never advertised as zero-slippage
- **Operator/asset quality** → Dob Validator gating; only approved assets receive liquidity
- **Regulatory** → KYC check on exit; anchor handles fiat compliance; legal review before launch

9. What Comes Next

After successfully delivering this stage:

Enhanced Features:

- Dynamic, data-driven discounting (on-chain + off-chain signals)
- Operator co-liquidity module (third exit layer) with matching incentives
- Secondary RWA order book for peer-to-peer resale at oracle price

Advanced Capabilities:

- Cross-chain exit liquidity bridging EVM and Stellar deployments
- Multiple anchors and multi-currency off-ramps across LATAM
- Institutional LP tranches with differentiated risk/return

Scale Operations:

- 50+ validated assets with continuous exit liquidity
- $10M+ in LP-backed exit capacity
- Autonomous LP-strategy agents managing liquidity via the DeFindex layer

10. Conclusion

DobDex delivers the one missing piece of the tokenized-RWA lifecycle on Stellar: a fair, transparent way for investors to exit before maturity. By pricing exits against Dob Validator's on-chain risk scores and settling through competing liquidity providers, DobDex gives

institutional capital the confidence to commit at scale — without pretending exits are risk-free or zero-slippage. The mechanism is already proven on EVM; this grant ports it to six native Soroban contracts and wires it into the live Stellar stack (Dob Validator, DeFindex, Blend, and anchors). The result is score-aware, LP-priced exit liquidity for validated Stellar RWAs — the infrastructure the ecosystem needs to close the loop between tokenization and real, liquid capital.

Annex

A. Complete Architecture Diagram

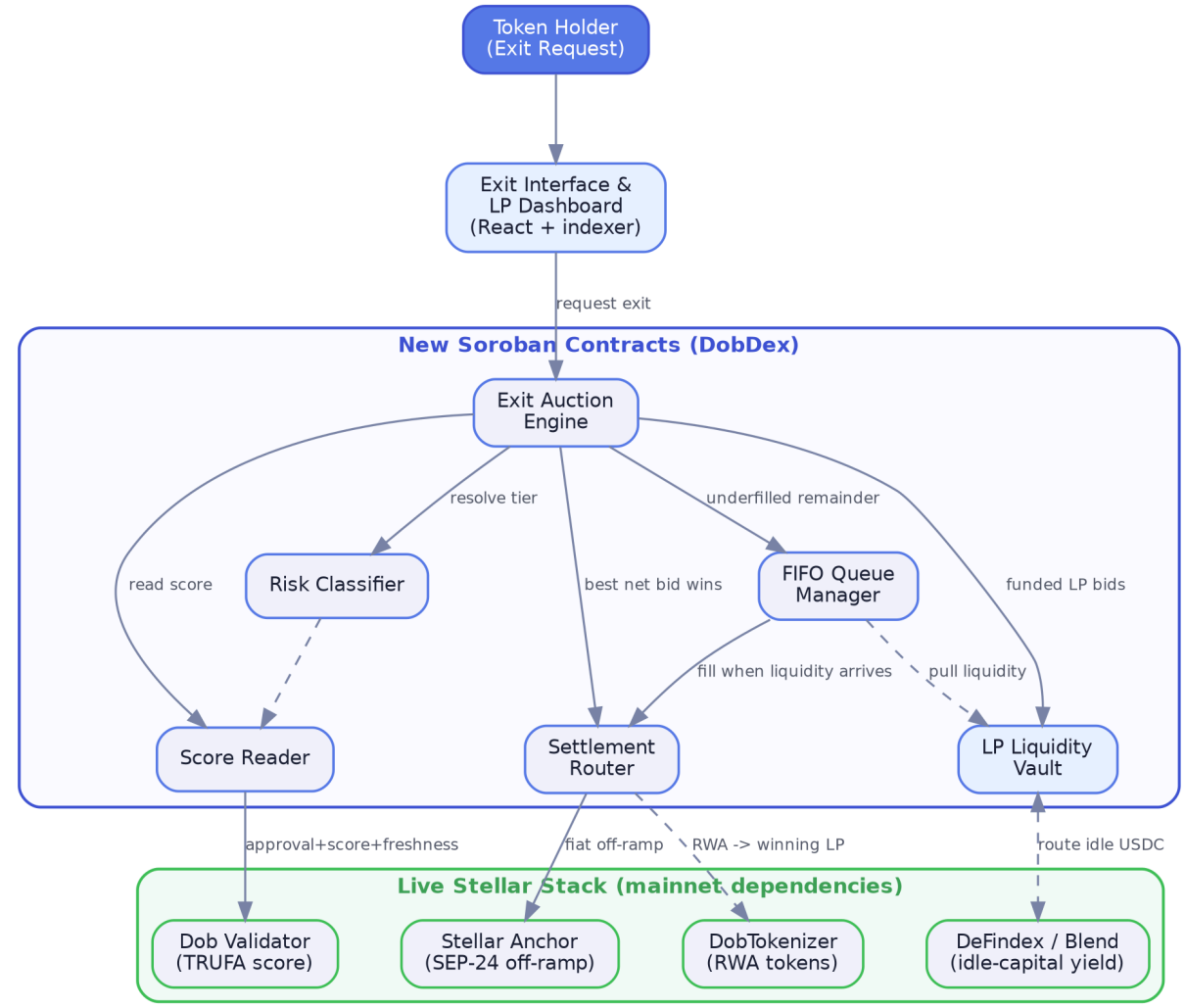

Fig A1. Full system architecture (see §3).

Fig A2. Exit lifecycle, eight stages (see §4).

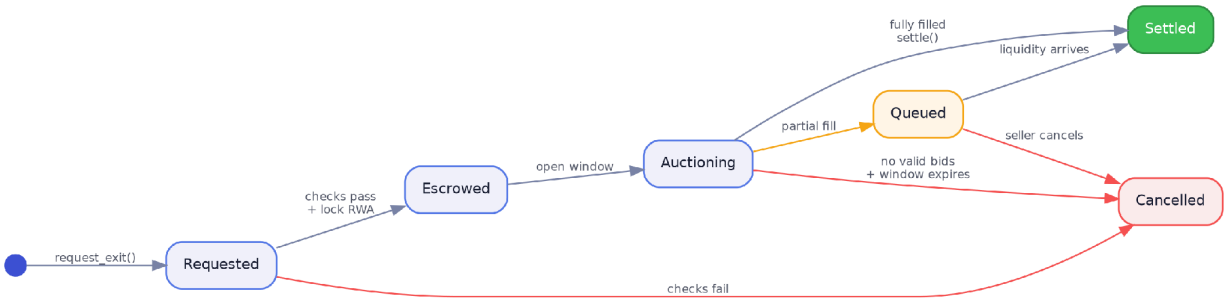

Fig A3. Exit auction state machine (see §4.4).

B. Team

- **Oscar Castillo, CEO.** Infrastructure partner onboarding, business architecture, and strategy.
- **Fernando Castillo, CTO.** PhD candidate, TU Berlin; designed and deployed every production Soroban contract in the Dob Protocol stack.
- **Simon Espinola, COO.** Go-to-market, partner onboarding, and the validation funnel.

Contact: simon@dobprotocol.com · www.dobprotocol.com