Data Models & Relationships

Overview

This document outlines the updated core data models in the L4VA system. Our revised architecture now supports dynamic vault management and a dual-level asset classification. This design not only covers standard use cases (e.g., works of art, real estate, gold bullion, and equity) but also enables the registration of custom vault and asset types as new real-world asset classes emerge.

Core Models

1. Vault

The primary entity representing a tokenized asset vault. The vault model now supports both fixed types (e.g., PUBLIC, PRIVATE, SEMI_PRIVATE) and dynamically registered types. It also allows for an open set of supported asset types, which can be expanded via a configuration registry or admin endpoint.

```
interface Vault {
  id: string;                  // Unique identifier
  contractAddress: string;       // On-chain vault contract address
  // VaultType can be one of the predefined values or a dynamically registered type.
  type: VaultType;               // e.g., PRIVATE | PUBLIC | SEMI_PRIVATE | custom types
  status: VaultStatus;           // DRAFT | ACTIVE | LOCKED | TERMINATED

  // Asset Configuration: supports dynamic asset types.
  // Defaults include NFT (for unique assets) and CNT (for fractional assets).
  assetTypes: AssetType[];
  assetWhitelist: string[];     // Allowed asset addresses
  contributorWhitelist: string[]; // Allowed contributor addresses

  // Windows Configuration
  assetWindow: {
    startTime: Date;
    duration: string;           // Format: "DD:HH:MM"
    status: WindowStatus;
  };

  investmentWindow: {
    startTime: Date;
    duration: string;           // Format: "DD:HH:MM"
    status: WindowStatus;
    valuationType: 'FIXED' | 'LBE';
  };
```

```
  // Fractionalization Settings
  fractionalization: {
    percentage: number;         // XX.XX%
    tokenSupply: number;
    tokenDecimals: number;      // 1-9
    tokenAddress: string;       // Fractional token contract address
  };

  // Investment Settings
  investment: {
    reserve: number;            // XX.XX%
    liquidityPool: number;      // XX.XX%
  };

  // Termination Settings
  termination: {
    type: 'DAO' | 'PROGRAMMED';
    fdp: number;                // Floor price deviation percentage
  };

  // Metadata: provides room for additional dynamic configuration.
  createdBy: string;            // Admin wallet address
  createdAt: Date;
  updatedAt: Date;
  metadata: Record<string, any>; // Additional configurable fields (e.g., custom vault parameters)
}

type WindowStatus = 'PENDING' | 'ACTIVE' | 'COMPLETED' | 'FAILED';
```

2. Asset

Represents assets held within vaults. This model introduces a dual-level classification: a primary tokenization method (e.g., NFT for unique items, CNT for fractional assets) and a secondary categorization (via metadata) to define asset-specific classes such as art, real estate, commodities, or equity.

```
interface Asset {
  id: string;
  vaultId: string;
  // AssetType may be one of the default types (e.g., NFT, CNT) or a dynamically registered type.
  type: AssetType;
```

```
  contractAddress: string;
  tokenId?: string;             // For NFTs
  quantity: number;             // For CNTs

  // Valuation
  floorPrice: number;           // For NFTs
  dexPrice: number;             // For CNTs
  lastValuation: Date;

  // Status
  status: AssetStatus;          // PENDING | LOCKED | RELEASED
  lockedAt?: Date;
  releasedAt?: Date;

  // Metadata now includes an additional "category" field.
  // This field can capture asset-specific classifications (e.g., art, real_estate, commodity, equity)
  metadata: {
    name: string;
    description: string;
    imageUrl: string;
    category?: string;
    attributes: Record<string, any>;
  };

  // Tracking
  addedBy: string;              // Wallet address
  addedAt: Date;
  updatedAt: Date;
}

type AssetStatus = 'PENDING' | 'LOCKED' | 'RELEASED';
```

3. Proposal

Governance proposals remain largely unchanged, but note that proposals may reference dynamically registered asset types or custom governance rules based on asset categories.

```
interface Proposal {
  id: string;
  vaultId: string;
  type: ProposalType;         // ASSET_SALE | BUY | STAKE | LIQUIDATE
```

```
// Phases
votingPhase: {
  duration: string;        // Format: "DD:HH:MM"
  startTime: Date;
  endTime: Date;
  status: PhaseStatus;
  quorum: number;          // Required participation percentage
};

lockPhase: {
  duration: string;
  startTime: Date;
  endTime: Date;
  status: PhaseStatus;
};

executionPhase: {
  duration: string;
  startTime: Date;
  endTime: Date;
  status: PhaseStatus;
  requiredCosigners: number;
};

// Proposal Details: can target assets of any supported type.
settings: {
  assets: string[];        // Affected asset IDs
  effects: ProposalEffect[];
};

// Results
votes: {
  approve: number;
  reject: number;
  totalVoted: number;
  uniqueVoters: number;
};

// Status
```

```
  status: ProposalStatus;    // DRAFT | ACTIVE | PASSED | FAILED | EXECUTED
  result?: ProposalResult;

  // Metadata
  createdBy: string;         // Proposer wallet address
  createdAt: Date;
  updatedAt: Date;
}

type PhaseStatus = 'PENDING' | 'ACTIVE' | 'COMPLETED' | 'FAILED';
type ProposalStatus = 'DRAFT' | 'ACTIVE' | 'PASSED' | 'FAILED' | 'EXECUTED';
```

4. Vote

Individual votes on proposals used to execute governance decisions. This model remains unchanged.

```
interface Vote {
  id: string;
  proposalId: string;
  wallet: string;          // Voter's wallet address
  decision: 'APPROVE' | 'REJECT';
  amount: number;          // Amount of FTs staked in vote

  // Status
  status: VoteStatus;      // CAST | CONFIRMED | REVOKED
  confirmationTx?: string; // Blockchain transaction hash

  // Timing
  castAt: Date;
  confirmedAt?: Date;
  revokedAt?: Date;
}
```

5. Stake

Represents staked fractional tokens that contribute to voting power. This model remains similar but supports dynamic governance processes tied to various asset classes.

```
interface Stake {
  id: string;
  vaultId: string;
  wallet: string;          // Staker's wallet address
```

```
  amount: number;         // Staked amount

  // Status
  status: StakeStatus;    // ACTIVE | LOCKED | RELEASED
  lockedUntil?: Date;     // For time-locked stakes

  // Voting Power
  votingPower: number;
  usedPower: number;      // Power used in active votes

  // Tracking
  createdAt: Date;
  updatedAt: Date;
  transactions: {
    stakeHash: string;
    unstakeHash?: string;
  };
}
```

Relationships

Entity Relationships Diagram

```
  Vault ||--o{ Asset : contains
  Vault ||--o{ Proposal : has
  Vault ||--o{ Stake : holds
  Proposal ||--o{ Vote : receives
  Stake ||--o{ Vote : powers
```

Key Relationships

1. **Vault ? Assets**
   - One-to-many relationship
   - A vault can contain multiple assets; each asset belongs to a single vault.
   - Relationship constraints are enforced by assetTypes and assetWhitelist.
2. **Vault ? Proposals**
   - One-to-many relationship
   - Proposals are tied to a single vault and may leverage asset category-specific rules.
3. **Proposal ? Votes**
   - One-to-many relationship

- Votes are associated with specific proposals, with vote weight determined by staked tokens.

4. **Stake ? Votes**
   - One-to-many relationship
   - Stakes power votes and help track voting power relative to the amount staked.

Validation Rules

1. Vault Validation

```
const vaultValidation = {
  assetWindow: {
    minDuration: "01:00:00",
    maxDuration: "30:00:00"
  },
  investmentWindow: {
    minDuration: "24:00:00",
    maxDuration: "168:00:00"
  },
  fractionalization: {
    minPercentage: 1,
    maxPercentage: 100,
    minDecimals: 1,
    maxDecimals: 9
  },
  investment: {
    minReserve: 5,
    maxReserve: 50,
    minLiquidity: 1,
    maxLiquidity: 20
  }
};
```

2. Asset Validation

```
const assetValidation = {
  nft: {
    maxPerVault: 100
  },
  cnt: {
    minQuantity: 1,
```

```
    maxQuantity: 1000000
  },
  valuation: {
    maxAge: "24:00:00"
  }
};
```

3. Proposal Validation

```
const proposalValidation = {
  voting: {
    minDuration: "24:00:00",
    maxDuration: "168:00:00",
    minQuorum: 10,
    maxQuorum: 100
  },
  lock: {
    minDuration: "12:00:00",
    maxDuration: "48:00:00"
  },
  execution: {
    minDuration: "24:00:00",
    maxDuration: "72:00:00"
  }
};
```

Implementation Guidelines

1. Database Indexes

```
// Vault Indexes
{
  "contractAddress": 1,
  "status": 1,
  "type": 1,
  "createdAt": -1
}

// Asset Indexes
{
```

```
  "vaultId": 1,
  "contractAddress": 1,
  "status": 1
}

// Proposal Indexes
{
  "vaultId": 1,
  "status": 1,
  "votingPhase.endTime": 1
}

// Vote Indexes
{
  "proposalId": 1,
  "wallet": 1,
  "status": 1
}

// Stake Indexes
{
  "vaultId": 1,
  "wallet": 1,
  "status": 1
}
```

2. Cascade Behaviors

```
// Delete cascade rules
const cascadeRules = {
  vault: {
    onDelete: ['assets', 'proposals', 'stakes']
  },
  proposal: {
    onDelete: ['votes']
  }
};
```

3. Data Integrity

```
interface DataIntegrityChecks {
  validateStakeBalance: () => Promise<void>;
  validateVotingPower: () => Promise<void>;
  validateProposalStatus: () => Promise<void>;
  validateAssetLocks: () => Promise<void>;
}
```

Best Practices

1. Data Access

- Use the repositories pattern.
- Implement a caching strategy.
- Handle concurrent updates.
- Maintain audit logs.

2. Validation

- Validate at the model level.
- Enforce business rules.
- Check relationships.
- Verify blockchain state.

3. Performance

- Use appropriate indexes.
- Implement pagination.
- Optimize queries.
- Monitor database load.

4. Security

- Encrypt sensitive data.
- Validate permissions.
- Audit data access.
- Handle PII properly.

Note: I recently updated this document to reflect a move toward a dynamic, extensible framework as detailed in our Vault Formation and Data Models & Relationships guidelines, ensuring our platform remains agile in representing any real-world asset.

Revision #4
Created 26 November 2024 01:21:43 by Aric Fedida
Updated 18 February 2025 06:04:14 by Aric Fedida