Smart Contracts

Purpose: Technical specification and documentation of all blockchain-related smart contracts and their interactions.

Key Contents:

- Smart contract architecture and design patterns
- Detailed function specifications and parameters
- Contract deployment procedures and addresses
- Security considerations and audit findings
- Gas optimization strategies
- Contract upgradeability design
- Cross-contract interaction patterns

Contract Architecture & Patterns

Technical Implementation Guide

Overview

This documentation outlines the implementation of a real-world asset tokenization system on Cardano, utilizing the Agora protocol for governance. The system enables the creation of fractionalized ownership of physical assets through secure smart contracts, with built-in governance mechanisms for collective decision-making.

The implementation combines vault management for asset handling, Agora's three-phase governance system, token policies for both fractional ownership and governance rights, and managed contribution windows. All components work together to create a secure, compliant, and flexible tokenization platform.

File Structure

```
plutus/
├── core/
│   ├── VaultFactory.hs       # Creates vault instances
│   ├── VaultRegistry.hs      # Global vault tracking
│   ├── FractionalizedVault.hs # Core vault functionality
│   └── VaultValidator.hs     # Main validation logic
├── agora/
│   ├── ProposalValidator.hs   # Proposal validation logic
│   ├── LockPhase.hs          # Lock phase implementation
│   ├── VotingPhase.hs        # Voting phase logic
│   ├── ExecutionPhase.hs     # Execution phase handling
│   └── ThresholdValidator.hs  # Governance thresholds
├── tokens/
│   ├── FractionalPolicy.hs    # Fractional token minting
│   └── GovernancePolicy.hs    # Agora governance token
├── windows/
│   ├── AssetWindowValidator.hs     # Asset contribution period
│   └── InvestmentWindowValidator.hs # Investment period
├── types/
│   ├── VaultTypes.hs         # Core type definitions
│   ├── AgoraTypes.hs         # Agora governance types
│   └── WindowTypes.hs        # Window related types
├── effects/
│   ├── ProposalEffects.hs     # Proposal action execution
│   └── RequirementEffects.hs  # Requirement modifications
└── utils/
```

```
    ├── Validators.hs         # Common validation functions
    └── Scripts.hs            # Script utilities
```

Core Vault Components

The vault system manages the fundamental asset tokenization functionality.

Vault Factory

Responsible for creating new vault instances with proper initialization:

```
data VaultParams = VaultParams {
    vaultType :: VaultType,
    assetTypes :: [AssetType],
    settings :: VaultSettings
}

newtype VaultFactory = VaultFactory {
    createVault :: VaultParams -> Contract w s Text VaultId
}

mkVaultValidator :: VaultParams -> TypedValidator VaultSchema
mkVaultValidator params = mkTypedValidator @VaultSchema
    ($$(PlutusTx.compile [|| validateVault ||])
        `PlutusTx.applyCode` PlutusTx.liftCode params)
    $$(PlutusTx.compile [|| wrap ||])
  where
    wrap = wrapValidator @VaultDatum @VaultRedeemer
```

Vault Registry

Maintains the global state of all vaults:

```
data RegistryDatum = RegistryDatum {
    vaults :: Map VaultId VaultInfo,
    totalVaults :: Integer
}

data VaultInfo = VaultInfo {
    vaultState :: VaultState,
    assetIds :: [AssetId],
    tokenPolicy :: CurrencySymbol
}
```

Fractionalized Vault

Manages the core vault operations and state:

```
data VaultDatum = VaultDatum {
   assets :: [AssetDetails],
   fractionalization :: FractionalizationParams,
   state :: VaultState
}

data AssetDetails = AssetDetails {
   assetId :: AssetId,
   amount :: Integer,
   locked :: Bool
}
```

Agora Governance Integration

The Agora protocol provides a structured governance system with three distinct phases.

Core Governance Structures

```
data GovernanceSettings = GovernanceSettings {
   creationThreshold :: Percentage,    -- Min FT % to create proposal
   startThreshold :: Percentage,       -- Min FT % to start voting
   voteThreshold :: Percentage,        -- Min staked FT for valid vote
   executionThreshold :: Percentage,   -- Min votes for execution
   cosigningThreshold :: Percentage,   -- Min FT for cosigning
   lockDuration :: POSIXTime           -- Lock phase duration
}

data ProposalPhase =
   LockPhase
 | VotingPhase
 | ExecutionPhase
```

Proposal Management

```
data ProposalParams = ProposalParams {
    description :: Text,
    votingDuration :: POSIXTime,
    lockDuration :: POSIXTime,
    executionDuration :: POSIXTime,
    requiredCosigners :: [PubKeyHash],
    effects :: [ProposalEffect]
}

createProposal :: VaultId -> ProposalParams -> Contract w s Text ProposalId
createProposal vaultId params = do
    validateCreatorStake
    proposalId <- submitTx $ mustPayToScript proposalValidator (proposalDatum params)
    emitEvent $ ProposalCreated proposalId
    pure proposalId
```

Phase Transitions

```
data PhaseTransition =
    StartVoting
  | StartLockPhase
  | StartExecution

validatePhaseTransition :: ProposalDatum -> PhaseTransition -> ScriptContext -> Bool
validatePhaseTransition datum transition ctx = case transition of
    StartVoting ->
        meetStartThreshold &&
        timeToVote
    StartLockPhase ->
        votingComplete &&
        sufficientVotes
    StartExecution ->
        lockPhaseComplete &&
        cosignersSigned
```

Token Management

Governance Token

```
data GovernanceToken = GovernanceToken {
    policyId :: CurrencySymbol,
    tokenName :: TokenName,
    totalSupply :: Integer
}

validateGovernanceToken :: AssetParams -> GovernanceToken -> ScriptContext -> Bool
validateGovernanceToken params token ctx =
    correctSupply &&
    correctDistribution &&
    hasGovernanceMetadata
```

Fractional Token

```
data FractionalTokenParams = FractionalTokenParams {
    tokenName :: TokenName,
    decimals :: Integer,
    totalSupply :: Integer
}

mkTokenPolicy :: FractionalTokenParams -> MintingPolicy
mkTokenPolicy params = mkMintingPolicyScript
    ($$(PlutusTx.compile [|| validateMinting ||])
        `PlutusTx.applyCode` PlutusTx.liftCode params)
```

Window Management

Asset Window

```
data WindowDatum = WindowDatum {
    windowStart :: POSIXTime,
    windowEnd :: POSIXTime,
    contributions :: Map AssetId Contribution
}

data WindowRedeemer =
    Contribute AssetContribution |
    CloseWindow
```

Types System

Vault Types

```
data VaultState =
    Draft
  | Active
  | Locked
  | Terminated

data VaultAction =
    Initialize VaultParams
  | AddAsset AssetDetails
  | RemoveAsset AssetId
  | LockAssets
  | UnlockAssets
```

Governance Types

```
data ProposalState =
    Pending
  | VotingActive
  | Locked
  | Executed
  | Failed

data ProposalAction =
    CreateProposal ProposalParams
  | CastVote VoteDetails
  | ExecuteProposal
  | CancelProposal
```

Implementation Notes

Best Practices

The implementation adheres to several key principles:

- Strong typing ensures that invalid states are unrepresentable in the type system. Every operation has explicit type definitions that capture its requirements and constraints.
- Explicit datum and redeemer structures clearly define the state transitions and valid operations. Each validator precisely specifies what constitutes a valid state change.
- The UTXO model is used efficiently, with careful consideration given to datum design and state management. This helps minimize resource usage and transaction costs.

- Validator patterns follow established security practices, with comprehensive checks and clear error messages. Multiple validation layers ensure system integrity.

Security Considerations

The implementation includes several security measures:

1. Multiple validation layers verify all operations
2. Strong typing prevents invalid state transitions
3. Explicit access control through stake-based governance
4. Comprehensive audit trail of all operations
5. Time-locked phases prevent rushed decisions

Integration Points

The system's main integration points are:

1. Asset registration and verification
2. Governance token distribution
3. Proposal creation and execution
4. Time Window management and transitions
5. Effect implementation and validation

Some useful implementation examples

Table of Contents

1. Core Implementation

Base Structure

```
data RealWorldAsset = RealWorldAsset {
    assetId :: AssetId,
    assetType :: AssetType,
    assetValue :: Integer,
    assetMetadata :: AssetMetadata,
    custodialInfo :: CustodianInfo,
    verificationData :: VerificationInfo,
    requirementSet :: [Requirement]
}

-- Extensible asset types to accommodate any real-world asset
data AssetType =
    ArtPiece ArtDetails
  | RealEstate PropertyDetails
  | PreciousMetals MetalDetails
  | Commodity CommodityDetails
  | CustomAsset CustomDetails

-- Requirement system that handles any type of requirement
data Requirement = Requirement {
    requirementType :: RequirementType,
    description :: Text,
    validationMethod :: ValidationMethod,
    verificationFrequency :: Frequency,
    requirementStatus :: RequirementStatus,
    lastVerified :: Maybe POSIXTime
}

data RequirementType =
    Legal
  | Physical
  | Operational
  | Environmental
  | Financial
  | Custom Text
```

Agora Governance Integration

```
data AgoraPhase =
    LockPhase LockPhaseInfo
  | VotingPhase VotingPhaseInfo
  | ExecutionPhase ExecutionPhaseInfo

data GovernanceConfig = GovernanceConfig {
    thresholds :: AgoraThresholds,
    phaseDurations :: PhaseDurations,
    votingPowerStrategy :: VotingStrategy,
    effectValidators :: Map EffectType EffectValidator
}

data AgoraThresholds = AgoraThresholds {
    creation :: Percentage,    -- e.g., 5%
    start :: Percentage,       -- e.g., 10%
    vote :: Percentage,        -- e.g., 15%
    execution :: Percentage,   -- e.g., 51%
    cosigning :: Percentage    -- e.g., 25%
}

data PhaseDurations = PhaseDurations {
    lockPhaseDuration :: POSIXTime,
    votingPhaseDuration :: POSIXTime,
    executionPhaseDuration :: POSIXTime
}
```

2. Asset Journey Examples

Example 1: Fine Art Tokenization

```
-- Picasso painting tokenization example
artExample :: RealWorldAsset
artExample = RealWorldAsset {
    assetId = "art_001",
    assetType = ArtPiece {
        artist = "Pablo Picasso",
        title = "Example Painting",
        year = 1937,
        medium = "Oil on canvas",
        dimensions = Dimensions 349.3 776.6,
        authenticity = [Certificate1, Certificate2]
    },
    assetValue = 10_000_000_000_000,  -- 10M ADA
    requirementSet = [
        Requirement {
            requirementType = Physical,
            description = "Temperature control 20-22°C",
            validationMethod = TemperatureSensorValidation,
            verificationFrequency = Hourly,
            requirementStatus = Active
        },
        Requirement {
            requirementType = Legal,
            description = "Insurance coverage",
            validationMethod = InsuranceDocValidation,
            verificationFrequency = Monthly,
            requirementStatus = Active
        }
    ]
}

-- Art-specific proposal example
proposeMoveToNewGallery :: ProposalParams
proposeMoveToNewGallery = ProposalParams {
    description = "Move artwork to Modern Gallery",
    effect = PhysicalLocationChange {
        newLocation = "Modern Gallery, NY",
        transportMethod = "Specialized Art Transport",
        insuranceCoverage = "Extended Transit Insurance"
    },
    requiredCosigners = [
        galleryCurator,
        insuranceProvider,
        securityProvider
    ]
}
```

Example 2: Commodity Batch Management

```
-- Coffee batch tokenization example
commodityExample :: RealWorldAsset
commodityExample = RealWorldAsset {
    assetId = "coffee_batch_001",
    assetType = Commodity {
        type = "Arabica Coffee",
        grade = "Premium",
        origin = "Colombia",
        harvest = "2024",
        quantity = MetricTons 100
    },
    assetValue = 1_000_000_000_000,  -- 1M ADA
    requirementSet = [
        Requirement {
            requirementType = Environmental,
            description = "Storage humidity 60-65%",
            validationMethod = HumiditySensorValidation,
            verificationFrequency = Daily,
            requirementStatus = Active
        }
    ]
}

-- Storage condition change proposal
proposeStorageChange :: ProposalParams
proposeStorageChange = ProposalParams {
   description = "Update storage conditions",
   effect = StorageConditionChange {
      newHumidity = Percentage 62,
      newTemperature = Celsius 20,
      implementation = "Automated climate control"
   },
   requiredCosigners = [
      warehouseManager,
      qualityInspector
   ]
}
```

3. Phase Transition Scenarios

Lock Phase Implementation

```
data LockPhaseInfo = LockPhaseInfo {
   startTime :: POSIXTime,
   endTime :: POSIXTime,
   requiredStake :: Integer,
   currentStake :: Integer,
   lockedTokens :: Map PubKeyHash Integer
}

startLockPhase :: ProposalId -> Contract w s Text ()
startLockPhase proposalId = do
   proposal <- getProposal proposalId
   currentTime <- getCurrentTime

   let lockInfo = LockPhaseInfo {
         startTime = currentTime,
         endTime = currentTime + proposal.lockDuration,
         requiredStake = calculateRequiredStake proposal,
         currentStake = 0,
         lockedTokens = Map.empty
      }

   validateLockPhaseStart proposal
   updateProposalPhase proposalId (LockPhase lockInfo)
```

```
    emitLockPhaseStarted proposalId

validateLockPhaseStart :: Proposal -> Bool
validateLockPhaseStart proposal =
    hasMinimumCreationStake &&
    notInActivePhase &&
    allRequirementsValid
```

Voting Phase Implementation

```
data VotingPhaseInfo = VotingPhaseInfo {
    votes :: Map PubKeyHash Vote,
    totalVotingPower :: Integer,
    usedVotingPower :: Integer,
    voteDistribution :: VoteDistribution
}

data Vote = Vote {
    direction :: VoteDirection,
    power :: Integer,
    timestamp :: POSIXTime,
    metadata :: VoteMetadata
}

startVotingPhase :: ProposalId -> Contract w s Text ()
startVotingPhase proposalId = do
    proposal <- getProposal proposalId

    -- Verify lock phase completion
    unless (isLockPhaseComplete proposal) $
        throwError "Lock phase incomplete"

    -- Verify start threshold
    unless (meetsStartThreshold proposal) $
        throwError "Insufficient stake for voting start"

    let votingInfo = VotingPhaseInfo {
            votes = Map.empty,
            totalVotingPower = calculateTotalPower proposal,
            usedVotingPower = 0,
            voteDistribution = initializeVoteDistribution
        }

    updateProposalPhase proposalId (VotingPhase votingInfo)
    emitVotingPhaseStarted proposalId
```

4. Threshold Calculations

Practical Examples

```
-- Example: Art piece worth 10M ADA
calculateThresholds :: AssetValue -> AgoraThresholds
calculateThresholds assetValue = AgoraThresholds {
  creation = Percentage 5,    -- 500k ADA to create proposal
  start = Percentage 10,      -- 1M ADA to start voting
  vote = Percentage 15,       -- 1.5M ADA for valid vote
  execution = Percentage 51,  -- 5.1M ADA to execute
  cosigning = Percentage 25   -- 2.5M ADA for cosigning
}

-- Example: Calculating voting power
calculateVotingPower :: TokenAmount -> VotingStrategy -> Integer
calculateVotingPower amount strategy = case strategy of
  Linear ->
    amount
  Quadratic ->
    floor $ sqrt $ fromIntegral amount
  WeightedByTime holdingTime ->
    amount * (1 + (holdingTime `div` 30))  -- Bonus for longer holds
```

5. Governance Flows

Complete Proposal Lifecycle

```
data ProposalLifecycle = ProposalLifecycle {
  proposal :: Proposal,
  currentPhase :: AgoraPhase,
  history :: [PhaseTransition],
  votes :: Map PubKeyHash Vote,
  effects :: [Effect],
  status :: ProposalStatus
}

executeProposalLifecycle :: ProposalId -> Contract w s Text ()
executeProposalLifecycle proposalId = do
  -- Initialize proposal
  proposal <- createProposal proposalId

  -- Lock phase
  startLockPhase proposalId
  awaitLockPhaseCompletion proposalId

  -- Voting phase
  startVotingPhase proposalId
  collectVotes proposalId

  -- Execution phase
  validateVotingResults proposalId
  collectCosignatures proposalId
  executeEffects proposalId
```

6. Implementation Considerations

Security Measures

```
data SecurityMeasures = SecurityMeasures {
    accessControl :: AccessControl,
    thresholdValidation :: ThresholdValidation,
    timelock :: TimelockConfig,
    emergencyProcedures :: EmergencyProcedures
}

validateSecurityMeasures :: SecurityMeasures -> ScriptContext -> Bool
validateSecurityMeasures measures ctx =
    validateAccess measures.accessControl ctx &&
    validateThresholds measures.thresholdValidation ctx &&
    validateTimelock measures.timelock ctx
```

Error Handling

```
data GovernanceError =
    InsufficientStake Text
  | InvalidPhaseTransition Text
  | ThresholdNotMet Text
  | ValidationFailed Text
  | TimelockNotExpired Text

handleGovernanceError :: GovernanceError -> Contract w s Text a
handleGovernanceError = \case
    InsufficientStake msg ->
        logError $ "Stake requirement not met: " <> msg
    InvalidPhaseTransition msg ->
        logError $ "Invalid phase transition: " <> msg
    ThresholdNotMet msg ->
        logError $ "Threshold not met: " <> msg
    ValidationFailed msg ->
        logError $ "Validation failed: " <> msg
    TimelockNotExpired msg ->
        logError $ "Timelock not expired: " <> msg
```

This implementation guide demonstrates how the Agora governance protocol can be used effectively with various real-world assets, providing both technical implementation details and practical examples that engineers can follow.

Contract Interfaces & Functions

Security & Audits

Gas & Optimization

Deployment & Upgradeability

Cross-Contract Integration

Testing & Verification

Story: Commercial Real Estate RWA

Real World Assets (RWA) on Cardano Blockchain with L4VA

This document outlines how L4VA enables the tokenization, aggregation, and fractionalization of commercial real estate assets on the Cardano blockchain, along with the specific transaction types and smart contracts needed for each step of implementation.

The Story

Introduction

A commercial real estate investment firm, RealX Holdings, identified an opportunity to increase liquidity and accessibility in the traditionally illiquid commercial real estate market. They decided to use L4VA to tokenize their portfolio of commercial properties and create fractionalized ownership tokens that could be traded on the blockchain.

Holding Company Tokenization

RealX Holdings had several LLCs, each holding different commercial properties (office buildings, retail spaces, and industrial warehouses). They tokenized these holding companies by issuing share classes representing percentage ownership of each LLC. These share classes were then minted into NFTs on the Cardano blockchain.

Vault Creation

Using L4VA, RealX created specialized vaults for different asset categories:

1. **Office Space Vault**: Aggregating tokens from office building LLCs
2. **Retail Space Vault**: Aggregating tokens from retail property LLCs
3. **Industrial Space Vault**: Aggregating tokens from industrial warehouse LLCs

Each vault was designed to manage the aggregated tokens and enable fractionalized ownership through fungible tokens (FTs).

Investment Round

Investors were able to purchase fractionalized tokens from any of the vaults, gaining exposure to specific commercial real estate sectors without needing to buy entire properties. Each investor could contribute any amount of ADA, making commercial real estate investment accessible to a much wider audience.

Diversified Portfolio Creation

A financial influencer named Charlie saw an opportunity to create a more diversified commercial real estate portfolio. Charlie created a new vault on L4VA that aggregated tokens from all three sector-specific vaults (Office, Retail, and Industrial) into a "Commercial Real Estate Index" vault. This vault issued its own tokens representing exposure to the entire commercial real estate market, reducing risk through diversification.

Governance Decisions

Each vault implemented strictly defined governance systems allowing token holders to vote on a specific set of permitted actions:

Permitted Governance Actions:

1. **Asset Sales:** Proposals to list or sell vault assets at market price to generate liquidity or rebalance the portfolio.
2. **Asset Acquisition:** Proposals to purchase new assets that match the vault's whitelist using available ADA in the vault.
3. **Asset Staking:** Proposals to stake assets to generate additional yield for the vault.
4. **Distribution to Token Holders:** Proposals to distribute fungible token assets (including ADA) to fractional token holders through an asset claim and token burn process.
5. **Vault Mergers (v2):** Proposals to merge compatible vaults, requiring approval from both vaults' token holders, creating a new vault with combined assets and a whitelist restricted to the sum of the original whitelists.

Prohibited Governance Actions: To prevent potential fraud and maintain vault integrity, certain operations were explicitly prohibited by the smart contract:

1. **Direct External Transfers:** The governance system prevented proposals to send ADA or fungible token assets from the vault to specific external addresses.
2. **Whitelist Expansion:** Vaults could not expand their asset whitelist beyond their initial scope without a full protocol upgrade.
3. **Non-Market-Based Transactions:** All asset sales and purchases required verifiable market price validation.

This strictly defined governance system ensured that vault operations remained secure, transparent, and aligned with token holders' interests while preventing potential abuse.

Revenue Distribution

As the commercial properties generated rental income, this revenue flowed into the respective vaults. The governance system automatically distributed these returns to token holders proportional to their ownership, creating a steady income stream for investors while maintaining complete transparency.

Blockchain Implementation: Transaction Types and Smart Contracts

1. LLC Share Tokenization

Transaction Type: Minting Transaction

- Creates NFTs representing LLC share classes
- Includes metadata about the commercial property details (location, size, income, valuation)

Smart Contract Required: `AssetTokenizationContract` (Contract Hash: `479f356943df735488e8f6ce7dd7dd9e757b68b9e01175da42031111`)

- Mints NFTs with property metadata
- Links legal documents to the asset (property deed, LLC operating agreement)
- Creates verifiable link between LLC shares and digital tokens

Implementation:

```
# Required functions from lib-assets.ts
createAsset("commercial_property", ["operating_agreement.pdf", "valuation_report.pdf", "title_deed.pdf"])
```

2. Vault Creation

Transaction Type: Minting Transaction

- Creates new tokens representing specialized vaults (Office, Retail, Industrial)
- Establishes governance parameters and fee structures

Smart Contract Required: `VaultContract` (Contract Hash: `ac2bb90a5a34ee1a7fe5340b73932132df67afb54d90605be6a8329f`)

- Creates vault structure with specified parameters
- Sets up token acceptance policies (which asset types can be deposited)
- Establishes governance rules
- Manages fractionalization mechanisms

Implementation:

```
deno run --env-file -A create_vault.ts
```

3. Asset Aggregation

Transaction Type: Transfer Transaction

- Transfers LLC NFTs into appropriate sector vaults
- Records ownership provenance

Smart Contract Required: `AssetAggregationContract` (utilizing `VaultContract`)

- Validates asset type matches vault criteria
- Records contribution details and ownership transfer
- Updates vault composition

Implementation:

```
# Uses getVaultUtxo function from lib.ts
getVaultUtxo(vaultPolicyId, vaultAssetName)
```

4. Fractionalization

Transaction Type: Minting Transaction

- Mints fungible tokens (FTs) representing fractional ownership of the vault
- Associates FT supply with vault asset value

Smart Contract Required: `FractionalizationContract` (part of `VaultContract`)

- Calculates appropriate token supply based on asset values
- Enforces proportional ownership rules
- Manages token supply updates when assets are added/removed

Implementation:

```
# Function from lib.ts to calculate token distributions
assetsToValue(vaultAssets)
```

5. Diversified Portfolio Creation

Transaction Type: Vault Creation + Transfer Transactions

- Creates a new vault that accepts tokens from other vaults
- Transfers tokens from sector-specific vaults to the diversified vault

Smart Contract Required: `MetaVaultContract` (extending `VaultContract`)

- Handles vault-of-vaults structure
- Manages nested governance rules
- Calculates appropriate index token supply based on underlying vault tokens

Implementation:

```
# Uses multiple functions from lib.ts
generate_assetname_from_txhash_index(metavaultTxHash, outputIndex)
```

6. Governance Voting

Transaction Type: Voting Transactions

- Records votes from token holders
- Enforces voting weight based on token ownership
- Executes approved actions within strict constraints

Smart Contract Required: `GovernanceContract` (part of `VaultContract`)

- Validates voter eligibility based on token ownership
- Enforces voting timeframes
- Requires minimum participation thresholds
- Validates that proposals match permitted operations
- Rejects prohibited operations (e.g., direct transfers to external addresses)
- Executes approved decisions automatically

Implementation:

```
# Function from lib.ts to track governance proposals
getUtxos(governanceAddress, minimumStake)
```

7. Asset Distribution

Transaction Type: Distribution Transactions

- Creates asset claims for token holders
- Requires token burn to claim assets
- Distributes assets proportionally to token ownership

Smart Contract Required: `DistributionContract` (part of `VaultContract`)

- Calculates distributions based on token ownership percentages
- Creates secure claim mechanism requiring token burn
- Validates distribution timeframes
- Executes automatic distributions to token holders
- Updates distribution records

Implementation:

```
# Asset claim and token burn process
createAssetClaim(vaultId, assetId, distributionAmount) +
burnTokenForClaim(fractionalTokenId, claimId)
```

8. Vault Merging (v2)

Transaction Type: Complex Transaction

- Requires approval from both vault governance systems
- Creates new vault with combined assets
- Burns original vault tokens
- Mints new vault tokens for holders from both original vaults

Smart Contract Required: `VaultMergerContract` (extending `VaultContract`)

- Handles complex multi-vault governance voting
- Creates new vault with properly restricted whitelist
- Manages asset transfers from original vaults
- Calculates fair token distribution in new vault
- Ensures continuity of ownership rights

Implementation:

```
# Implementation for vault merging
proposeVaultMerger(sourceVaultId, targetVaultId, newParams) +
approveVaultMerger(sourceVaultId, targetVaultId, approvalSignature) +
executeVaultMerger(sourceVaultId, targetVaultId, newVaultId)
```

9. DeFi Integration

Transaction Type: Collateralization Transactions

- Uses vault tokens as collateral for lending platforms
- Enables yield farming with vault tokens

Smart Contract Required: `DeFiIntegrationContract`

- Creates compatible interfaces with other DeFi protocols
- Manages collateralization ratios and liquidation parameters
- Enables automated yield strategies

Implementation:

```
# Integration with external DeFi protocols
getUtxos(userAddress) + collateralizeAssets(vaultTokens, loanAmount)
```

L4VA System Architecture Diagram

```
+---------------------------+    +--------------------------+    +-------------------------+
|                           |    |                          |    |                         |
| LLC Holdings (Real World) |    | Blockchain Tokenization  |    | L4VA Vault System       |
|                           |    |                          |    |                         |
+---------------------------+    +--------------------------+    +-------------------------+
            |                               |                              |
            v                               v                              v
+---------------------------+    +---------------------------+    +-------------------------+
| - Office Building LLC     |    | - Office Building NFT     |    | - Office Space Vault    |
| - Retail Space LLC        | => | - Retail Space NFT        | => | - Retail Space Vault    |
| - Industrial Warehouse LLC|    | - Industrial Warehouse NFT|    | - Industrial Space Vault|
+---------------------------+    +---------------------------+    +-------------------------+
                                                                           |
                                                                           v
                                                              +-------------------------+
                                                              |                         |
                                                              | Diversified Meta-Vault  |
                                                              |                         |
                                   +------------------------+
                                              |
                  Investors                   v
                     |             +-------------------------+
                     v             |                         |
+--------------------------+  +-------------------------+  | - Fractional Ownership  |
|                   |   |                    |   | - Governance Rights     |
| DeFi Applications      | <= | Fractionalized Tokens    | <= | - Automated Distributions|
|                   |   |                    |   | - Index Exposure        |
+--------------------------+  +-------------------------+  |                         |
                                   +-------------------------+
```

Transaction Types and Flow

Step	Transaction Type	Asset Involved	Smart Contract	Purpose
LLC Tokenization	Minting	Commercial Property NFT	AssetTokenizationContract	Create digital representation of LLC shares
Vault Creation	Minting	Vault Token	VaultContract	Create specialized vaults for asset categories
Asset Aggregation	Transfer	Property NFT ? Vault	AssetAggregationContract	Move assets into appropriate vaults
Fractionalization	Minting	Fungible Tokens	FractionalizationContract	Create tradable fractional ownership tokens
Meta-Vault Creation	Minting	Meta-Vault Token	MetaVaultContract	Create diversified portfolio vault
Token Transfer	Transfer	Vault FT ? Meta-Vault	VaultContract	Move tokens between vaults for diversification
Governance Vote	Voting	Governance Token	GovernanceContract	Record holder votes on proposals
Asset Sale	Market Sale	Vault Asset ? ADA	VaultContract	Sell assets at market price
Asset Purchase	Market Purchase	ADA ? New Asset	VaultContract	Buy assets from whitelist at market price
Asset Distribution	Claim + Burn	Token Burn ? Asset Claim	DistributionContract	Distribute assets to token holders
Vault Merger	Complex	Vault A + Vault B ? New Vault	VaultMergerContract	Combine compatible vaults

Implementation Workflow

This implementation demonstrates how L4VA facilitates the fractionalization, aggregation, and governance of commercial real estate assets on the Cardano blockchain. The key value propositions include:

1. **Tokenization**: Converting illiquid real estate into tradable digital assets
2. **Fractionalization**: Breaking down large investments into affordable units
3. **Aggregation**: Combining multiple assets into thematic vaults
4. **Diversification**: Creating index-like products across property types
5. **Governance**: Enabling democratic management of real estate portfolios within strict security constraints
6. **DeFi Integration**: Unlocking new financial use cases for real estate assets

By leveraging L4VA, creators with influence and ideas can aggregate large vaults by attracting existing tokenized asset holders to contribute to vaults with specific configurations that represent an investment theme or strategy. The creators can then apply utility to their tokens as part of their community or other decentralized applications.

To execute this implementation on the Cardano preprod network, we'll use the Blockfrost API (project ID: `preprodGLOrJOlqUt1HBVbDhBTEh9oq7GVUBszv`) and our custom scripts to interact with the smart contracts.