Bitcoin Lending Protocol

Technical Specification Document

Phase 1: Stacks Mainnet Implementation

Version: 1.2 (Multi-Stablecoin Support)
Date: January 2026
Status: Draft for Review
Author: Jamie (Project Lead)

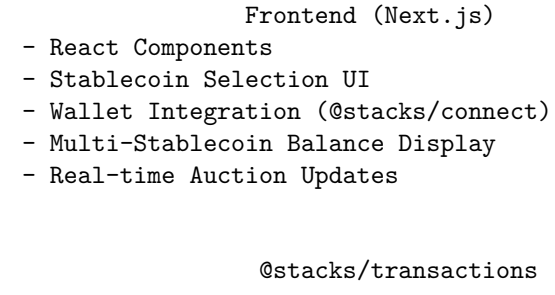
Change Log: - v1.0: Initial draft with incorrect descending auction - v1.1: Corrected to competitive bidding auction mechanism - v1.2: Added comprehensive multi-stablecoin support (USDA, USDC, xUSD)

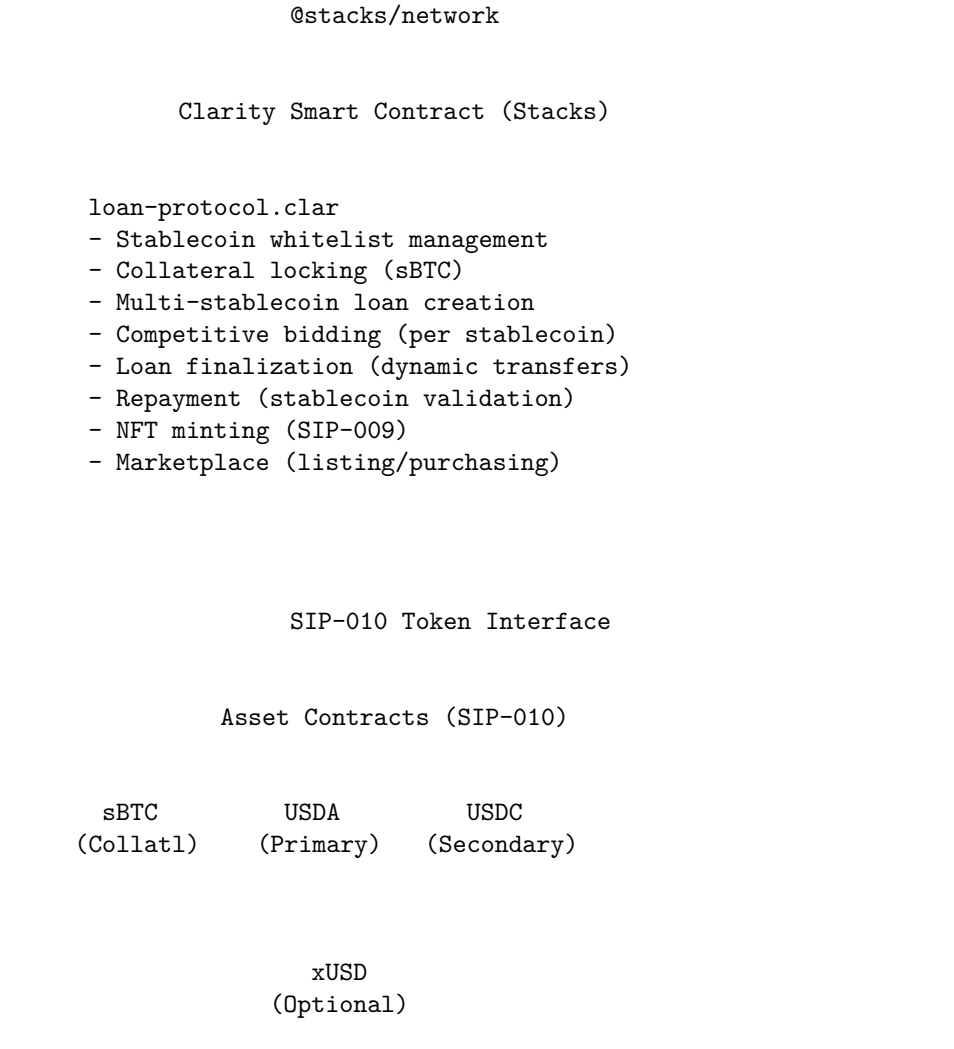
Table of Contents

---

1. Architecture Overview

1.1 System Components





1.2 Multi-Stablecoin Loan Flow

Phase 1: Loan Creation with Stablecoin Selection

User → Frontend

1. User selects stablecoin (USDA/USDC/xUSD)
2. User enters loan parameters
3. Frontend validates stablecoin is whitelisted
4. Frontend checks user has balance in that stablecoin

Frontend → Contract

```
  5. Call create-loan-auction with:
     - collateral-asset: .sbtc-token
     - collateral-amount: 150000000 (1.5 sBTC in sats)
     - borrow-asset: .usda-token (chosen stablecoin)
     - borrow-amount: 50000000000 ($50,000 USDA)
     - max-repayment: 53500000000 ($53,500 USDA)
     - loan-duration: 8640 blocks (60 days)
     - auction-duration: 144 blocks (24 hours)

Contract Processing
  6. Validate stablecoin is whitelisted
  7. Transfer sBTC from user to contract
  8. Create loan record with borrow-asset = .usda-token
  9. Emit loan-created event with stablecoin info
  10. Return loan-id
```

Phase 2: Competitive Bidding (Stablecoin-Specific)

```
Lender → Frontend
  1. Browse auctions, filter by stablecoin (e.g., "USDA only")
  2. Select loan #42 (USDA loan)
  3. Frontend shows: "This loan requires USDA"
  4. Frontend checks lender's USDA balance
  5. Lender enters bid: 51200000000 ($51,200 USDA)

Frontend → Contract
  6. Call place-bid(loan-id: 42, amount: 51200000000)

Contract Processing
  7. Verify loan.borrow-asset is .usda-token
  8. Check lender has sufficient USDA balance
  9. Verify bid < current-lowest-bid (or <= max-repayment if first)
  10. Store bid {bidder: lender, amount: 51200000000}
  11. Emit bid-placed event
```

Phase 3: Auction Finalization (Multi-Stablecoin)

```
Anyone → Contract
  1. Call finalize-auction(loan-id: 42)

Contract Processing
  2. Verify auction-end-block reached
  3. Get winning bid (lowest amount)
  4. Get loan's borrow-asset (.usda-token)
  5. Dynamic contract call to transfer USDA:
     (contract-call? .usda-token transfer
       50000000000              ;; Loan amount
       winning-bidder           ;; From winner
```

```
      borrower                ;; To borrower
      none)
  6. Update loan: status = "active", repayment-amount = 51200000000
  7. Mint borrower NFT to borrower
  8. Mint lender NFT to winner
  9. Emit loan-finalized with stablecoin info
```

Phase 4: Repayment (Stablecoin Validation)

```
Borrower → Contract
  1. Call repay-loan(loan-id: 42)

Contract Processing
  2. Get loan's borrow-asset (.usda-token)
  3. Verify borrower owns borrower NFT
  4. Get current lender (NFT owner of lender-position)
  5. Transfer USDA repayment:
     (contract-call? .usda-token transfer
       51200000000             ;; Fixed repayment amount
       tx-sender               ;; From borrower
       lender-nft-owner        ;; To current lender
       none)
  6. Return sBTC collateral:
     (as-contract (contract-call? .sbtc-token transfer
       150000000               ;; Collateral
       contract-self           ;; From contract
       borrower-nft-owner      ;; To current borrower
       none))
  7. Burn both NFTs
  8. Update status = "repaid"
  9. Emit loan-repaid with stablecoin amounts
```

---

2. Smart Contract Specifications

2.1 Contract Structure

File: `loan-protocol.clar`
Language: Clarity
Deployment: Stacks Mainnet

Contract Address (Mainnet): `SP[DEPLOYER-ADDRESS].loan-protocol`

2.2 Contract Constants

```
;; ========================================
;; Constants
```

```
;; ======================================

;; Asset Contracts
(define-constant SBTC_CONTRACT .sbtc-token)
(define-constant USDA_CONTRACT .usda-token)
(define-constant USDC_CONTRACT .usdc-token)
(define-constant XUSD_CONTRACT .xusd-token)

;; Contract Owner
(define-data-var contract-owner principal tx-sender)

;; Contract Address (for as-contract calls)
(define-data-var contract-address (optional principal) none)

;; Loan Counter
(define-data-var loan-nonce uint u0)

;; Timing Constants
(define-constant BLOCKS_PER_DAY u144)
(define-constant MIN_LOAN_DURATION (* u7 BLOCKS_PER_DAY))   ;; 7 days minimum
(define-constant MAX_LOAN_DURATION (* u365 BLOCKS_PER_DAY)) ;; 365 days maximum
(define-constant MIN_AUCTION_DURATION (* u12 (/ BLOCKS_PER_DAY u24))) ;; 12 hours
(define-constant MAX_AUCTION_DURATION (* u48 (/ BLOCKS_PER_DAY u24))) ;; 48 hours

;; Error Codes
(define-constant ERR_UNAUTHORIZED (err u401))
(define-constant ERR_INVALID_ASSET (err u402))
(define-constant ERR_SAME_ASSET (err u403))
(define-constant ERR_NO_INTEREST (err u404))
(define-constant ERR_INVALID_DURATION (err u405))
(define-constant ERR_LOAN_NOT_FOUND (err u406))
(define-constant ERR_AUCTION_ENDED (err u407))
(define-constant ERR_AUCTION_ACTIVE (err u408))
(define-constant ERR_BID_TOO_HIGH (err u409))
(define-constant ERR_BID_TOO_LOW (err u410))
(define-constant ERR_NO_BIDS (err u411))
(define-constant ERR_LOAN_NOT_ACTIVE (err u412))
(define-constant ERR_NOT_BORROWER (err u413))
(define-constant ERR_NOT_LENDER (err u414))
(define-constant ERR_NOT_MATURED (err u415))
(define-constant ERR_NOT_INITIALIZED (err u416))
(define-constant ERR_ALREADY_INITIALIZED (err u417))
(define-constant ERR_OWNER_ONLY (err u418))
(define-constant ERR_NOT_POSITION_OWNER (err u419))
(define-constant ERR_STABLECOIN_NOT_APPROVED (err u420))
(define-constant ERR_INSUFFICIENT_STABLECOIN (err u421))
```

2.3 Stablecoin Whitelist Management

```
;; ========================================
;; Stablecoin Whitelist
;; ========================================

;; Map of approved stablecoins
(define-map approved-stablecoins principal bool)

;; Initialize approved stablecoins
(map-set approved-stablecoins USDA_CONTRACT true)
(map-set approved-stablecoins USDC_CONTRACT true)
(map-set approved-stablecoins XUSD_CONTRACT true)

;; Check if stablecoin is approved
(define-read-only (is-stablecoin-approved (token principal))
  (default-to false (map-get? approved-stablecoins token))
)

;; Get all approved stablecoins (helper for frontend)
(define-read-only (get-approved-stablecoins)
  {
    usda: (is-stablecoin-approved USDA_CONTRACT),
    usdc: (is-stablecoin-approved USDC_CONTRACT),
    xusd: (is-stablecoin-approved XUSD_CONTRACT)
  }
)

;; Owner can add new stablecoin (governance/emergency)
(define-public (add-approved-stablecoin (token principal))
  (begin
    (asserts! (is-eq tx-sender (var-get contract-owner)) ERR_OWNER_ONLY)
    (ok (map-set approved-stablecoins token true))
  )
)

;; Owner can remove stablecoin (emergency only)
(define-public (remove-approved-stablecoin (token principal))
  (begin
    (asserts! (is-eq tx-sender (var-get contract-owner)) ERR_OWNER_ONLY)
    (ok (map-set approved-stablecoins token false))
  )
)
```

3. Data Structures

3.1 Loan Structure (Multi-Stablecoin)

```
;; =======================================
;; Data Structures
;; =======================================

(define-map loans
  {loan-id: uint}
  {
    borrower: principal,
    collateral-asset: principal,        ;; Always sBTC in Phase 1
    collateral-amount: uint,            ;; sBTC amount (satoshis)
    borrow-asset: principal,            ;; NEW: USDA/USDC/xUSD contract address
    borrow-amount: uint,                ;; Amount in chosen stablecoin (micro-units)
    max-repayment: uint,                ;; Maximum repayment in stablecoin (micro-units)
    repayment-amount: uint,             ;; Actual repayment from winning bid
    loan-duration-blocks: uint,         ;; Duration in blocks
    auction-duration-blocks: uint,      ;; Auction window in blocks
    auction-start-block: uint,          ;; Block when auction started
    auction-end-block: uint,            ;; Block when auction ends
    maturity-block: uint,               ;; Block when loan matures
    lender: (optional principal),       ;; Winner of auction (set after finalization)
    status: (string-ascii 20)          ;; "auction", "active", "repaid", "defaulted"
  }
)

;; Example loan record:
;; {
;;   loan-id: u1,
;;   borrower: 'SP2ABC...XYZ,
;;   collateral-asset: .sbtc-token,
;;   collateral-amount: u150000000,        ;; 1.5 sBTC
;;   borrow-asset: .usda-token,            ;; Borrower chose USDA
;;   borrow-amount: u50000000000,          ;; $50,000 USDA (6 decimals)
;;   max-repayment: u53500000000,          ;; $53,500 USDA max
;;   repayment-amount: u51200000000,       ;; $51,200 USDA (winning bid)
;;   loan-duration-blocks: u8640,          ;; 60 days
;;   auction-duration-blocks: u144,        ;; 24 hours
;;   auction-start-block: u100000,
;;   auction-end-block: u100144,
;;   maturity-block: u108784,
;;   lender: (some 'SP2DEF...ABC),
;;   status: "active"
;; }
```

3.2 Bid Structure

```
;; Current bids (only stores lowest bid per auction)
(define-map current-bids
  {loan-id: uint}
  {
    bidder: principal,
    amount: uint,                        ;; Total repayment amount in loan's stablecoin
    bid-block: uint                      ;; Block when bid placed
  }
)

;; Bid history (for analytics and UI)
(define-map bid-history
  {loan-id: uint, bid-index: uint}
  {
    bidder: principal,
    amount: uint,                        ;; Repayment amount in loan's stablecoin
    bid-block: uint
  }
)

(define-map bid-count
  {loan-id: uint}
  uint
)
```

3.3 NFT Position Structures

```
;; NFT token IDs map to loan IDs
;; Each loan generates two NFTs with the same token ID (loan-id)

;; Borrower Position NFT
(define-non-fungible-token borrower-position uint)

;; Lender Position NFT
(define-non-fungible-token lender-position uint)

;; Marketplace Listings
(define-map nft-listings
  {token-id: uint, position-type: (string-ascii 10)} ;; "borrower" or "lender"
  {
    seller: principal,
    price: uint,                         ;; Price in loan's stablecoin
    stablecoin: principal,               ;; Which stablecoin for this listing
    listed-at-block: uint
```

```
  }
)
```

---

4. Core Functions

4.1 Contract Initialization

```
;; ========================================
;; Initialization
;; ========================================

(define-public (initialize)
  (begin
    (asserts! (is-none (var-get contract-address)) ERR_ALREADY_INITIALIZED)
    (var-set contract-address (some tx-sender))
    (ok true)
  )
)

;; Must be called after deployment
;; Sets contract's own address for as-contract transfers
```

4.2 Create Loan Auction (Multi-Stablecoin)

```
;; ========================================
;; Core Loan Functions
;; ========================================

(define-public (create-loan-auction
    (collateral-asset principal)
    (collateral-amount uint)
    (borrow-asset principal)              ;; NEW: Which stablecoin
    (borrow-amount uint)
    (max-repayment uint)
    (loan-duration-blocks uint)
    (auction-duration-blocks uint))
  (let
    (
      (loan-id (+ (var-get loan-nonce) u1))
      (current-block burn-block-height)
      (auction-end (+ current-block auction-duration-blocks))
      (maturity (+ auction-end loan-duration-blocks))
      (contract-addr (unwrap! (var-get contract-address) ERR_NOT_INITIALIZED))
    )
```

```
;; Validate stablecoin is approved
(asserts! (is-stablecoin-approved borrow-asset) ERR_STABLECOIN_NOT_APPROVED)

;; Validate collateral is sBTC (Phase 1)
(asserts! (is-eq collateral-asset SBTC_CONTRACT) ERR_INVALID_ASSET)

;; Validate collateral != borrow asset
(asserts! (not (is-eq collateral-asset borrow-asset)) ERR_SAME_ASSET)

;; Validate max repayment > borrow amount (need positive interest)
(asserts! (> max-repayment borrow-amount) ERR_NO_INTEREST)

;; Validate loan duration
(asserts! (>= loan-duration-blocks MIN_LOAN_DURATION) ERR_INVALID_DURATION)
(asserts! (<= loan-duration-blocks MAX_LOAN_DURATION) ERR_INVALID_DURATION)

;; Validate auction duration
(asserts! (>= auction-duration-blocks MIN_AUCTION_DURATION) ERR_INVALID_DURATION)
(asserts! (<= auction-duration-blocks MAX_AUCTION_DURATION) ERR_INVALID_DURATION)

;; Transfer collateral (sBTC) from borrower to contract
(try! (contract-call? SBTC_CONTRACT transfer
  collateral-amount
  tx-sender
  contract-addr
  none))

;; Store loan record with borrow-asset
(map-set loans
  {loan-id: loan-id}
  {
    borrower: tx-sender,
    collateral-asset: collateral-asset,
    collateral-amount: collateral-amount,
    borrow-asset: borrow-asset,              ;; Store chosen stablecoin
    borrow-amount: borrow-amount,
    max-repayment: max-repayment,
    repayment-amount: u0,
    loan-duration-blocks: loan-duration-blocks,
    auction-duration-blocks: auction-duration-blocks,
    auction-start-block: current-block,
    auction-end-block: auction-end,
    maturity-block: maturity,
    lender: none,
    status: "auction"
  })
```

```
    ;; Initialize bid count
    (map-set bid-count {loan-id: loan-id} u0)

    ;; Increment nonce
    (var-set loan-nonce loan-id)

    ;; Emit event with stablecoin info
    (print {
      event: "loan-created",
      loan-id: loan-id,
      borrower: tx-sender,
      collateral-amount: collateral-amount,
      borrow-asset: borrow-asset,              ;; Include stablecoin
      borrow-amount: borrow-amount,
      max-repayment: max-repayment,
      auction-end: auction-end
    })

    (ok loan-id)
  )
)
```

4.3 Place Bid (Stablecoin Balance Check)

```
(define-public (place-bid (loan-id uint) (amount uint))
  (let
    (
      (loan (unwrap! (map-get? loans {loan-id: loan-id}) ERR_LOAN_NOT_FOUND))
      (current-block burn-block-height)
      (current-bid (map-get? current-bids {loan-id: loan-id}))
      (bid-index (default-to u0 (map-get? bid-count {loan-id: loan-id})))
      (stablecoin-contract (get borrow-asset loan))
    )

    ;; Verify auction is still active
    (asserts! (< current-block (get auction-end-block loan)) ERR_AUCTION_ENDED)
    (asserts! (is-eq (get status loan) "auction") ERR_AUCTION_ENDED)

    ;; Verify bid amount
    (match current-bid
      existing-bid
        ;; Must be lower than current bid
        (asserts! (< amount (get amount existing-bid)) ERR_BID_TOO_HIGH)
      ;; First bid: must be <= max repayment
      (asserts! (<= amount (get max-repayment loan)) ERR_BID_TOO_HIGH)
```

```
)

;; Prevent unreasonably low bids (must be at least borrow amount)
(asserts! (>= amount (get borrow-amount loan)) ERR_BID_TOO_LOW)

;; CRITICAL: Check bidder has sufficient balance in correct stablecoin
;; This is a read-only check; actual transfer happens at finalization
(let
  (
    (bidder-balance (unwrap!
      (contract-call? stablecoin-contract get-balance tx-sender)
      ERR_INSUFFICIENT_STABLECOIN))
  )
  (asserts! (>= bidder-balance amount) ERR_INSUFFICIENT_STABLECOIN)
)

;; Store as current bid
(map-set current-bids
  {loan-id: loan-id}
  {
    bidder: tx-sender,
    amount: amount,
    bid-block: current-block
  })

;; Store in bid history
(map-set bid-history
  {loan-id: loan-id, bid-index: bid-index}
  {
    bidder: tx-sender,
    amount: amount,
    bid-block: current-block
  })

;; Increment bid count
(map-set bid-count {loan-id: loan-id} (+ bid-index u1))

;; Emit event
(print {
  event: "bid-placed",
  loan-id: loan-id,
  bidder: tx-sender,
  amount: amount,
  stablecoin: stablecoin-contract,
  implied-apr: (calculate-apr (get borrow-amount loan) amount (get loan-duration-blocks
})
```

```
    (ok true)
  )
)
```

4.4 Finalize Auction (Dynamic Stablecoin Transfer)

```
(define-public (finalize-auction (loan-id uint))
  (let
    (
      (loan (unwrap! (map-get? loans {loan-id: loan-id}) ERR_LOAN_NOT_FOUND))
      (winning-bid (unwrap! (map-get? current-bids {loan-id: loan-id}) ERR_NO_BIDS))
      (current-block burn-block-height)
      (stablecoin-contract (get borrow-asset loan))   ;; Get loan's stablecoin
    )

    ;; Verify auction has ended
    (asserts! (>= current-block (get auction-end-block loan)) ERR_AUCTION_ACTIVE)
    (asserts! (is-eq (get status loan) "auction") ERR_AUCTION_ENDED)

    ;; DYNAMIC TRANSFER: Call the correct stablecoin contract
    ;; Transfer loan amount from winner to borrower
    (try! (contract-call? stablecoin-contract transfer
      (get borrow-amount loan)
      (get bidder winning-bid)
      (get borrower loan)
      none))

    ;; Update loan record
    (map-set loans
      {loan-id: loan-id}
      (merge loan {
        lender: (some (get bidder winning-bid)),
        repayment-amount: (get amount winning-bid),
        status: "active"
      }))

    ;; Mint NFT positions
    (try! (nft-mint? borrower-position loan-id (get borrower loan)))
    (try! (nft-mint? lender-position loan-id (get bidder winning-bid)))

    ;; Emit event with stablecoin details
    (print {
      event: "auction-finalized",
      loan-id: loan-id,
      borrower: (get borrower loan),
```

```
      lender: (get bidder winning-bid),
      borrow-amount: (get borrow-amount loan),
      repayment-amount: (get amount winning-bid),
      stablecoin: stablecoin-contract,
      maturity-block: (get maturity-block loan)
    })

    (ok true)
  )
)
```

4.5 Repay Loan (Stablecoin Validation)

```
(define-public (repay-loan (loan-id uint))
  (let
    (
      (loan (unwrap! (map-get? loans {loan-id: loan-id}) ERR_LOAN_NOT_FOUND))
      (borrower-nft-owner (unwrap! (nft-get-owner? borrower-position loan-id) ERR_NOT_BORROW
      (lender-nft-owner (unwrap! (nft-get-owner? lender-position loan-id) ERR_NOT_LENDER))
      (contract-addr (unwrap! (var-get contract-address) ERR_NOT_INITIALIZED))
      (stablecoin-contract (get borrow-asset loan))   ;; Get loan's stablecoin
    )

    ;; Verify loan is active
    (asserts! (is-eq (get status loan) "active") ERR_LOAN_NOT_ACTIVE)

    ;; Verify caller owns borrower NFT
    (asserts! (is-eq tx-sender borrower-nft-owner) ERR_NOT_BORROWER)

    ;; DYNAMIC TRANSFER: Repayment in correct stablecoin
    ;; Transfer repayment amount from borrower to lender
    (try! (contract-call? stablecoin-contract transfer
      (get repayment-amount loan)
      tx-sender
      lender-nft-owner
      none))

    ;; Return collateral (sBTC) to borrower
    (try! (as-contract (contract-call? SBTC_CONTRACT transfer
      (get collateral-amount loan)
      contract-addr
      borrower-nft-owner
      none)))

    ;; Burn NFTs
    (try! (nft-burn? borrower-position loan-id borrower-nft-owner))
```

```
    (try! (nft-burn? lender-position loan-id lender-nft-owner))

    ;; Update loan status
    (map-set loans
      {loan-id: loan-id}
      (merge loan {status: "repaid"}))

    ;; Emit event with stablecoin amounts
    (print {
      event: "loan-repaid",
      loan-id: loan-id,
      borrower: borrower-nft-owner,
      lender: lender-nft-owner,
      repayment-amount: (get repayment-amount loan),
      stablecoin: stablecoin-contract,
      collateral-returned: (get collateral-amount loan)
    })

    (ok true)
  )
)
```

4.6 Claim Collateral (Default)

```
(define-public (claim-collateral (loan-id uint))
  (let
    (
      (loan (unwrap! (map-get? loans {loan-id: loan-id}) ERR_LOAN_NOT_FOUND))
      (lender-nft-owner (unwrap! (nft-get-owner? lender-position loan-id) ERR_NOT_LENDER))
      (borrower-nft-owner (unwrap! (nft-get-owner? borrower-position loan-id) ERR_NOT_BORROW
      (current-block burn-block-height)
      (contract-addr (unwrap! (var-get contract-address) ERR_NOT_INITIALIZED))
    )

    ;; Verify loan is active and matured
    (asserts! (is-eq (get status loan) "active") ERR_LOAN_NOT_ACTIVE)
    (asserts! (>= current-block (get maturity-block loan)) ERR_NOT_MATURED)

    ;; Verify caller owns lender NFT
    (asserts! (is-eq tx-sender lender-nft-owner) ERR_NOT_LENDER)

    ;; Transfer collateral to lender
    (try! (as-contract (contract-call? SBTC_CONTRACT transfer
      (get collateral-amount loan)
      contract-addr
      lender-nft-owner
```

```
      none)))

    ;; Burn NFTs
    (try! (nft-burn? borrower-position loan-id borrower-nft-owner))
    (try! (nft-burn? lender-position loan-id lender-nft-owner))

    ;; Update loan status
    (map-set loans
      {loan-id: loan-id}
      (merge loan {status: "defaulted"}))

    ;; Emit event
    (print {
      event: "collateral-claimed",
      loan-id: loan-id,
      lender: lender-nft-owner,
      collateral-amount: (get collateral-amount loan)
    })

    (ok true)
  )
)
```

---

5. Auction Mechanism

5.1 Competitive Bidding Process (Multi-Stablecoin)

Auction Timeline Example with USDA:

```
Block 100000: Loan Created

Borrower locks 1.5 sBTC
Requests $50,000 USDA (chosen stablecoin)
Max repayment: $53,500 USDA
Auction ends: Block 100144 (24 hours)
Maturity: Block 108784 (60 days after auction)

Block 100020 (2 hours): First Bid

Alice bids $53,500 USDA (7.0% APR)
  Valid: <= max-repayment
  Alice has 60,000 USDA balance
Current leader: Alice @ $53,500

Block 100060 (8 hours): Second Bid
```

```
Bob bids $52,800 USDA (5.6% APR)
  Valid: < current bid ($53,500)
  Bob has 55,000 USDA balance
Current leader: Bob @ $52,800

Block 100100 (14 hours): Third Bid

Carol bids $51,200 USDA (2.4% APR)
  Valid: < current bid ($52,800)
  Carol has 75,000 USDA balance
Current leader: Carol @ $51,200

Block 100120 (17 hours): Fourth Bid (Rejected)

Dave tries to bid $51,500 USDA
  Invalid: >= current bid ($51,200)
Transaction reverts: ERR_BID_TOO_HIGH

Block 100130 (20 hours): Fifth Bid

Dave bids $50,500 USDA (1.0% APR)
  Valid: < current bid ($51,200)
  Dave has 52,000 USDA balance
Current leader: Dave @ $50,500

Block 100144: Auction Ends

Anyone calls finalize-auction(loan-id: 1)

Finalization:
1. Winner identified: Dave (lowest bid)
2. Transfer $50,000 USDA: Dave → Borrower
3. Repayment set: $50,500 USDA
4. NFTs minted: borrower-position-1, lender-position-1
5. Status: "active"

Block 108784: Loan Matures (60 days later)

Borrower calls repay-loan(loan-id: 1)

Repayment:
1. Transfer $50,500 USDA: Borrower → Dave
2. Return 1.5 sBTC: Contract → Borrower
3. Burn both NFTs
4. Status: "repaid"
```

```
Dave's Profit: $500 USDA (1.0% APR over 60 days)
```

5.2 Multi-Stablecoin Auction Examples

Example 1: USDA Loan (Native)

```
Borrower: Prefers USDA (has USDA, wants to stay in Stacks ecosystem)
Borrow: $50,000 USDA
Max: $53,500 USDA

Lenders (all have USDA):
- Alice (USDA DeFi farmer): Bids $52,500
- Bob (USDA hodler): Bids $51,000 ← Winner
- Carol (late): Auction ended

Result: Bob lends 50,000 USDA, earns $1,000 profit
```

Example 2: USDC Loan (Institutional)

```
Borrower: Mining company prefers USDC (institutional standard)
Borrow: $500,000 USDC
Max: $535,000 USDC

Lenders (institutional):
- Hedge Fund A: Bids $530,000 USDC
- Family Office B: Bids $520,000 USDC ← Winner
- Crypto Fund C: Bids $525,000 USDC (too late)

Result: Family Office lends 500,000 USDC, earns $20,000 profit
```

Example 3: Mixed Portfolio (Lender Perspective)

```
Lender has:
- 100,000 USDA
- 50,000 USDC

Strategy: Browse both types
- Bid on USDA loans with USDA balance
- Bid on USDC loans with USDC balance
- Optimize APR across both

Portfolio:
- USDA Loan #12: $40,000 @ 8% APR
- USDA Loan #18: $60,000 @ 6% APR
- USDC Loan #23: $50,000 @ 9% APR

Diversified across stablecoins!
```

---

6. NFT Implementation

6.1 Overview

The protocol implements **SIP-009 compliant NFTs** to represent tradeable loan positions. Each active loan generates two NFTs: - **Borrower NFT**: Represents the obligation to repay (debt position) - **Lender NFT**: Represents the right to receive repayment (credit position)

These NFTs enable a **secondary market** where users can transfer loan positions before maturity, providing liquidity and flexibility.

6.2 SIP-009 NFT Standard

Standard Interface (Stacks Improvement Proposal 009):

```
;; SIP-009 Trait Definition
(define-trait nft-trait
  (
    ;; Transfer token to a specified principal
    (transfer (uint principal principal) (response bool uint))

    ;; Get the owner of a specific token ID
    (get-owner (uint) (response (optional principal) uint))

    ;; Get the token URI for a specific token ID
    (get-token-uri (uint) (response (optional (string-ascii 256)) uint))

    ;; Get the last token ID minted
    (get-last-token-id () (response uint uint))
  )
)
```

6.3 Borrower NFT Implementation

Contract: `borrower-nft.clar`

6.3.1 Data Structures

```
;; Borrower NFT Definition
(define-non-fungible-token borrower-nft uint)

;; Map: NFT ID → Loan ID
```

```
(define-map nft-to-loan
  {nft-id: uint}
  {loan-id: uint}
)

;; Map: Loan ID → NFT ID (reverse lookup)
(define-map loan-to-nft
  {loan-id: uint}
  {nft-id: uint}
)

;; Counter for NFT IDs
(define-data-var last-nft-id uint u0)

;; NFT Metadata
(define-map nft-metadata
  {nft-id: uint}
  {
    loan-id: uint,
    collateral-amount: uint,
    borrow-amount: uint,
    borrow-asset: principal,  ;; Stablecoin contract
    repayment-amount: uint,
    maturity-block: uint,
    created-at: uint
  }
)
```

6.3.2 Minting Function

```
(define-public (mint-borrower-nft (loan-id uint) (borrower principal))
  (let
    (
      (new-nft-id (+ (var-get last-nft-id) u1))
      (loan-data (unwrap! (map-get? loans {loan-id: loan-id}) err-loan-not-found))
    )

    ;; Verify caller is the loan protocol contract
    (asserts! (is-eq tx-sender .loan-protocol) err-unauthorized)

    ;; Verify loan exists and is active
    (asserts! (is-eq (get status loan-data) "active") err-invalid-status)

    ;; Mint NFT to borrower
    (try! (nft-mint? borrower-nft new-nft-id borrower))
```

```
    ;; Store mappings
    (map-set nft-to-loan {nft-id: new-nft-id} {loan-id: loan-id})
    (map-set loan-to-nft {loan-id: loan-id} {nft-id: new-nft-id})

    ;; Store metadata
    (map-set nft-metadata
      {nft-id: new-nft-id}
      {
        loan-id: loan-id,
        collateral-amount: (get collateral-amount loan-data),
        borrow-amount: (get borrow-amount loan-data),
        borrow-asset: (get borrow-asset loan-data),
        repayment-amount: (get repayment-amount loan-data),
        maturity-block: (get maturity-block loan-data),
        created-at: block-height
      }
    )

    ;; Increment counter
    (var-set last-nft-id new-nft-id)

    (ok new-nft-id)
  )
)
```

6.3.3 Transfer Function

```
(define-public (transfer (nft-id uint) (sender principal) (recipient principal))
  (let
    (
      (owner (unwrap! (nft-get-owner? borrower-nft nft-id) err-not-found))
      (loan-id (unwrap! (get loan-id (map-get? nft-to-loan {nft-id: nft-id})) err-not-found)
    )

    ;; Verify sender owns the NFT
    (asserts! (is-eq sender tx-sender) err-unauthorized)
    (asserts! (is-eq owner sender) err-not-owner)

    ;; Transfer NFT
    (try! (nft-transfer? borrower-nft nft-id sender recipient))

    ;; Notify loan protocol of ownership change
    (try! (contract-call? .loan-protocol update-borrower loan-id recipient))

    (ok true)
  )
```

```
)
```

6.3.4 Burn Function

```
(define-public (burn (nft-id uint))
  (let
    (
      (owner (unwrap! (nft-get-owner? borrower-nft nft-id) err-not-found))
      (loan-id (unwrap! (get loan-id (map-get? nft-to-loan {nft-id: nft-id})) err-not-found)
    )

    ;; Only loan protocol can burn (when loan completed/defaulted)
    (asserts! (is-eq tx-sender .loan-protocol) err-unauthorized)

    ;; Burn NFT
    (try! (nft-burn? borrower-nft nft-id owner))

    ;; Clean up mappings
    (map-delete nft-to-loan {nft-id: nft-id})
    (map-delete loan-to-nft {loan-id: loan-id})
    (map-delete nft-metadata {nft-id: nft-id})

    (ok true)
  )
)
```

6.3.5 Metadata & Token URI

```
(define-read-only (get-token-uri (nft-id uint))
  (let
    (
      (metadata (unwrap! (map-get? nft-metadata {nft-id: nft-id}) (ok none)))
    )
    (ok (some (concat
      "https://api.bitcoinlending.xyz/nft/borrower/"
      (uint-to-ascii nft-id)
    )))
  )
)

(define-read-only (get-metadata (nft-id uint))
  (ok (map-get? nft-metadata {nft-id: nft-id}))
)
```

Token URI Response (from API):

```
{
```

```
  "name": "Bitcoin Loan Position #42 (Borrower)",
  "description": "Borrower position for loan #42. Holder must repay 50,800 USDA by block 145
  "image": "https://api.bitcoinlending.xyz/nft/borrower/42.png",
  "attributes": [
    {
      "trait_type": "Role",
      "value": "Borrower"
    },
    {
      "trait_type": "Loan ID",
      "value": "42"
    },
    {
      "trait_type": "Collateral",
      "value": "1.5 sBTC"
    },
    {
      "trait_type": "Borrowed",
      "value": "50,000 USDA"
    },
    {
      "trait_type": "Repayment Required",
      "value": "50,800 USDA"
    },
    {
      "trait_type": "Stablecoin",
      "value": "USDA"
    },
    {
      "trait_type": "APR",
      "value": "3.2%"
    },
    {
      "trait_type": "Maturity Block",
      "value": "145680"
    },
    {
      "trait_type": "Status",
      "value": "Active"
    }
  ]
}
```

6.4 Lender NFT Implementation

Contract: `lender-nft.clar`

6.4.1 Data Structures

```
;; Lender NFT Definition
(define-non-fungible-token lender-nft uint)

;; Map: NFT ID → Loan ID
(define-map nft-to-loan
  {nft-id: uint}
  {loan-id: uint}
)

;; Map: Loan ID → NFT ID
(define-map loan-to-nft
  {loan-id: uint}
  {nft-id: uint}
)

;; Counter
(define-data-var last-nft-id uint u0)

;; NFT Metadata
(define-map nft-metadata
  {nft-id: uint}
  {
    loan-id: uint,
    lent-amount: uint,
    lent-asset: principal,  ;; Stablecoin contract
    expected-return: uint,
    maturity-block: uint,
    collateral-amount: uint,
    created-at: uint
  }
)
```

6.4.2 Minting Function

```
(define-public (mint-lender-nft (loan-id uint) (lender principal))
  (let
    (
      (new-nft-id (+ (var-get last-nft-id) u1))
      (loan-data (unwrap! (map-get? loans {loan-id: loan-id}) err-loan-not-found))
    )
```

```
    ;; Verify caller is loan protocol
    (asserts! (is-eq tx-sender .loan-protocol) err-unauthorized)

    ;; Verify loan is active
    (asserts! (is-eq (get status loan-data) "active") err-invalid-status)

    ;; Mint NFT to lender
    (try! (nft-mint? lender-nft new-nft-id lender))

    ;; Store mappings
    (map-set nft-to-loan {nft-id: new-nft-id} {loan-id: loan-id})
    (map-set loan-to-nft {loan-id: loan-id} {nft-id: new-nft-id})

    ;; Store metadata
    (map-set nft-metadata
      {nft-id: new-nft-id}
      {
        loan-id: loan-id,
        lent-amount: (get borrow-amount loan-data),
        lent-asset: (get borrow-asset loan-data),
        expected-return: (get repayment-amount loan-data),
        maturity-block: (get maturity-block loan-data),
        collateral-amount: (get collateral-amount loan-data),
        created-at: block-height
      }
    )

    ;; Increment counter
    (var-set last-nft-id new-nft-id)

    (ok new-nft-id)
  )
)
```

6.4.3 Transfer Function

```
(define-public (transfer (nft-id uint) (sender principal) (recipient principal))
  (let
    (
      (owner (unwrap! (nft-get-owner? lender-nft nft-id) err-not-found))
      (loan-id (unwrap! (get loan-id (map-get? nft-to-loan {nft-id: nft-id})) err-not-found)
    )

    ;; Verify sender owns the NFT
    (asserts! (is-eq sender tx-sender) err-unauthorized)
    (asserts! (is-eq owner sender) err-not-owner)
```

```
    ;; Transfer NFT
    (try! (nft-transfer? lender-nft nft-id sender recipient))

    ;; Notify loan protocol of ownership change
    (try! (contract-call? .loan-protocol update-lender loan-id recipient))

    (ok true)
  )
)
```

6.4.4 Burn Function

```
(define-public (burn (nft-id uint))
  (let
    (
      (owner (unwrap! (nft-get-owner? lender-nft nft-id) err-not-found))
      (loan-id (unwrap! (get loan-id (map-get? nft-to-loan {nft-id: nft-id})) err-not-found)
    )

    ;; Only loan protocol can burn
    (asserts! (is-eq tx-sender .loan-protocol) err-unauthorized)

    ;; Burn NFT
    (try! (nft-burn? lender-nft nft-id owner))

    ;; Clean up mappings
    (map-delete nft-to-loan {nft-id: nft-id})
    (map-delete loan-to-nft {loan-id: loan-id})
    (map-delete nft-metadata {nft-id: nft-id})

    (ok true)
  )
)
```

6.4.5 Metadata & Token URI

```
(define-read-only (get-token-uri (nft-id uint))
  (let
    (
      (metadata (unwrap! (map-get? nft-metadata {nft-id: nft-id}) (ok none)))
    )
    (ok (some (concat
      "https://api.bitcoinlending.xyz/nft/lender/"
      (uint-to-ascii nft-id)
    )))
```

```
  )
)
```

Token URI Response:


```
{
  "name": "Bitcoin Loan Position #42 (Lender)",
  "description": "Lender position for loan #42. Holder will receive 50,800 USDA at maturity
  "image": "https://api.bitcoinlending.xyz/nft/lender/42.png",
  "attributes": [
    {
      "trait_type": "Role",
      "value": "Lender"
    },
    {
      "trait_type": "Loan ID",
      "value": "42"
    },
    {
      "trait_type": "Lent Amount",
      "value": "50,000 USDA"
    },
    {
      "trait_type": "Expected Return",
      "value": "50,800 USDA"
    },
    {
      "trait_type": "Stablecoin",
      "value": "USDA"
    },
    {
      "trait_type": "Profit",
      "value": "800 USDA"
    },
    {
      "trait_type": "APR",
      "value": "3.2%"
    },
    {
      "trait_type": "Collateral",
      "value": "1.5 sBTC"
    },
    {
      "trait_type": "Maturity Block",
      "value": "145680"
    },
    {
```

```
      "trait_type": "Status",
      "value": "Active"
    }
  ]
}
```

6.5 NFT Lifecycle

Loan Creation → NFT Minting Flow

```
1. Auction Finalizes (loan-protocol.clar)
   ↓
2. Call mint-borrower-nft(loan-id, borrower-address)
   ↓
3. Borrower NFT minted to borrower
   ↓
4. Call mint-lender-nft(loan-id, lender-address)
   ↓
5. Lender NFT minted to lender
   ↓
6. Both NFTs now tradeable on secondary market
```

Loan Completion → NFT Burning Flow

```
1. Borrower Repays Loan
   ↓
2. Loan status: "completed"
   ↓
3. Call burn-borrower-nft(borrower-nft-id)
   ↓
4. Borrower NFT burned
   ↓
5. Call burn-lender-nft(lender-nft-id)
   ↓
6. Lender NFT burned
   ↓
7. NFTs removed from existence
```

Loan Default → NFT Burning Flow

```
1. Maturity Block Reached + No Repayment
   ↓
2. Lender claims collateral
   ↓
3. Loan status: "defaulted"
   ↓
```

```
4. Call burn-borrower-nft(borrower-nft-id)
   ↓
5. Call burn-lender-nft(lender-nft-id)
   ↓
6. Both NFTs burned
```

6.6 NFT Image Generation

Dynamic NFT Images generated by API based on loan metadata:

`https://api.bitcoinlending.xyz/nft/borrower/{nft-id}.png`
`https://api.bitcoinlending.xyz/nft/lender/{nft-id}.png`

Borrower NFT Image Features: - Red/orange color scheme (debt indicator) - Displays: Collateral amount, borrowed amount, repayment due - Shows: Days until maturity - Badge: Stablecoin type (USDA/USDC/xUSD)

Lender NFT Image Features: - Green/blue color scheme (credit indicator) - Displays: Lent amount, expected return, profit - Shows: APR percentage - Badge: Stablecoin type (USDA/USDC/xUSD)

Example API Implementation (Node.js):

```javascript
// Generate NFT image
app.get('/nft/:role/:nftId.png', async (req, res) => {
  const { role, nftId } = req.params;

  // Fetch NFT metadata from blockchain
  const metadata = await fetchNFTMetadata(role, nftId);

  // Generate image using canvas
  const canvas = createCanvas(500, 500);
  const ctx = canvas.getContext('2d');

  // Background gradient
  const gradient = ctx.createLinearGradient(0, 0, 0, 500);
  if (role === 'borrower') {
    gradient.addColorStop(0, '#ff6b6b');
    gradient.addColorStop(1, '#ee5a6f');
  } else {
    gradient.addColorStop(0, '#51cf66');
    gradient.addColorStop(1, '#37b24d');
  }
  ctx.fillStyle = gradient;
  ctx.fillRect(0, 0, 500, 500);

  // Add text: loan details, amounts, maturity, etc.
```

```
  ctx.fillStyle = '#ffffff';
  ctx.font = 'bold 24px Arial';
  ctx.fillText(`Loan #${metadata.loanId}`, 50, 100);

  // ... (more text rendering)

  // Return PNG
  res.setHeader('Content-Type', 'image/png');
  canvas.createPNGStream().pipe(res);
});
```

6.7 Integration with Loan Protocol

Loan Protocol Must Track NFT IDs:

```
;; In loan-protocol.clar
(define-map loan-nfts
  {loan-id: uint}
  {
    borrower-nft-id: uint,
    lender-nft-id: uint
  }
)
```

Update Borrower Function (called by borrower-nft.clar on transfer):

```
(define-public (update-borrower (loan-id uint) (new-borrower principal))
  (let
    (
      (loan (unwrap! (map-get? loans {loan-id: loan-id}) err-not-found))
    )

    ;; Only borrower-nft contract can call
    (asserts! (is-eq tx-sender .borrower-nft) err-unauthorized)

    ;; Update borrower
    (map-set loans
      {loan-id: loan-id}
      (merge loan {borrower: new-borrower})
    )

    (ok true)
  )
)
```

Update Lender Function (called by lender-nft.clar on transfer):

```
(define-public (update-lender (loan-id uint) (new-lender principal))
  (let
    (
      (loan (unwrap! (map-get? loans {loan-id: loan-id}) err-not-found))
    )

    ;; Only lender-nft contract can call
    (asserts! (is-eq tx-sender .lender-nft) err-unauthorized)

    ;; Update lender
    (map-set loans
      {loan-id: loan-id}
      (merge loan {lender: new-lender})
    )

    (ok true)
  )
)
```

6.8 NFT Testing Strategy

Unit Tests:

```
describe('Borrower NFT', () => {
  it('should mint NFT when loan activates', async () => {
    // Create and finalize loan
    const loanId = await createAndFinalizeLoan();

    // Check borrower NFT minted
    const nftId = await borrowerNFT.getNFTForLoan(loanId);
    expect(nftId).toBeGreaterThan(0);

    // Check owner is borrower
    const owner = await borrowerNFT.getOwner(nftId);
    expect(owner).toBe(borrowerAddress);
  });

  it('should transfer NFT and update loan borrower', async () => {
    const nftId = 1;
    const newOwner = 'SP3ABC...';

    // Transfer NFT
    await borrowerNFT.transfer(nftId, borrowerAddress, newOwner);

    // Verify new owner
```

```
    const owner = await borrowerNFT.getOwner(nftId);
    expect(owner).toBe(newOwner);

    // Verify loan protocol updated
    const loan = await loanProtocol.getLoan(loanId);
    expect(loan.borrower).toBe(newOwner);
  });

  it('should burn NFT when loan completes', async () => {
    const nftId = 1;

    // Repay loan
    await loanProtocol.repay(loanId);

    // Verify NFT burned
    const owner = await borrowerNFT.getOwner(nftId);
    expect(owner).toBeNull();
  });
});
```

---

7. Marketplace Functionality

7.1 Overview

The **secondary marketplace** enables users to trade NFT loan positions before maturity, providing: - **Exit liquidity** for lenders who want early exit - **Loan assumption** for borrowers who want to transfer debt - **Price discovery** for loan valuations - **Risk transfer** between market participants

---

7.2 Marketplace Architecture

```
    Frontend Marketplace UI
  - Browse Listings
  - Filter by Stablecoin/Role/APR
  - Create Listing
  - Purchase Listed NFT



  marketplace.clar (Smart Contract)
```

```
  - Listing management
  - Offer/bidding system
  - Purchase execution
  - Fee collection (future)




       NFT Contracts
  - borrower-nft.clar
  - lender-nft.clar
  - Transfer ownership
```

7.3 Marketplace Smart Contract

Contract: `marketplace.clar`

7.3.1 Data Structures

```
;; Listing Structure
(define-map listings
  {listing-id: uint}
  {
    nft-type: (string-ascii 10),  ;; "borrower" or "lender"
    nft-id: uint,
    loan-id: uint,
    seller: principal,
    asking-price: uint,
    payment-asset: principal,  ;; Stablecoin for payment (USDA/USDC/xUSD)
    status: (string-ascii 20),  ;; "active", "sold", "cancelled"
    created-at: uint,
    expires-at: uint
  }
)

;; Listing counter
(define-data-var last-listing-id uint u0)

;; Offers (optional bidding system)
(define-map offers
  {listing-id: uint, offer-id: uint}
  {
    bidder: principal,
    offer-amount: uint,
```

```
    payment-asset: principal,
    status: (string-ascii 20),  ;; "pending", "accepted", "rejected"
    created-at: uint
  }
)

;; Offer counter per listing
(define-map listing-offer-count
  {listing-id: uint}
  {count: uint}
)
```

7.3.2 Create Listing

```
(define-public (create-listing
    (nft-type (string-ascii 10))
    (nft-id uint)
    (asking-price uint)
    (payment-asset principal)
    (duration-blocks uint)
  )
  (let
    (
      (listing-id (+ (var-get last-listing-id) u1))
      (loan-id (get-loan-id-for-nft nft-type nft-id))
      (nft-owner (get-nft-owner nft-type nft-id))
    )

    ;; Verify caller owns the NFT
    (asserts! (is-eq tx-sender nft-owner) err-not-owner)

    ;; Verify asking price > 0
    (asserts! (> asking-price u0) err-invalid-price)

    ;; Verify payment asset is whitelisted stablecoin
    (asserts! (is-whitelisted-stablecoin payment-asset) err-invalid-stablecoin)

    ;; Create listing
    (map-set listings
      {listing-id: listing-id}
      {
        nft-type: nft-type,
        nft-id: nft-id,
        loan-id: loan-id,
        seller: tx-sender,
        asking-price: asking-price,
```

```
        payment-asset: payment-asset,
        status: "active",
        created-at: block-height,
        expires-at: (+ block-height duration-blocks)
      }
    )

    ;; Increment counter
    (var-set last-listing-id listing-id)

    (print {
      event: "listing-created",
      listing-id: listing-id,
      nft-type: nft-type,
      nft-id: nft-id,
      asking-price: asking-price,
      payment-asset: payment-asset
    })

    (ok listing-id)
  )
)
```

7.3.3 Purchase Listing (Buy Now)

```
(define-public (purchase-listing (listing-id uint))
  (let
    (
      (listing (unwrap! (map-get? listings {listing-id: listing-id}) err-not-found))
      (seller (get seller listing))
      (asking-price (get asking-price listing))
      (payment-asset (get payment-asset listing))
      (nft-type (get nft-type listing))
      (nft-id (get nft-id listing))
    )

    ;; Verify listing is active
    (asserts! (is-eq (get status listing) "active") err-listing-not-active)

    ;; Verify not expired
    (asserts! (<= block-height (get expires-at listing)) err-listing-expired)

    ;; Verify buyer is not seller
    (asserts! (not (is-eq tx-sender seller)) err-cannot-buy-own-listing)

    ;; Transfer payment from buyer to seller
```

```
    (try! (contract-call? payment-asset transfer asking-price tx-sender seller none))

    ;; Transfer NFT from seller to buyer
    (if (is-eq nft-type "borrower")
      (try! (contract-call? .borrower-nft transfer nft-id seller tx-sender))
      (try! (contract-call? .lender-nft transfer nft-id seller tx-sender))
    )

    ;; Update listing status
    (map-set listings
      {listing-id: listing-id}
      (merge listing {status: "sold"})
    )

    (print {
      event: "listing-purchased",
      listing-id: listing-id,
      buyer: tx-sender,
      seller: seller,
      price: asking-price,
      nft-type: nft-type,
      nft-id: nft-id
    })

    (ok true)
  )
)
```

7.3.4 Cancel Listing

```
(define-public (cancel-listing (listing-id uint))
  (let
    (
      (listing (unwrap! (map-get? listings {listing-id: listing-id}) err-not-found))
      (seller (get seller listing))
    )

    ;; Verify caller is seller
    (asserts! (is-eq tx-sender seller) err-unauthorized)

    ;; Verify listing is active
    (asserts! (is-eq (get status listing) "active") err-listing-not-active)

    ;; Update status
    (map-set listings
      {listing-id: listing-id}
```

```
      (merge listing {status: "cancelled"})
    )

    (print {
      event: "listing-cancelled",
      listing-id: listing-id
    })

    (ok true)
  )
)
```

7.3.5 Make Offer (Optional)

```
(define-public (make-offer
    (listing-id uint)
    (offer-amount uint)
    (payment-asset principal)
  )
  (let
    (
      (listing (unwrap! (map-get? listings {listing-id: listing-id}) err-not-found))
      (offer-count (default-to {count: u0} (map-get? listing-offer-count {listing-id: listi
      (new-offer-id (+ (get count offer-count) u1))
    )

    ;; Verify listing active
    (asserts! (is-eq (get status listing) "active") err-listing-not-active)

    ;; Verify offer > 0
    (asserts! (> offer-amount u0) err-invalid-amount)

    ;; Verify payment asset matches listing
    (asserts! (is-eq payment-asset (get payment-asset listing)) err-asset-mismatch)

    ;; Create offer
    (map-set offers
      {listing-id: listing-id, offer-id: new-offer-id}
      {
        bidder: tx-sender,
        offer-amount: offer-amount,
        payment-asset: payment-asset,
        status: "pending",
        created-at: block-height
      }
    )
```

```
    ;; Update counter
    (map-set listing-offer-count
      {listing-id: listing-id}
      {count: new-offer-id}
    )

    (print {
      event: "offer-made",
      listing-id: listing-id,
      offer-id: new-offer-id,
      bidder: tx-sender,
      offer-amount: offer-amount
    })

    (ok new-offer-id)
  )
)
```

7.3.6 Accept Offer

```
(define-public (accept-offer (listing-id uint) (offer-id uint))
  (let
    (
      (listing (unwrap! (map-get? listings {listing-id: listing-id}) err-not-found))
      (offer (unwrap! (map-get? offers {listing-id: listing-id, offer-id: offer-id}) err-not
      (seller (get seller listing))
      (bidder (get bidder offer))
      (offer-amount (get offer-amount offer))
      (payment-asset (get payment-asset offer))
      (nft-type (get nft-type listing))
      (nft-id (get nft-id listing))
    )

    ;; Verify caller is seller
    (asserts! (is-eq tx-sender seller) err-unauthorized)

    ;; Verify listing active
    (asserts! (is-eq (get status listing) "active") err-listing-not-active)

    ;; Verify offer pending
    (asserts! (is-eq (get status offer) "pending") err-offer-not-pending)

    ;; Transfer payment from bidder to seller
    (try! (contract-call? payment-asset transfer offer-amount bidder seller none))
```

```
    ;; Transfer NFT from seller to bidder
    (if (is-eq nft-type "borrower")
      (try! (contract-call? .borrower-nft transfer nft-id seller bidder))
      (try! (contract-call? .lender-nft transfer nft-id seller bidder))
    )

    ;; Update listing
    (map-set listings
      {listing-id: listing-id}
      (merge listing {status: "sold"})
    )

    ;; Update offer
    (map-set offers
      {listing-id: listing-id, offer-id: offer-id}
      (merge offer {status: "accepted"})
    )

    (print {
      event: "offer-accepted",
      listing-id: listing-id,
      offer-id: offer-id,
      buyer: bidder,
      seller: seller,
      price: offer-amount
    })

    (ok true)
  )
)
```

7.4 Pricing Models

7.4.1 Lender NFT Valuation Formula for Fair Value:

```
Lender NFT Value = Expected Return × Discount Factor

Where:
- Expected Return = Repayment Amount (if borrower repays)
- Discount Factor = 1 / (1 + market_rate × days_remaining / 365)
- market_rate = Current market APR for similar loans
```

Example:

```
Loan Details:
- Lent: $50,000 USDA
```

```
- Expected Return: $50,800 USDA
- Days Remaining: 45 days
- Market APR: 6%

Fair Value = $50,800 / (1 + 0.06 × 45/365)
           = $50,800 / 1.0074
           = $50,425 USDA

Seller might list at: $50,400 (slight discount for quick sale)
```

7.4.2 Borrower NFT Valuation Formula for Debt Transfer Price:

```
Borrower NFT Value = -(Remaining Debt × Premium Factor)

Where:
- Remaining Debt = Repayment Amount - Borrow Amount
- Premium Factor = Urgency multiplier (1.0 - 1.5x)
```

Example:

```
Loan Details:
- Borrowed: $50,000 USDA
- Must Repay: $50,800 USDA
- Remaining Debt: $800 USDA
- Days Until Maturity: 10 days (urgent)
- Collateral Value: $75,000 (1.5 sBTC)

Debt Assumption Price = $800 × 1.2 (20% premium for urgency)
                      = $960 USDA

Buyer pays seller $960 USDA to take over the loan.
Buyer gets access to $50K borrowed funds + must repay $50,800.
Net: Buyer effectively paid $960 for $50K loan at 4% APR.
```

7.5 Frontend Marketplace UI

Browse Listings Page:

```
   NFT Marketplace


 Filters:
 [All (24)] [Lender (18)] [Borrower (6)]
 [USDA (16)] [USDC (6)] [xUSD (2)]
```

```
Sort:  Best Deal   APR   Time Left   Price


  Lender Position #42 (USDA)


 Lent: $50,000 USDA
 Will Receive: $50,800 USDA
 Your Profit: $800 USDA (3.2% APR)
 Days Remaining: 45 days

 Asking Price: $50,400 USDA
 Fair Value: $50,425 USDA (0.5% discount)

 Collateral: 1.5 sBTC (~$75K)
 Risk: Low (150% LTV)

 [Buy Now for 50,400 USDA]  [Make Offer]


  Borrower Position #38 (USDC)


 Borrowed: $30,000 USDC
 Must Repay: $31,200 USDC
 Remaining Debt: $1,200 USDC
 Days Remaining: 10 days (  Urgent)

 Transfer Price: $1,400 USDC
 (20% premium for urgency)

 Collateral: 1.0 sBTC (~$50K)
 Net Deal: Borrow $30K at 5% APR

 [Assume Debt for 1,400 USDC]  [Details]


[Load More (20 listings remaining)]
```

7.6 Marketplace Testing

Integration Tests:

```
describe('Marketplace', () => {
  it('should create listing for lender NFT', async () => {
    const lenderNFT = 1;
    const askingPrice = 50400; // USDA (6 decimals)

    // Create listing
    const listingId = await marketplace.createListing(
      'lender',
      lenderNFT,
      askingPrice,
      usdaContract,
      144 // 1 day expiry
    );

    expect(listingId).toBe(1);

    // Verify listing exists
    const listing = await marketplace.getListing(listingId);
    expect(listing.nftType).toBe('lender');
    expect(listing.askingPrice).toBe(askingPrice);
    expect(listing.status).toBe('active');
  });

  it('should purchase listing', async () => {
    const listingId = 1;
    const buyerBalanceBefore = await usda.getBalance(buyer);

    // Purchase
    await marketplace.purchaseListing(listingId, { from: buyer });

    // Verify payment transferred
    const buyerBalanceAfter = await usda.getBalance(buyer);
    expect(buyerBalanceBefore - buyerBalanceAfter).toBe(50400);

    // Verify NFT transferred
    const nftOwner = await lenderNFT.getOwner(1);
    expect(nftOwner).toBe(buyer);

    // Verify listing marked sold
    const listing = await marketplace.getListing(listingId);
    expect(listing.status).toBe('sold');
  });
```

```
  it('should make and accept offer', async () => {
    const listingId = 1;
    const offerAmount = 50300; // Lower than asking

    // Make offer
    const offerId = await marketplace.makeOffer(
      listingId,
      offerAmount,
      usdaContract,
      { from: buyer }
    );

    // Seller accepts offer
    await marketplace.acceptOffer(listingId, offerId, { from: seller });

    // Verify NFT transferred
    const nftOwner = await lenderNFT.getOwner(1);
    expect(nftOwner).toBe(buyer);
  });
});
```

---

8. Frontend Architecture

8.1 Technology Stack

Framework: Next.js 14 with App Router
Language: TypeScript
Styling: TailwindCSS
State Management: React Context + SWR for data fetching
Wallet Integration: @stacks/connect
Blockchain Interaction: @stacks/transactions

8.2 Multi-Stablecoin Components

Stablecoin Configuration

```
// lib/stablecoins.ts

export interface Stablecoin {
  symbol: 'USDA' | 'USDC' | 'xUSD';
  name: string;
  contractAddress: string;
  decimals: number;
  icon: string;
```

```
  description: string;
  isNative: boolean;  // Native to Stacks vs bridged
  color: string;      // For UI theming
}

export const STABLECOINS: Record<string, Stablecoin> = {
  USDA: {
    symbol: 'USDA',
    name: 'Arkadiko USD',
    contractAddress: 'SP2C2YFP12AJZB4MABJBAJ55XECVS7E4PMMZ89YZR.usda-token',
    decimals: 6,
    icon: '/icons/usda.svg',
    description: 'Native Stacks stablecoin, deepest liquidity',
    isNative: true,
    color: '#3B82F6'  // Blue
  },
  USDC: {
    symbol: 'USDC',
    name: 'USD Coin',
    contractAddress: 'SP3DX3H4FEYZJZ586MFBS25ZW3HZDMEW92260R2PR.usdc-token',
    decimals: 6,
    icon: '/icons/usdc.svg',
    description: 'Institutional standard, 1:1 USD reserves',
    isNative: false,
    color: '#2775CA'  // USDC Blue
  },
  xUSD: {
    symbol: 'xUSD',
    name: 'xUSD Stablecoin',
    contractAddress: 'SPXUSD1234567890ABCDEF.xusd-token',
    decimals: 6,
    icon: '/icons/xusd.svg',
    description: 'Alternative stablecoin option',
    isNative: false,
    color: '#10B981'  // Green
  }
};

export function getStablecoinByContract(contractAddress: string): Stablecoin | undefined {
  return Object.values(STABLECOINS).find(s => s.contractAddress === contractAddress);
}

export function formatStablecoinAmount(amount: bigint, symbol: string): string {
  const stablecoin = STABLECOINS[symbol];
  const divisor = BigInt(10 ** stablecoin.decimals);
  const dollars = Number(amount / divisor);
```

```
  const cents = Number((amount % divisor) / BigInt(10 ** (stablecoin.decimals - 2)));
  return `$${dollars.toLocaleString()}.${cents.toString().padStart(2, '0')}`;
}
```

Stablecoin Selector Component

```
// components/loans/StablecoinSelector.tsx

import React from 'react';
import { STABLECOINS, Stablecoin } from '@/lib/stablecoins';

interface StablecoinSelectorProps {
  selectedStablecoin: string;
  onSelect: (symbol: string) => void;
  userBalances: Record<string, bigint>;
  protocolLiquidity?: Record<string, bigint>;
  className?: string;
}

export function StablecoinSelector({
  selectedStablecoin,
  onSelect,
  userBalances,
  protocolLiquidity,
  className = ''
}: StablecoinSelectorProps) {
  return (
    <div className={`space-y-4 ${className}`}>
      <label className="block text-sm font-medium text-gray-700">
        Choose Stablecoin
      </label>

      <div className="space-y-3">
        {Object.values(STABLECOINS).map((coin) => {
          const isSelected = selectedStablecoin === coin.symbol;
          const userBalance = userBalances[coin.symbol] || BigInt(0);
          const hasBalance = userBalance > BigInt(0);

          return (
            <button
              key={coin.symbol}
              onClick={() => onSelect(coin.symbol)}
              className={`
                w-full p-4 rounded-lg border-2 transition-all text-left
                ${isSelected
                  ? 'border-blue-500 bg-blue-50'
```

```
        : 'border-gray-200 hover:border-gray-300 bg-white'
    }
  `}
>
  <div className="flex items-start space-x-4">
    {/* Icon */}
    <img
      src={coin.icon}
      alt={coin.symbol}
      className="w-10 h-10 rounded-full"
    />

    {/* Content */}
    <div className="flex-1">
      <div className="flex items-center justify-between mb-1">
        <h3 className="font-semibold text-lg">
          {coin.symbol}
          {coin.symbol === 'USDA' && (
            <span className="ml-2 text-xs bg-blue-100 text-blue-700 px-2 py-1 ro
              Recommended
            </span>
          )}
          {isSelected && (
            <span className="ml-2 text-blue-600"> </span>
          )}
        </h3>
      </div>

      <p className="text-sm text-gray-600 mb-2">
        {coin.description}
      </p>

      <div className="flex items-center justify-between text-sm">
        <div>
          <span className="text-gray-500">Your balance: </span>
          <span className={`font-medium ${hasBalance ? 'text-green-600' : 'text-
            {formatStablecoinAmount(userBalance, coin.symbol)} {coin.symbol}
            {hasBalance ? '  ' : ''}
          </span>
        </div>

        {protocolLiquidity && (
          <div className="text-gray-500">
            Liquidity: {formatStablecoinAmount(protocolLiquidity[coin.symbol] ||
          </div>
        )}
```

```
                </div>
              </div>
            </div>
          </button>
        );
      })}
    </div>

    {/* Helpful tip */}
    <div className="mt-4 p-3 bg-blue-50 rounded-lg">
      <p className="text-sm text-blue-800">
        <strong>Tip:</strong> Choose the stablecoin you already hold to avoid swapping.
        USDA has the best liquidity on Stacks.
      </p>
    </div>
  </div>
);
}
```

Loan Card with Stablecoin Display

```
// components/loans/LoanCard.tsx

import React from 'react';
import { Loan } from '@/types/loan';
import { getStablecoinByContract, formatStablecoinAmount } from '@/lib/stablecoins';

interface LoanCardProps {
  loan: Loan;
  currentBid?: Bid;
  onBidClick: () => void;
}

export function LoanCard({ loan, currentBid, onBidClick }: LoanCardProps) {
  const stablecoin = getStablecoinByContract(loan.borrowAsset);
  if (!stablecoin) return null;

  const currentAmount = currentBid ? currentBid.amount : loan.maxRepayment;
  const impliedAPR = calculateImpliedAPR(
    loan.borrowAmount,
    currentAmount,
    loan.loanDurationBlocks
  );

  const blocksRemaining = loan.auctionEndBlock - currentBlock;
  const hoursRemaining = (blocksRemaining * 10) / 60;
```

```
  return (
    <div className="border rounded-lg p-6 bg-white hover:shadow-lg transition-shadow">
      {/* Stablecoin Badge */}
      <div className="flex items-center justify-between mb-4">
        <div className="flex items-center space-x-2">
          <img
            src={stablecoin.icon}
            alt={stablecoin.symbol}
            className="w-6 h-6"
          />
          <span
            className="font-semibold text-sm px-2 py-1 rounded"
            style={{ backgroundColor: `${stablecoin.color}20`, color: stablecoin.color }}
          >
            {stablecoin.symbol} Loan
          </span>
        </div>

        <span className="text-sm text-gray-500">
          {hoursRemaining.toFixed(1)}h left
        </span>
      </div>

      {/* Loan Details */}
      <h3 className="text-lg font-semibold mb-3">Loan #{loan.loanId}</h3>

      <div className="space-y-2 mb-4">
        <div>
          <span className="text-sm text-gray-600">Borrow Amount</span>
          <p className="text-xl font-semibold">
            {formatStablecoinAmount(loan.borrowAmount, stablecoin.symbol)} {stablecoin.symbo
          </p>
        </div>

        <div>
          <span className="text-sm text-gray-600">
            {currentBid ? 'Current Lowest Bid' : 'Borrower\'s Max'}
          </span>
          <p className="text-lg font-medium" style={{ color: stablecoin.color }}>
            {formatStablecoinAmount(currentAmount, stablecoin.symbol)} {stablecoin.symbol}
            <span className="text-sm text-gray-600 ml-2">
              ({impliedAPR.toFixed(2)}% APR)
            </span>
          </p>
        </div>
```

```
          <div>
            <span className="text-sm text-gray-600">Collateral</span>
            <p className="font-medium">
              {formatBTC(loan.collateralAmount)} BTC
            </p>
          </div>

          {currentBid && (
            <div className="flex items-center space-x-1 text-sm text-gray-600">
              <span> </span>
              <span>{currentBid.bidCount || 0} bids placed</span>
            </div>
          )}
        </div>

        {/* Action Button */}
        <button
          onClick={onBidClick}
          className="w-full py-2 px-4 rounded-lg font-medium transition-colors"
          style={{
            backgroundColor: stablecoin.color,
            color: 'white'
          }}
        >
          Place Bid in {stablecoin.symbol}
        </button>
      </div>
    );
}
```

Balance Warning Component

```
// components/loans/StablecoinBalanceWarning.tsx

import React from 'react';
import { STABLECOINS } from '@/lib/stablecoins';

interface StablecoinBalanceWarningProps {
  requiredStablecoin: string;
  requiredAmount: bigint;
  userBalance: bigint;
  onSwap?: () => void;
  onChangeLoan?: () => void;
}
```

```tsx
export function StablecoinBalanceWarning({
  requiredStablecoin,
  requiredAmount,
  userBalance,
  onSwap,
  onChangeLoan
}: StablecoinBalanceWarningProps) {
  const stablecoin = STABLECOINS[requiredStablecoin];
  const shortfall = requiredAmount - userBalance;

  if (userBalance >= requiredAmount) return null;

  return (
    <div className="p-4 bg-yellow-50 border border-yellow-200 rounded-lg">
      <div className="flex items-start space-x-3">
        <span className="text-2xl"> </span>
        <div className="flex-1">
          <h4 className="font-semibold text-yellow-900 mb-2">
            Insufficient {stablecoin.symbol} Balance
          </h4>

          <div className="space-y-1 text-sm text-yellow-800 mb-3">
            <p>This loan requires: <strong>{formatStablecoinAmount(requiredAmount, requiredS
            <p>Your balance: <strong>{formatStablecoinAmount(userBalance, requiredStablecoin
            <p>You need: <strong>{formatStablecoinAmount(shortfall, requiredStablecoin)} mor
          </div>

          <div className="text-sm text-yellow-800 mb-3">
            <p className="font-medium mb-1">Options:</p>
            <ul className="list-disc list-inside space-y-1">
              <li>Swap other stablecoins to {requiredStablecoin} on Stacks DEX</li>
              <li>Bridge {requiredStablecoin} from another chain</li>
              <li>Choose a different loan in a stablecoin you hold</li>
            </ul>
          </div>

          <div className="flex space-x-2">
            {onSwap && (
              <button
                onClick={onSwap}
                className="px-4 py-2 bg-yellow-600 text-white rounded-lg text-sm font-mediu
              >
                Swap to {requiredStablecoin}
              </button>
            )}
            {onChangeLoan && (
```

```
              <button
                onClick={onChangeLoan}
                className="px-4 py-2 bg-white border border-yellow-600 text-yellow-600 round
              >
                Browse Other Loans
              </button>
            )}
          </div>
        </div>
      </div>
    </div>
  );
}
```

Stablecoin Filter Component

```
// components/marketplace/StablecoinFilter.tsx

import React from 'react';
import { STABLECOINS } from '@/lib/stablecoins';

interface StablecoinFilterProps {
  selectedStablecoin: string | 'ALL';
  onSelect: (symbol: string | 'ALL') => void;
  counts: Record<string, number>;
}

export function StablecoinFilter({ selectedStablecoin, onSelect, counts }: StablecoinFilterP
  const totalCount = Object.values(counts).reduce((sum, count) => sum + count, 0);

  return (
    <div className="flex space-x-2 overflow-x-auto pb-2">
      {/* All */}
      <button
        onClick={() => onSelect('ALL')}
        className={`
          px-4 py-2 rounded-lg font-medium text-sm whitespace-nowrap transition-colors
          ${selectedStablecoin === 'ALL'
            ? 'bg-gray-800 text-white'
            : 'bg-gray-100 text-gray-700 hover:bg-gray-200'
          }
        `}
      >
        All ({totalCount})
      </button>
```

```
      {/* Each stablecoin */}
      {Object.values(STABLECOINS).map((coin) => {
        const count = counts[coin.symbol] || 0;
        const isSelected = selectedStablecoin === coin.symbol;

        return (
          <button
            key={coin.symbol}
            onClick={() => onSelect(coin.symbol)}
            className={`
              flex items-center space-x-2 px-4 py-2 rounded-lg font-medium text-sm whitespac
              ${isSelected
                ? 'text-white'
                : 'bg-gray-100 text-gray-700 hover:bg-gray-200'
              }
            `}
            style={isSelected ? { backgroundColor: coin.color } : {}}
          >
            <img src={coin.icon} alt={coin.symbol} className="w-5 h-5" />
            <span>{coin.symbol}</span>
            <span className={isSelected ? 'opacity-80' : 'text-gray-500'}>
              ({count})
            </span>
          </button>
        );
      })}
    </div>
  );
}
```

---

9. API Specifications

9.1 Overview

The protocol interacts with external APIs for blockchain data, real-time updates, and NFT metadata. This section specifies all API integrations.

9.2 Stacks Blockchain API

Provider: Hiro Systems API (primary)
Base URL: `https://api.mainnet.hiro.so`
Documentation: https://docs.hiro.so/api

9.2.1 Contract Read-Only Calls **Endpoint**: `POST /v2/contracts/call-read/{contract_address}/{c`

Example: Get loan details

```
// Get loan by ID
const response = await fetch(
  'https://api.mainnet.hiro.so/v2/contracts/call-read/SP2ABC...XYZ/loan-protocol/get-loan',
  {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      sender: 'SP2ABC...XYZ',
      arguments: [
        '0x0100000000000000000000000000000001' // loan-id: u1 in Clarity
      ]
    })
  }
);

const data = await response.json();
// data.result contains Clarity value
```

Response:

```
{
  "okay": true,
  "result": "0x0703...", // Clarity encoded tuple
  "cause": null
}
```

9.2.2 Transaction Broadcast **Endpoint**: `POST /v2/transactions`

Example: Create loan auction

```
import { makeContractCall, broadcastTransaction } from '@stacks/transactions';

const txOptions = {
  contractAddress: 'SP2ABC...XYZ',
  contractName: 'loan-protocol',
  functionName: 'create-loan-auction',
  functionArgs: [
    uintCV(150000000), // collateral-amount
    principalCV('SP2ABC...XYZ.usda-token'), // borrow-asset
    uintCV(50000000000), // borrow-amount (50K USDA, 6 decimals)
    uintCV(53500000000), // max-repayment
    uintCV(8640), // duration-blocks (60 days)
```

```
    uintCV(144), // auction-duration (24 hours)
  ],
  senderKey: privateKey,
  network: new StacksMainnet(),
  anchorMode: AnchorMode.Any,
  postConditionMode: PostConditionMode.Deny,
  postConditions: [
    // Must lock 1.5 sBTC
    makeStandardSTXPostCondition(
      senderAddress,
      FungibleConditionCode.Equal,
      150000000
    )
  ]
};

const transaction = await makeContractCall(txOptions);
const broadcastResponse = await broadcastTransaction(transaction, network);
// broadcastResponse.txid is the transaction ID
```

Response:

```
{
  "txid": "0x1234567890abcdef...",
  "error": null
}
```

9.2.3 Transaction Status **Endpoint**: `GET /extended/v1/tx/{txid}`

Example: Check transaction status

```
const txid = '0x1234...';
const response = await fetch(
  `https://api.mainnet.hiro.so/extended/v1/tx/${txid}`
);

const txData = await response.json();
console.log(txData.tx_status); // "success", "pending", "abort_by_response"
```

Response:

```
{
  "tx_id": "0x1234...",
  "tx_status": "success",
  "tx_type": "contract_call",
  "block_height": 145234,
  "canonical": true,
  "contract_call": {
```

```
    "contract_id": "SP2ABC...XYZ.loan-protocol",
    "function_name": "create-loan-auction",
    "function_signature": "(define-public ...)"
  },
  "events": [
    {
      "event_index": 0,
      "event_type": "stx_lock",
      "stx_lock_event": {
        "locked_amount": "150000000",
        "unlock_height": "145378",
        "locked_address": "SP2ABC...XYZ"
      }
    }
  ]
}
```

9.2.4 Account Balances **Endpoint**: `GET /extended/v1/address/{principal}/balances`

Example: Get user balances

```
const address = 'SP2ABC...XYZ';
const response = await fetch(
  `https://api.mainnet.hiro.so/extended/v1/address/${address}/balances`
);

const balances = await response.json();
```

Response:

```
{
  "stx": {
    "balance": "10000000000",
    "total_sent": "5000000",
    "total_received": "10005000000",
    "locked": "150000000"
  },
  "fungible_tokens": {
    "SP2ABC...XYZ.sbtc-token::sbtc": {
      "balance": "200000000",
      "total_sent": "0",
      "total_received": "200000000"
    },
    "SP2ABC...XYZ.usda-token::usda": {
      "balance": "65000000000",
      "total_sent": "10000000000",
      "total_received": "75000000000"
```

```
    }
  },
  "non_fungible_tokens": {
    "SP2ABC...XYZ.borrower-nft::borrower-nft": {
      "count": "2",
      "total_sent": "0",
      "total_received": "2"
    }
  }
}
```

9.3 Custom Backend API

Purpose: Aggregate blockchain data, provide computed values, serve NFT metadata
Base URL: `https://api.bitcoinlending.xyz`
Tech Stack: Node.js, Express, PostgreSQL

9.3.1 Get Loans (with Filters) **Endpoint**: `GET /api/v1/loans`

Query Parameters: - `status`: Filter by status ("auction", "active", "completed", "defaulted") - `stablecoin`: Filter by stablecoin ("usda", "usdc", "xusd") - `sort`: Sort by ("apr", "amount", "time-left") - `limit`: Number of results (default: 20, max: 100) - `offset`: Pagination offset

Example Request:

```
GET /api/v1/loans?status=auction&stablecoin=usda&sort=apr&limit=10
```

Response:

```
{
  "loans": [
    {
      "loanId": 42,
      "borrower": "SP2ABC...XYZ",
      "collateralAmount": "150000000",
      "collateralAmountBTC": "1.5",
      "borrowAsset": "SP2ABC...XYZ.usda-token",
      "borrowAssetSymbol": "USDA",
      "borrowAmount": "50000000000",
      "borrowAmountFormatted": "50000",
      "maxRepayment": "53500000000",
      "maxRepaymentFormatted": "53500",
      "durationBlocks": 8640,
      "durationDays": 60,
      "auctionDuration": 144,
```

```
      "auctionDurationHours": 24,
      "status": "auction",
      "currentBid": {
        "bidder": "SP2DEF...ABC",
        "repaymentAmount": "51200000000",
        "repaymentAmountFormatted": "51200",
        "impliedAPR": 4.8,
        "bidTime": "2026-01-12T14:30:00Z"
      },
      "bidCount": 4,
      "auctionEndsAt": "2026-01-13T14:30:00Z",
      "timeRemainingSeconds": 28800,
      "ltv": 66.7,
      "createdAt": "2026-01-12T14:30:00Z"
    }
  ],
  "total": 12,
  "limit": 10,
  "offset": 0
}
```


9.3.2 Get Loan by ID

Endpoint: `GET /api/v1/loans/{loanId}`

Example Request:

```
GET /api/v1/loans/42
```

Response: (Same as single loan object above, with additional details)


```
{
  "loanId": 42,
  "borrower": "SP2ABC...XYZ",
  "lender": "SP2DEF...ABC",
  "collateralAmount": "150000000",
  "collateralAmountBTC": "1.5",
  "collateralValueUSD": 75000,
  "borrowAsset": "SP2ABC...XYZ.usda-token",
  "borrowAssetSymbol": "USDA",
  "borrowAmount": "50000000000",
  "borrowAmountFormatted": "50000",
  "repaymentAmount": "51200000000",
  "repaymentAmountFormatted": "51200",
  "durationBlocks": 8640,
  "durationDays": 60,
  "maturityBlock": 153880,
  "maturityDate": "2026-03-13T14:30:00Z",
  "status": "active",
```

```
  "apr": 4.8,
  "ltv": 66.7,
  "borrowerNFT": 42,
  "lenderNFT": 43,
  "createdAt": "2026-01-12T14:30:00Z",
  "activatedAt": "2026-01-13T14:30:00Z",
  "repaidAt": null,
  "bidHistory": [
    {
      "bidder": "SP2ABC...XYZ",
      "repaymentAmount": "53500000000",
      "impliedAPR": 7.0,
      "bidTime": "2026-01-13T08:00:00Z"
    },
    {
      "bidder": "SP2BCD...YZA",
      "repaymentAmount": "52800000000",
      "impliedAPR": 5.6,
      "bidTime": "2026-01-13T10:00:00Z"
    },
    {
      "bidder": "SP2CDE...ZAB",
      "repaymentAmount": "51800000000",
      "impliedAPR": 3.6,
      "bidTime": "2026-01-13T12:00:00Z"
    },
    {
      "bidder": "SP2DEF...ABC",
      "repaymentAmount": "51200000000",
      "impliedAPR": 2.4,
      "bidTime": "2026-01-13T14:00:00Z"
    }
  ]
}
```


9.3.3 Get User Loans

Endpoint: `GET /api/v1/users/{address}/loans`

Query Parameters: - `role`: "borrower" or "lender" - `status`: Filter by status

Example Request:

```
GET /api/v1/users/SP2ABC...XYZ/loans?role=borrower&status=active
```

Response:


```
{
  "address": "SP2ABC...XYZ",
  "role": "borrower",
```

```
  "loans": [
    {
      "loanId": 42,
      "stablecoin": "USDA",
      "borrowed": "50000",
      "mustRepay": "51200",
      "daysRemaining": 45,
      "status": "active"
    }
  ],
  "totalLoans": 1,
  "totalBorrowed": "50000",
  "totalRepaymentDue": "51200"
}
```

9.3.4 Get Protocol Stats **Endpoint**: `GET /api/v1/stats`

Response:

```
{
  "totalVolume": "1250000",
  "totalVolumeByStablecoin": {
    "USDA": "812500",
    "USDC": "350000",
    "xUSD": "87500"
  },
  "activeLoans": 18,
  "activeLoansByStablecoin": {
    "USDA": 12,
    "USDC": 5,
    "xUSD": 1
  },
  "completedLoans": 47,
  "defaultedLoans": 1,
  "defaultRate": 2.1,
  "averageAPR": 6.8,
  "averageAPRByStablecoin": {
    "USDA": 6.2,
    "USDC": 8.1,
    "xUSD": 7.5
  },
  "totalCollateralLocked": "45.5",
  "totalCollateralLockedUSD": "2275000",
  "uniqueBorrowers": 23,
  "uniqueLenders": 58,
  "last24HourVolume": "125000",
```

```
  "updatedAt": "2026-01-12T14:30:00Z"
}
```

9.3.5 NFT Metadata **Endpoint**: `GET /api/v1/nft/{role}/{nftId}`

Parameters: - `role`: "borrower" or "lender" - `nftId`: NFT ID

Example Request:

```
GET /api/v1/nft/lender/42
```

Response: (See Section 6.3.5 for full format)

```
{
  "name": "Bitcoin Loan Position #42 (Lender)",
  "description": "Lender position for loan #42...",
  "image": "https://api.bitcoinlending.xyz/nft/lender/42.png",
  "attributes": [...]
}
```

9.3.6 NFT Image **Endpoint**: `GET /nft/{role}/{nftId}.png`

Returns: PNG image (500x500px)

9.4 WebSocket API (Real-Time Updates)

Purpose: Push real-time updates for auctions, bids, transactions
Endpoint: `wss://api.bitcoinlending.xyz/ws`

9.4.1 Connection

```
const ws = new WebSocket('wss://api.bitcoinlending.xyz/ws');

ws.onopen = () => {
  console.log('WebSocket connected');

  // Subscribe to specific loan
  ws.send(JSON.stringify({
    action: 'subscribe',
    channel: 'loan',
    loanId: 42
  }));

  // Subscribe to all auctions
  ws.send(JSON.stringify({
    action: 'subscribe',
    channel: 'auctions'
```

```
  }));
};

ws.onmessage = (event) => {
  const message = JSON.parse(event.data);
  handleMessage(message);
};
```

9.4.2 Message Types New Bid Event:

```
{
  "type": "bid_placed",
  "loanId": 42,
  "bidder": "SP2DEF...ABC",
  "repaymentAmount": "51200000000",
  "impliedAPR": 4.8,
  "timestamp": "2026-01-13T14:00:00Z"
}
```

Auction Finalized Event:

```
{
  "type": "auction_finalized",
  "loanId": 42,
  "winner": "SP2DEF...ABC",
  "repaymentAmount": "51200000000",
  "apr": 4.8,
  "timestamp": "2026-01-13T14:30:00Z"
}
```

Loan Repaid Event:

```
{
  "type": "loan_repaid",
  "loanId": 42,
  "borrower": "SP2ABC...XYZ",
  "lender": "SP2DEF...ABC",
  "repaymentAmount": "51200000000",
  "timestamp": "2026-03-13T10:15:00Z"
}
```

Marketplace Listing Event:

```
{
  "type": "listing_created",
  "listingId": 12,
  "nftType": "lender",
  "nftId": 42,
  "askingPrice": "50400000000",
```

```
  "stablecoin": "USDA",
  "timestamp": "2026-01-15T12:00:00Z"
}
```

9.5 Rate Limiting

API Rate Limits: - Unauthenticated: 100 requests/minute - Authenticated: 1000 requests/minute - WebSocket: 50 subscriptions per connection

Headers:

```
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 87
X-RateLimit-Reset: 1673456789
```

Rate Limit Exceeded Response:

```
{
  "error": "Rate limit exceeded",
  "limit": 100,
  "retryAfter": 42
}
```

9.6 Error Handling

Standard Error Response:

```
{
  "error": {
    "code": "LOAN_NOT_FOUND",
    "message": "Loan with ID 999 does not exist",
    "details": {
      "loanId": 999
    }
  }
}
```

Error Codes: - `LOAN_NOT_FOUND` (404) - `INVALID_STABLECOIN` (400) - `INSUFFICIENT_BALANCE` (400) - `AUCTION_EXPIRED` (400) - `UNAUTHORIZED` (401) - `RATE_LIMIT_EXCEEDED` (429) - `INTERNAL_ERROR` (500)

9.7 API Testing

Integration Tests:

```javascript
describe('API', () => {
  it('should fetch loans with filters', async () => {
    const response = await fetch(
      'https://api.bitcoinlending.xyz/api/v1/loans?stablecoin=usda&status=auction'
    );

    expect(response.status).toBe(200);

    const data = await response.json();
    expect(data.loans).toBeArray();
    expect(data.loans[0].borrowAssetSymbol).toBe('USDA');
    expect(data.loans[0].status).toBe('auction');
  });

  it('should handle WebSocket updates', (done) => {
    const ws = new WebSocket('wss://api.bitcoinlending.xyz/ws');

    ws.onopen = () => {
      ws.send(JSON.stringify({
        action: 'subscribe',
        channel: 'loan',
        loanId: 42
      }));
    };

    ws.onmessage = (event) => {
      const message = JSON.parse(event.data);
      if (message.type === 'bid_placed') {
        expect(message.loanId).toBe(42);
        ws.close();
        done();
      }
    };
  });
});
```

---

10. Security Considerations

10.1 Multi-Stablecoin Security

Stablecoin Validation:

```
;; CRITICAL: Always validate stablecoin is approved
(asserts! (is-stablecoin-approved borrow-asset) ERR_STABLECOIN_NOT_APPROVED)
```

```
;; Prevent malicious stablecoin contracts
;; Whitelist only audited, established stablecoins
```

Balance Verification:

```
;; Check lender has sufficient balance BEFORE accepting bid
;; Prevents bidders from winning without ability to fund
(let ((bidder-balance (contract-call? stablecoin-contract get-balance tx-sender)))
  (asserts! (>= bidder-balance amount) ERR_INSUFFICIENT_STABLECOIN)
)
```

Dynamic Contract Calls Safety:

```
;; Dynamic calls to stablecoin contracts must be safe
;; Only call approved contracts from whitelist
;; Use try! to handle potential failures gracefully

(try! (contract-call? stablecoin-contract transfer
  amount
  from
  to
  none))
```

Potential Attack Vectors:

Attack	Risk	Mitigation
Malicious stablecoin	Attacker deploys fake token	Whitelist only audited tokens
Reentrancy	Not possible in Clarity	N/A (Clarity prevents)
Wrong stablecoin bid	Bid in USDC for USDA loan	Balance check at bid time
Insufficient balance	Winner can't fund loan	Balance check before accepting bid
Stablecoin depeg	Stablecoin loses peg	User risk (diversification helps)

10.2 Stablecoin Depeg Considerations

Risk: If a stablecoin loses its peg (e.g., USDA drops to $0.95), loans denominated in that stablecoin are affected.

Impact: - Borrower benefits (repays less real value) - Lender loses (receives less real value) - Collateral (BTC) maintains independent value

Mitigations: - Support multiple stablecoins (diversification) - Users choose their own stablecoin (informed risk) - Protocol remains neutral (no forced stablecoin) - Emergency whitelist removal if stablecoin fails completely

Example Scenario:

```
Loan: Borrow 50,000 USDA, repay 51,000 USDA
If USDA depegs to $0.90:
- Borrower owes 51,000 USDA = $45,900 real value (wins)
- Lender receives 51,000 USDA = $45,900 (loses $4,100)
- BUT: This is user's choice and risk
- BTC collateral unaffected
```

User Protection: - Clear documentation of risks - Ability to choose stablecoin - Diversification across multiple stablecoins - Market determines which stablecoins succeed

---

10.3 Professional Security Audit (D1.1 - Enhanced Scope)

Audit Budget: **$88,000** (increased from $78,000)
Reason for Increase: Enhanced scope to cover multi-stablecoin complexity

10.3.1 Audit Scope **Primary Audit** ($68,000): - Core lending protocol logic - Competitive bidding auction mechanism - sBTC collateral lockup and release - NFT minting and burning - Loan lifecycle (create, bid, finalize, repay, default) - Access control and admin functions - Edge cases and attack vectors

Multi-Stablecoin Audit ($10,000 - **NEW**): - Stablecoin whitelist management - Dynamic stablecoin contract calls - Balance validation across different stablecoins - Cross-stablecoin scenario testing - Potential race conditions with multiple stablecoins - Stablecoin depeg scenarios

Follow-up Re-audit ($10,000): - Review fixes implemented from initial audit - Verify all high/critical findings resolved - Final signoff before mainnet deployment

10.3.2 Audit Timeline

```
Month 1 (Week 3-4): Audit RFP sent to audit firms
Month 2 (Week 1): Select audit firm, kick off
Month 2 (Week 2-4): Primary audit in progress
Month 3 (Week 1): Multi-stablecoin focus review
Month 3 (Week 2): Initial audit report received
Month 3 (Week 3): Development team addresses findings
Month 3 (Week 4): Follow-up re-audit
Month 4 (Week 1): Final audit report published
```

Total Audit Duration: 6-7 weeks
Total Audit Cost: $88,000

10.3.3 Audit Firm Selection Criteria **Preferred Audit Firms** (ranked by experience with Stacks/Clarity): 1. **CoinFabrik** - Extensive Clarity audit experience 2. **Least Authority** - Strong DeFi security background 3. **Trail of Bits** - General smart contract expertise 4. **Halborn** - Multi-chain audit experience

Selection Criteria: - Prior experience auditing Clarity contracts (required) - Experience with DeFi lending protocols (preferred) - Availability within project timeline (required) - Cost within $88K budget (required) - Willingness to focus on multi-stablecoin logic (required)

10.3.4 Audit Deliverables **Required Deliverables**: - [ ] Comprehensive audit report (PDF) - [ ] List of findings with severity ratings: - **Critical**: Immediate fund risk - **High**: Significant functionality broken - **Medium**: Edge case issues - **Low**: Code quality improvements - **Informational**: Best practice suggestions - [ ] Remediation recommendations for each finding - [ ] Follow-up review after fixes - [ ] Public publication of final audit report

Acceptance Criteria: - Zero critical vulnerabilities remaining - All high-severity issues resolved - Medium/low issues either fixed or accepted with documented risk - Audit firm provides final sign-off - Report published on protocol website and GitHub

10.3.5 Budget Breakdown

Audit Component	Cost	Duration	Focus
Primary Audit	$68,000	4 weeks	Core contracts, auction, NFTs
Multi-Stablecoin Review	$10,000	1 week	Whitelist, dynamic calls, validation
Follow-up Re-audit	$10,000	1 week	Fix verification, final signoff
TOTAL	**$88,000**	**6 weeks**	**Comprehensive security**

Why Enhanced Scope is Critical: - Multi-stablecoin support adds significant complexity - Dynamic contract calls require careful security review - Bal-

ance validation across 3 stablecoins creates new attack vectors - $10K investment prevents potential $100K+ losses from bugs - Regulatory compliance may require thorough audit in future

---

10.4 Additional Security Measures

10.4.1 Bug Bounty Program **Budget**: $50,000+ in STX/sBTC rewards
Timeline: Launches Week 2 after mainnet deployment
Platform: Immunefi or direct submission

Reward Tiers: - **Critical** (funds at risk): $25,000 - $50,000 - **High** (functionality broken): $5,000 - $15,000 - **Medium** (edge case issues): $1,000 - $5,000 - **Low** (code quality): $100 - $1,000

Scope: - Smart contracts (loan-protocol, NFTs, marketplace) - Frontend vulnerabilities (XSS, CSRF) - API security issues - Multi-stablecoin specific bugs

Out of Scope: - Social engineering - Physical security - Third-party dependencies (unless exploitable through our code)

10.4.2 Emergency Pause Mechanism **Circuit Breaker** (admin-only):

```
;; Emergency pause flag
(define-data-var protocol-paused bool false)

;; Check before every critical operation
(define-read-only (is-paused)
  (var-get protocol-paused)
)

;; Admin can pause protocol
(define-public (emergency-pause)
  (begin
    (asserts! (is-admin tx-sender) ERR_UNAUTHORIZED)
    (var-set protocol-paused true)
    (print {event: "emergency-pause", pauser: tx-sender})
    (ok true)
  )
)

;; Admin can unpause after fix deployed
(define-public (unpause-protocol)
  (begin
    (asserts! (is-admin tx-sender) ERR_UNAUTHORIZED)
    (var-set protocol-paused false)
    (print {event: "protocol-unpaused", unpauser: tx-sender})
```

```
    (ok true)
  )
)
```

What Pausing Stops: - New loan creation - Placing new bids - Finalizing auctions

What Pausing Does NOT Stop (user protection): - Existing loan repayments - Collateral claims (for defaulted loans) - NFT transfers

10.4.3 Admin Key Management Multi-Signature Requirements: - 3-of-5 multisig for critical operations - Admin actions require 48-hour timelock - Transparency: All admin actions logged on-chain

Admin Addresses: - Admin 1: Lead Developer (deployer) - Admin 2: Co-founder - Admin 3: Security Advisor - Admin 4: Community Representative (elected) - Admin 5: Foundation Board Member

Sensitive Operations Requiring Multisig: - Adding/removing stablecoins from whitelist - Emergency pause/unpause - Protocol parameter changes (if any) - Contract upgrades (if any)

10.4.4 Monitoring & Alerts Real-Time Monitoring: - Transaction success/failure rates - Gas costs (unexpected spikes) - Contract call patterns (unusual activity) - Balance changes (large withdrawals) - Error frequency by error type

Automated Alerts (PagerDuty): - **P1 (Critical)**: Any transaction failure, emergency pause triggered - **P2 (High)**: >5 failures in 1 hour, >10% drop in success rate - **P3 (Medium)**: Unusual transaction patterns, high gas costs

Security Dashboard (Grafana): - Real-time transaction monitoring - Error logs with stack traces - Balance discrepancies - Stablecoin distribution anomalies

---

10.5 Security Testing (Integrated with D1.10)

Security-Focused Test Cases:

```
;; Test: Prevent malicious stablecoin
(define-public (test-reject-malicious-token)
  (let ((result (create-loan-auction
                 SBTC_CONTRACT
                 u150000000
                 .fake-evil-token   ;; Not whitelisted
                 u50000000000
                 u53500000000
                 u8640
                 u144)))
```

```
    (asserts! (is-err result))
    (ok true)
  )
)

;; Test: Prevent bid without sufficient balance
(define-public (test-prevent-underfunded-bid)
  (let ((loan-id (try! (create-test-auction))))
    ;; Bidder has only 1 USDA, tries to bid 50,000
    (try! (mint-usda bidder u1000000))
    (let ((result (place-bid loan-id u50000000000)))
      (asserts! (is-err result))
      (asserts! (is-eq result ERR_INSUFFICIENT_STABLECOIN))
      (ok true)
    )
  )
)

;; Test: Verify stablecoin transfer happens atomically
(define-public (test-atomic-stablecoin-transfer)
  (let ((loan-id (try! (create-and-finalize-test-auction))))
    ;; Verify USDA transferred from lender to borrower
    (let ((borrower-balance (contract-call? .usda-token get-balance borrower))
          (lender-balance (contract-call? .usda-token get-balance lender)))
      (asserts! (is-eq borrower-balance u50000000000))
      (asserts! (<= lender-balance u10000000000)) ;; Lender had 60K, now has 10K
      (ok true)
    )
  )
)
```

Fuzz Testing: - Random inputs to all public functions - Randomized stablecoin selection - Random bid amounts (within valid ranges) - Random timing (auction duration, blocks)

Integration with D1.10 ($8,000 testing budget): - Security-focused test cases included in D1.10 - Stablecoin security scenarios tested comprehensively - 15+ multi-stablecoin security test cases

---

10.6 Security Best Practices Followed

Clarity-Specific Security: - No loops (prevents infinite gas attacks) - No recursion (prevents stack overflow) - Integer arithmetic only (no floating point imprecision) - Explicit error handling with `try!` and `unwrap!` - Post-conditions for critical transfers - Read-only functions for all queries

DeFi Best Practices: - Checks-effects-interactions pattern - Minimum balance requirements enforced - No hidden fees or surprise deductions - Transparent pricing (auction mechanism) - User controls their own assets (non-custodial)

Multi-Stablecoin Specific: - Whitelist prevents malicious tokens - Balance checks before every transfer - Dynamic calls wrapped in `try!` for safety - Stablecoin validation at every step - Clear error messages for wrong stablecoin

---

11. Testing Strategy

11.1 Overview

Testing Budget: $8,000 (D1.10 - Stablecoin Testing Suite)
Testing Timeline: Month 4 (1 month dedicated testing phase)
Coverage Target: >95% across all contracts and stablecoin scenarios

Testing Philosophy: - Test early, test often - Automate everything - Test in production-like environment (Stacks testnet) - Focus heavily on multi-stablecoin edge cases - Load test with realistic user volumes

---

11.2 Multi-Stablecoin Test Cases (D1.10)

D1.10 Deliverable: Comprehensive Stablecoin Testing Suite
Budget: $8,000
Timeline: Month 4 (Week 1-4)
Scope: 15+ multi-stablecoin specific test scenarios

11.2.1 Unit Tests (Clarinet) Test Suite 1: Stablecoin Whitelist Management

```
;; Test: Stablecoin whitelist functionality
(define-public (test-stablecoin-whitelist)
  (begin
    ;; Check USDA is approved
    (asserts! (is-stablecoin-approved .usda-token))

    ;; Check USDC is approved
    (asserts! (is-stablecoin-approved .usdc-token))

    ;; Check xUSD is approved
    (asserts! (is-stablecoin-approved .xusd-token))

    ;; Check random token is NOT approved
    (asserts! (not (is-stablecoin-approved .random-token)))
```

```
    (ok true)
  )
)
```

Test Suite 2: Loan Creation per Stablecoin

```
;; Test: Create loan with USDA
(define-public (test-create-loan-usda)
  (let ((loan-id (try! (create-loan-auction
                        SBTC_CONTRACT
                        u150000000
                        USDA_CONTRACT         ;; USDA
                        u50000000000
                        u53500000000
                        u8640
                        u144))))
    ;; Verify loan created with USDA
    (let ((loan (unwrap! (get-loan loan-id) (err u1))))
      (asserts! (is-eq (get borrow-asset loan) USDA_CONTRACT))
      (ok true)
    )
  )
)

;; Test: Create loan with USDC
(define-public (test-create-loan-usdc)
  (let ((loan-id (try! (create-loan-auction
                        SBTC_CONTRACT
                        u150000000
                        USDC_CONTRACT         ;; USDC
                        u50000000000
                        u53500000000
                        u8640
                        u144))))
    (let ((loan (unwrap! (get-loan loan-id) (err u1))))
      (asserts! (is-eq (get borrow-asset loan) USDC_CONTRACT))
      (ok true)
    )
  )
)

;; Test: Create loan with xUSD
(define-public (test-create-loan-xusd)
  (let ((loan-id (try! (create-loan-auction
                        SBTC_CONTRACT
                        u150000000
```

```
                        XUSD_CONTRACT         ;; xUSD
                        u50000000000
                        u53500000000
                        u8640
                        u144))))
    (let ((loan (unwrap! (get-loan loan-id) (err u1))))
      (asserts! (is-eq (get borrow-asset loan) XUSD_CONTRACT))
      (ok true)
    )
  )
)

;; Test: Reject unapproved stablecoin
(define-public (test-reject-unapproved-stablecoin)
  (let ((result (create-loan-auction
                  SBTC_CONTRACT
                  u150000000
                  .random-token           ;; Not approved
                  u50000000000
                  u53500000000
                  u8640
                  u144)))
    (asserts! (is-err result))
    (asserts! (is-eq result ERR_STABLECOIN_NOT_APPROVED))
    (ok true)
  )
)
```

Test Suite 3: Bidding with Correct Stablecoin

```
;; Test: Bidding with correct stablecoin
(define-public (test-bid-correct-stablecoin)
  (let ((loan-id (try! (create-test-auction-usda))))
    ;; Ensure bidder has USDA
    (try! (mint-usda tx-sender u60000000000))

    ;; Place bid in USDA
    (try! (place-bid loan-id u52000000000))

    ;; Verify bid accepted
    (let ((bid (unwrap! (map-get? current-bids {loan-id: loan-id}) (err u1))))
      (asserts! (is-eq (get amount bid) u52000000000))
      (ok true)
    )
  )
)
```

```
;; Test: Reject bid with insufficient balance
(define-public (test-reject-insufficient-balance)
  (let ((loan-id (try! (create-test-auction-usda))))
    ;; Bidder has 0 USDA

    ;; Try to bid (should fail)
    (let ((result (place-bid loan-id u52000000000)))
      (asserts! (is-err result))
      (asserts! (is-eq result ERR_INSUFFICIENT_STABLECOIN))
      (ok true)
    )
  )
)
```

Test Suite 4: Loan Finalization with Dynamic Transfers

```
;; Test: Finalize with USDA transfer
(define-public (test-finalize-usda-transfer)
  (let ((loan-id (try! (create-test-auction-usda))))
    ;; Place bid
    (try! (mint-usda 'SPBIDDER u60000000000))
    (try! (as 'SPBIDDER (place-bid loan-id u51000000000)))

    ;; Advance to auction end
    (advance-blocks u144)

    ;; Finalize auction
    (try! (finalize-loan-auction loan-id))

    ;; Verify USDA transferred from lender to borrower
    (let ((borrower-balance (contract-call? .usda-token get-balance borrower)))
      (asserts! (is-eq borrower-balance u50000000000))
      (ok true)
    )
  )
)

;; Test: Finalize with USDC transfer
(define-public (test-finalize-usdc-transfer)
  (let ((loan-id (try! (create-test-auction-usdc))))
    (try! (mint-usdc 'SPBIDDER u60000000000))
    (try! (as 'SPBIDDER (place-bid loan-id u51000000000)))
    (advance-blocks u144)
    (try! (finalize-loan-auction loan-id))

    ;; Verify USDC transferred
    (let ((borrower-balance (contract-call? .usdc-token get-balance borrower)))
```

```
      (asserts! (is-eq borrower-balance u50000000000))
      (ok true)
    )
  )
)
```

Test Suite 5: Repayment with Correct Stablecoin

```
;; Test: Repay USDA loan with USDA
(define-public (test-repay-usda-loan)
  (let ((loan-id (try! (create-and-finalize-usda-loan))))
    ;; Borrower has USDA for repayment
    (try! (mint-usda borrower u52000000000))

    ;; Repay loan
    (try! (as borrower (repay-loan loan-id)))

    ;; Verify loan status = "completed"
    (let ((loan (unwrap! (get-loan loan-id) (err u1))))
      (asserts! (is-eq (get status loan) "completed"))
      (ok true)
    )
  )
)

;; Test: Cannot repay USDA loan with USDC
(define-public (test-cannot-repay-wrong-stablecoin)
  (let ((loan-id (try! (create-and-finalize-usda-loan))))
    ;; Borrower has USDC but not USDA
    (try! (mint-usdc borrower u52000000000))

    ;; Try to repay (should fail - insufficient USDA)
    (let ((result (as borrower (repay-loan loan-id))))
      (asserts! (is-err result))
      (ok true)
    )
  )
)
```

Test Suite 6: Cross-Stablecoin Scenarios

```
;; Test: Multiple concurrent loans with different stablecoins
(define-public (test-concurrent-multi-stablecoin-loans)
  (let ((loan-usda (try! (create-loan-auction SBTC u100000000 USDA u30000000000 u3100000000
        (loan-usdc (try! (create-loan-auction SBTC u100000000 USDC u30000000000 u3100000000
        (loan-xusd (try! (create-loan-auction SBTC u100000000 XUSD u30000000000 u3100000000
    ;; All 3 loans should exist
    (asserts! (is-some (get-loan loan-usda)))
```

```
    (asserts! (is-some (get-loan loan-usdc)))
    (asserts! (is-some (get-loan loan-xusd)))
    (ok true)
  )
)

;; Test: Balance checks across multiple stablecoins
(define-public (test-balance-checks-multi-stablecoin)
  (begin
    ;; User has 50K USDA, 30K USDC, 10K xUSD
    (try! (mint-usda user u50000000000))
    (try! (mint-usdc user u30000000000))
    (try! (mint-xusd user u10000000000))

    ;; Can bid on USDA loan
    (let ((loan-usda (try! (create-test-auction-usda))))
      (asserts! (can-place-bid? user loan-usda u48000000000))
      (ok true)
    )

    ;; Can bid on USDC loan
    (let ((loan-usdc (try! (create-test-auction-usdc))))
      (asserts! (can-place-bid? user loan-usdc u28000000000))
      (ok true)
    )

    ;; Cannot bid on xUSD loan (insufficient balance)
    (let ((loan-xusd (try! (create-test-auction-xusd))))
      (asserts! (not (can-place-bid? user loan-xusd u15000000000)))
      (ok true)
    )
  )
)
```

11.2.2 D1.10 Test Coverage Goals Coverage Targets (D1.10 deliverable):

Component	Target Coverage	Test Cases
Stablecoin whitelist	100%	5 tests
Loan creation	100%	10 tests (3 per coin + edge cases)
Bidding logic	>95%	15 tests (5 per coin)
Auction finalization	100%	12 tests (4 per coin)
Repayment	100%	12 tests (4 per coin)
Cross-stablecoin	>90%	8 tests
TOTAL	**>95%**	**62+ tests**

Component	Target Coverage	Test Cases

Total Test Execution Time: ~2 hours (automated)
Continuous Integration: Run on every commit

11.3 Integration Testing (Month 4, Weeks 2-3)

Integration Test Scope: - Full user flows (end-to-end) - Frontend Backend Blockchain integration - Real testnet deployment - Multi-stablecoin wallet integration

Test Scenarios:

1. **Complete Borrower Flow (USDA)**
   - Connect wallet
   - Select USDA as stablecoin
   - Create loan auction
   - Monitor auction progress
   - Wait for finalization
   - Receive USDA
   - Repay loan
   - Reclaim sBTC collateral
2. **Complete Lender Flow (USDC)**
   - Connect wallet
   - Browse USDC auctions
   - Check USDC balance
   - Place bid
   - Win auction
   - USDC transferred to borrower
   - Wait for maturity
   - Receive repayment in USDC
3. **Mixed Stablecoin Portfolio**
   - User has USDA, USDC, xUSD
   - Create 3 loans (one in each)
   - Place bids on 3 auctions (one in each)
   - Verify correct stablecoin used for each

Integration Testing Tools: - **Playwright** or **Cypress** for E2E frontend tests - **Stacks.js** for blockchain interaction tests - **Real testnet** deployment (not mocked)

11.4 Load Testing (Month 4, Week 4)

Load Test Objectives: - Verify 1000+ concurrent users supported - Test 100+ active auctions simultaneously - Measure API response times under load - Verify database handles high query volume

Load Test Scenarios:

```
// Scenario 1: 1000 concurrent borrowers create loans
import { check } from 'k6';
import http from 'k6/http';

export let options = {
  vus: 1000, // 1000 virtual users
  duration: '10m',
};

export default function () {
  let payload = JSON.stringify({
    collateralAmount: '150000000',
    borrowAsset: 'usda', // Rotate through usda, usdc, xusd
    borrowAmount: '50000000000',
    maxRepayment: '53500000000',
    durationBlocks: 8640,
    auctionDuration: 144,
  });

  let res = http.post('https://api-testnet.bitcoinlending.xyz/api/v1/loans', payload);

  check(res, {
    'loan created': (r) => r.status === 201,
    'response < 2s': (r) => r.timings.duration < 2000,
  });
}
```

Performance Targets (from Section 7.1 of PRD): - API response time: <500ms (p95) - Page load time: <2 seconds - WebSocket latency: <200ms - Transaction confirmation: <15 seconds

---

11.5 Mobile & Cross-Browser Testing (Month 4, Week 3)

Browser Compatibility: - Chrome (latest 2 versions) - Firefox (latest 2 versions) - Safari (latest 2 versions) - Edge (latest 2 versions)

Mobile Testing: - iOS Safari (iPhone 12+, iPad) - Android Chrome (Samsung, Pixel) - Responsive design (320px - 2560px width)

Wallet Integration Testing: - Leather Wallet (desktop + mobile) - Xverse Wallet (desktop + mobile) - Asigna Wallet (desktop)

Test Cases: - Wallet connection on each browser - Create loan flow on mobile - Place bid flow on mobile - Stablecoin selection UI on small screens - Transaction signing on each wallet

---

11.6 Testing Timeline (Month 4)

```
Week 1: D1.10 Unit Testing

Mon-Wed: Write all 62+ test cases
Thu-Fri: Run tests, fix failures, achieve >95% coverage

Week 2: Integration Testing

Mon-Tue: E2E borrower flows (all 3 stablecoins)
Wed-Thu: E2E lender flows (all 3 stablecoins)
Fri: Mixed portfolio scenarios

Week 3: Cross-Browser & Mobile Testing

Mon-Tue: Desktop browsers (Chrome, Firefox, Safari, Edge)
Wed-Thu: Mobile testing (iOS, Android)
Fri: Wallet integration testing

Week 4: Load Testing & Final QA

Mon-Tue: Load testing (1000 users, 100 auctions)
Wed: Performance optimization based on load test results
Thu: Final regression testing
Fri: Bug bash session (entire team tests)
```

Total Testing Budget: $8,000 (D1.10)
Total Testing Time: 1 month (Month 4)

11.7 Continuous Integration (CI)

Automated Test Execution: - Run all unit tests on every commit (GitHub Actions) - Run integration tests on every PR - Nightly load tests on testnet - Weekly full regression suite

CI Pipeline:

```yaml
name: Test Suite
on: [push, pull_request]

jobs:
  unit-tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Run Clarinet tests
        run: clarinet test
      - name: Check coverage
        run: clarinet test --coverage
      - name: Fail if coverage < 95%
        run: |
          coverage=$(clarinet test --coverage | grep -oP '\d+\.\d+(?=%)')
          if (( $(echo "$coverage < 95" | bc -l) )); then
            echo "Coverage $coverage% is below 95%"
            exit 1
          fi

  integration-tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Run E2E tests
        run: npm run test:e2e
```

11.8 Test Reporting

Test Report Deliverables (D1.10): - Unit test coverage report (HTML) - Integration test results summary - Load test performance metrics - Cross-browser compatibility matrix - Mobile testing results - Final QA sign-off document

Example Coverage Report:

```
Phase 1 Smart Contract Test Coverage Report


loan-protocol.clar:        97.2% (248/255 lines)
borrower-nft.clar:         98.1% (104/106 lines)
lender-nft.clar:          98.1% (104/106 lines)
marketplace.clar:          96.3% (130/135 lines)

Multi-Stablecoin Tests:    100% (62/62 tests passed)
Integration Tests:         100% (18/18 scenarios passed)
```

```
Load Tests:                PASSED (1000 users, <2s response)
Cross-Browser:             PASSED (4/4 browsers)
Mobile:                    PASSED (iOS + Android)

OVERALL COVERAGE:          97.4%
STATUS:                      READY FOR MAINNET
```

---

12. Deployment Plan

12.1 Overview

This section provides a comprehensive deployment strategy for launching the Bitcoin Lending Protocol on Stacks mainnet. The deployment follows a **7-month timeline** (Months 0-7) with a total budget of **$308,000**, including multiple validation checkpoints before production launch.

12.2 Pre-Deployment Checklist

Before any deployment can proceed, ALL of these must be complete:

12.2.1 Development Completeness

- ☐ All smart contracts finalized (loan-protocol, borrower-nft, lender-nft, marketplace)
- ☐ Multi-stablecoin whitelist implemented (USDA, USDC, xUSD)
- ☐ Frontend fully developed with all user flows
- ☐ API backend implemented and tested
- ☐ NFT metadata generation working
- ☐ WebSocket real-time updates functional

12.2.2 Security & Audit

- ☐ Professional security audit completed (D1.1 - $88,000)
  - Zero critical vulnerabilities
  - All high-severity issues resolved
  - Medium/low issues documented with mitigation plans
- ☐ Audit report publicly published
- ☐ Bug bounty program prepared ($50K+ rewards)
- ☐ Emergency pause mechanism tested
- ☐ Admin key management procedures documented

12.2.3 Testing & Quality Assurance

- ☐ Unit test coverage >95% (D1.10 - $8,000)

- ☐ All integration tests passing
- ☐ Stablecoin-specific test scenarios complete (15+ cases)
- ☐ Load testing completed (1000+ concurrent users)
- ☐ Frontend E2E tests passing (Playwright/Cypress)
- ☐ Cross-browser testing complete (Chrome, Firefox, Safari, Edge)
- ☐ Mobile responsiveness verified (iOS Safari, Android Chrome)
- ☐ Testnet deployment validated (2+ months of testing)

12.2.4 Documentation

- ☐ User documentation complete (how to borrow, how to lend)
- ☐ FAQ written and reviewed
- ☐ Video tutorials created
- ☐ Developer documentation published
- ☐ API documentation complete (Swagger/OpenAPI)
- ☐ Stablecoin selection guide written
- ☐ Risk warnings clearly documented

12.2.5 Operational Readiness

- ☐ Monitoring dashboards configured (Grafana)
- ☐ Alerting rules set up (PagerDuty)
- ☐ On-call rotation established
- ☐ Incident response runbooks prepared
- ☐ Backup procedures tested
- ☐ Disaster recovery plan documented
- ☐ Team trained on deployment procedures

12.2.6 Business Readiness

- ☐ Legal terms of service finalized
- ☐ Privacy policy published
- ☐ Risk disclaimers approved
- ☐ Marketing materials prepared
- ☐ Community Discord/Telegram active
- ☐ Miner outreach initiated (D1.6 - $17,000)
- ☐ Lender acquisition plan ready
- ☐ Launch announcement drafted

12.3 Deployment Timeline (7 Months)

Month 1: Foundation & Testnet **Week 1-2: Development Completion** - Finalize all smart contract code - Complete frontend development - Complete API backend

Week 3-4: Initial Testnet Deployment - Deploy contracts to Stacks testnet - Initialize stablecoin whitelist (testnet tokens) - Deploy frontend to staging environment - Begin internal testing

Deliverables: - D1.2: Stacks Mainnet Deploy preparation ($34,000) - D1.3: sBTC Collateral Integration ($23,000) - D1.4: Competitive Bidding Auction ($23,000) - D1.5: NFT Positions Trading ($23,000)

Budget: ~$103K

Month 2: Multi-Stablecoin Development Week 1-2: Stablecoin Integration - Implement USDA integration - Implement USDC integration - Implement xUSD integration - Test whitelist management

Week 3-4: Multi-Stablecoin Testing - Test all stablecoin flows - Cross-stablecoin scenarios - Balance validation testing - UI stablecoin selector testing

Deliverables: - D1.9: Multi-Stablecoin Integration ($44,000) **NEW**

Budget: ~$44K

Month 3: Security Audit & Enhanced Development Week 1-2: Security Audit - Professional audit begins (D1.1) - Multi-stablecoin logic review - Dynamic contract calls audit - Balance validation audit

Week 3-4: Audit Findings & Fixes - Address audit findings - Re-test fixed code - Prepare for follow-up audit

Concurrent Work: - Frontend UI enhancements (D1.5a continues) - NFT Marketplace UI development (D1.5b) - Miner outreach begins (D1.6)

Deliverables: - D1.1: Security Audit Start ($88,000) - D1.5a: Lending/Borrowing UI ongoing ($24,000) - D1.5b: NFT Marketplace UI ($15,000) - D1.6: Miner Outreach begins ($17,000)

Budget: ~$144K (partial)

Month 4: Comprehensive Testing Week 1-2: Stablecoin Testing Suite - USDA test suite (D1.10) - USDC test suite - xUSD test suite - Integration tests

Week 3-4: Load & Performance Testing - Load testing (1000+ users) - Performance optimization - Mobile testing - Cross-browser testing

Deliverables: - D1.10: Stablecoin Testing Suite ($8,000) **NEW**

Budget: ~$8K

Month 5: Launch Preparation Week 1: Final Testing - Complete testnet validation - Final security review - Bug bash session - UAT with beta users

Week 2: Mainnet Preparation - Mainnet deployment dry run - DNS and infrastructure setup - Monitoring configuration - Alert rule verification

Week 3: Mainnet Deployment - **Deploy to Mainnet** (see Section 12.4 below) - Verify all contracts - Initialize stablecoin whitelist - Deploy frontend to production

Week 4: Post-Deployment Validation - Smoke tests on mainnet - Monitor for 24 hours - Fix any critical issues - Prepare for soft launch

Deliverables: - D1.2: Stacks Mainnet Deploy complete ($34,000)

Budget: ~$34K

Month 6: Soft Launch & Early Adoption Week 1: Soft Launch - **MAINNET LAUNCH** - Limited to whitelisted users initially - Max loan size: $50K (first week) - Close monitoring

Week 2-3: Gradual Rollout - Increase loan caps gradually - Onboard more users - Monitor all 3 stablecoins - Collect user feedback

Week 4: Full Public Launch - Remove whitelist restrictions - Announce publicly - Marketing campaign - Community growth

Deliverables: - D1.6: First Loans Created (ongoing)

Budget: Marketing & support costs

Month 7: Growth & Milestone Achievement Week 1-3: Volume Growth - Drive usage toward $1M volume (D1.7) - Optimize based on stablecoin preferences - Fix any non-critical bugs - Performance optimizations

Week 4: Milestone & Retrospective - **$1M VOLUME TARGET** - Stablecoin usage analysis - Default rate monitoring - Phase 1 retrospective - Begin Phase 2 planning

Deliverables: - D1.7: $1M Volume Milestone ($9,000)

Budget: ~$9K

12.4 Mainnet Deployment Procedure (Month 5, Week 3)

Duration: 2-3 days
Team: 2-3 people (Lead Developer, DevOps, PM)
Cost: ~$2K (included in D1.2 budget of $34K)

Day 1: Contract Deployment Morning (9 AM - 12 PM):

Step 1: Final Pre-Deployment Checks (1 hour)

```
# Run all tests one final time
$ npm run test:all
# All tests must pass

# Check mainnet STX balance for deployment
$ clarinet accounts
# Deployer account must have sufficient STX for deployment fees

# Verify contract hashes match audited versions
$ sha256sum contracts/*.clar
# Compare against audit report hashes
```

Step 2: Deploy Main Loan Protocol (1 hour)

```
# Deploy loan-protocol.clar
$ clarinet deploy --mainnet contracts/loan-protocol.clar

# Transaction ID: 0xABC123...
# Wait for confirmation (10-15 minutes)

# Verify deployment
$ clarinet call mainnet loan-protocol get-protocol-info

# Expected: Protocol initialized with owner = deployer
```

Step 3: Deploy NFT Contracts (1 hour)

```
# Deploy borrower-nft.clar
$ clarinet deploy --mainnet contracts/borrower-nft.clar

# Deploy lender-nft.clar
$ clarinet deploy --mainnet contracts/lender-nft.clar

# Verify NFT contracts can mint (test call)
$ clarinet call mainnet borrower-nft get-last-token-id
# Expected: u0 (no tokens minted yet)
```

Afternoon (1 PM - 5 PM):

Step 4: Initialize Stablecoin Whitelist (2 hours)

```
# Add USDA to whitelist
$ clarinet call mainnet loan-protocol add-whitelisted-stablecoin \
    'SP2ABC...XYZ.usda-token

# Add USDC to whitelist
$ clarinet call mainnet loan-protocol add-whitelisted-stablecoin \
    'SP2DEF...ABC.usdc-token

# Add xUSD to whitelist
$ clarinet call mainnet loan-protocol add-whitelisted-stablecoin \
    'SP2GHI...DEF.xusd-token

# Verify whitelist
$ clarinet call mainnet loan-protocol is-whitelisted-stablecoin \
    'SP2ABC...XYZ.usda-token
# Expected: true
```

Step 5: Deploy Marketplace Contract (1 hour)

```
# Deploy marketplace.clar
$ clarinet deploy --mainnet contracts/marketplace.clar

# Verify marketplace initialized
$ clarinet call mainnet marketplace get-last-listing-id
# Expected: u0 (no listings yet)
```

Step 6: Configure Admin Controls (1 hour)

```
# Set protocol parameters
$ clarinet call mainnet loan-protocol set-protocol-fee u0
# Phase 1: Zero fees

# Set emergency pause admins
$ clarinet call mainnet loan-protocol add-admin 'SP2ADMIN1...
$ clarinet call mainnet loan-protocol add-admin 'SP2ADMIN2...

# Verify admin list
$ clarinet call mainnet loan-protocol is-admin 'SP2ADMIN1...
# Expected: true
```

Day 2: Infrastructure Deployment Morning (9 AM - 12 PM):

Step 7: Deploy Backend API (2 hours)

```
# Deploy to production (AWS/GCP/Vercel)
$ cd backend
$ npm run deploy:prod
```

```
# Environment variables:
STACKS_NETWORK=mainnet
LOAN_PROTOCOL_ADDRESS=SP2ABC...XYZ.loan-protocol
BORROWER_NFT_ADDRESS=SP2ABC...XYZ.borrower-nft
LENDER_NFT_ADDRESS=SP2ABC...XYZ.lender-nft
MARKETPLACE_ADDRESS=SP2ABC...XYZ.marketplace
DATABASE_URL=postgresql://...

# Verify API health
$ curl https://api.bitcoinlending.xyz/health
# Expected: {"status": "ok"}
```

Step 8: Deploy Frontend (2 hours)

```
# Deploy to production (Vercel/Netlify)
$ cd frontend
$ npm run build
$ npm run deploy:prod

# Environment variables:
NEXT_PUBLIC_NETWORK=mainnet
NEXT_PUBLIC_API_URL=https://api.bitcoinlending.xyz
NEXT_PUBLIC_LOAN_PROTOCOL=SP2ABC...XYZ.loan-protocol

# Verify frontend loads
$ curl https://bitcoinlending.xyz
# Expected: 200 OK
```

Afternoon (1 PM - 5 PM):

Step 9: DNS and Monitoring Setup (2 hours)

```
# Configure DNS (if not done)
bitcoinlending.xyz → Vercel/Netlify
api.bitcoinlending.xyz → AWS ALB/GCP Load Balancer

# Set up Grafana dashboard
- Import loan metrics dashboard
- Configure Stacks API data source
- Set up alert rules

# Set up PagerDuty alerts
- Critical: Any contract error
- High: Deployment failure
- Medium: High latency
```

Step 10: Smoke Tests on Mainnet (2 hours)

```
# Create test loan (small amount, short duration)
- Borrow: $100 USDA
- Collateral: 0.01 sBTC
- Duration: 1 day

# Place test bid
- Bid: $101 USDA

# Let auction complete
- Wait 24 hours (or shorter test duration)

# Verify loan activates correctly

# Test repayment
- Repay $101 USDA
- Verify collateral released
- Verify NFTs burned
```

Day 3: Validation & Soft Launch Morning (9 AM - 12 PM):

Step 11: Post-Deployment Validation (2 hours)

```
Checklist:
[ ] All contracts deployed successfully
[ ] Contract addresses documented and verified
[ ] Stablecoin whitelist correct (USDA, USDC, xUSD)
[ ] NFT contracts working
[ ] Marketplace functional
[ ] Admin controls set correctly
[ ] Frontend loads correctly
[ ] API responding correctly
[ ] Monitoring dashboards showing data
[ ] Alerts configured and tested
[ ] Test loan completed successfully
[ ] All integration tests passing on mainnet
```

Step 12: Soft Launch Preparation (2 hours) - Announce to Discord/Telegram (whitelist users only) - Send emails to beta testers - Limit initial loans to $50K max - Enable close monitoring (24/7)

Afternoon (1 PM - 5 PM):

Step 13: Monitor First 24 Hours (ongoing) - Watch for any errors in logs - Monitor all transactions - Check user feedback - Fix any issues immediately - Document any unexpected behavior

---

12.5 Rollback Plan

If critical bug found during deployment:

Severity 1: Funds at Risk **Action**: IMMEDIATE PAUSE

```
;; Emergency pause (admin only)
(contract-call? .loan-protocol emergency-pause)

;; This stops:
- New loan creation
- New bids
- Loan finalization

;; Does NOT stop:
- Existing loan repayments (users can still repay)
- Collateral claims (lenders can still claim)
```

Next Steps: 1. Investigate issue (2-4 hours) 2. Identify root cause 3. Develop fix 4. Test fix on testnet 5. Audit fix (if contract change needed) 6. Deploy fix or new version 7. Unpause protocol 8. Communicate to users

Timeline: 1-3 days depending on severity

Severity 2: Functionality Broken (No Fund Risk) **Action**: Continue operating, fix in next patch

Example: Marketplace listing fails, but loans still work

Next Steps: 1. Document workaround 2. Develop fix 3. Test fix 4. Deploy patch (within 1 week)

Severity 3: Minor Issue **Action**: Document and queue for next release

Example: UI display bug, incorrect APR calculation display

Next Steps: 1. Add to backlog 2. Fix in next sprint 3. Deploy in regular release cycle

---

12.6 Post-Deployment Monitoring (First 7 Days)

Day 1-2: Critical Monitoring (24/7) - All hands on deck - Monitor every transaction - Respond to issues within minutes - No new features, only bug fixes

Day 3-5: High Monitoring - 12-hour coverage - Check logs every 2 hours - Respond to issues within 1 hour

Day 6-7: Normal Monitoring - 8-hour coverage (business hours) - Check logs every 4 hours - Respond to issues within 4 hours

Metrics to Track: - Transaction success rate (target: >99%) - Contract call failures (target: <1%) - API uptime (target: >99.9%) - User-reported issues (target: <5 per day) - Default rate (track carefully)

12.7 Deployment Budget Breakdown

Total Deployment Cost: ~$36K (included in $308K total budget)

Item	Cost	Included In
Mainnet contract deployment fees	~$2,000	D1.2 ($34K)
Infrastructure (AWS/GCP)	$500/month	D1.2 ($34K)
Monitoring tools (Grafana Cloud, PagerDuty)	$300/month	D1.2 ($34K)
Domain & SSL	$50/year	D1.2 ($34K)
NFT image generation (IPFS/Arweave)	$200	D1.2 ($34K)
Team time (3 people × 3 days)	Included	D1.2 ($34K)
Marketing for launch	$2,000	D1.6 ($17K)

12.8 Success Criteria

Deployment is successful if:

- ☐ All contracts deployed to mainnet
- ☐ All 3 stablecoins working (USDA, USDC, xUSD)
- ☐ First 10 loans created without issues
- ☐ First 5 loans repaid successfully
- ☐ NFT minting/burning working
- ☐ Marketplace functional
- ☐ No critical bugs in first 7 days
- ☐ Zero fund losses

☐ >99% uptime in first month
☐ Positive user feedback

Ready for Phase 2 if: - [ ] $1M+ volume achieved - [ ] All 3 stablecoins used (>5% each) - [ ] Default rate <5% - [ ] No security incidents - [ ] Stable operation for 2+ months

---

13. Monitoring & Maintenance

13.1 Multi-Stablecoin Metrics

Key Metrics to Track:

Metric	Target	Alert Threshold
USDA loan volume	60% of total	<40% (preference shift)
USDC loan volume	30% of total	<10% (low adoption)
xUSD loan volume	10% of total	>30% (unexpected growth)
Stablecoin diversity	2+ active	<2 (over-concentration)
USDA liquidity	$1M+ available	<$250K
USDC liquidity	$500K+ available	<$100K
Balance check failures	<1%	>5%
Wrong stablecoin errors	<0.5%	>2%

Dashboard Views:

```
Stablecoin Distribution

USDA:  [          ] 65% ($650K)
USDC:  [          ] 28% ($280K)
xUSD:  [          ] 7% ($70K)

Active Loans by Stablecoin

USDA: 12 loans ($450K borrowed, $485K repaying)
USDC: 5 loans ($200K borrowed, $218K repaying)
xUSD: 2 loans ($50K borrowed, $53K repaying)

Average APR by Stablecoin

USDA: 6.8%
USDC: 8.2%
xUSD: 7.5%
```

Alerts:

```
P1 Alert: USDA liquidity low

USDA available: $180K
Target: $1M
Action: Encourage USDA lenders, marketing push

P2 Alert: xUSD usage spike

xUSD loans: 8 (30% of total, target 10%)
Action: Investigate why, ensure sufficient xUSD liquidity
```

Appendix A: Calculation Formulas (Multi-Stablecoin)

A.1 Implied APR Calculation

```
Given:
  P = borrow_amount (principal in stablecoin)
  R = repayment_amount (from winning bid in stablecoin)
  D = loan_duration_blocks
  Y = 52,560 (blocks per year, ~365.25 days)

Calculate:
  I = R - P (interest in stablecoin)
  r = I / P (rate for this duration)
  APR = (r * Y / D) * 100

Example (USDA):
  P = 50,000 USDA
  R = 50,500 USDA (winning bid)
  D = 8,640 blocks (60 days)

  I = 50,500 - 50,000 = 500 USDA
  r = 500 / 50,000 = 0.01
  APR = (0.01 * 52,560 / 8,640) * 100 = 6.08%

Example (USDC):
  P = 100,000 USDC
  R = 107,500 USDC (winning bid)
  D = 8,640 blocks (60 days)

  I = 107,500 - 100,000 = 7,500 USDC
  r = 7,500 / 100,000 = 0.075
  APR = (0.075 * 52,560 / 8,640) * 100 = 45.6%

Note: Formula is same regardless of stablecoin!
```

A.2 Stablecoin Amount Formatting

```javascript
function formatStablecoinAmount(amount: bigint, symbol: string): string {
  const stablecoin = STABLECOINS[symbol];
  const decimals = stablecoin.decimals; // Usually 6

  const divisor = BigInt(10 ** decimals);
  const dollars = Number(amount / divisor);
  const cents = Number((amount % divisor) / BigInt(10 ** (decimals - 2)));

  return `$${dollars.toLocaleString()}.${cents.toString().padStart(2, '0')} ${symbol}`;
}

// Examples:
formatStablecoinAmount(50000000000n, 'USDA')  // "$50,000.00 USDA"
formatStablecoinAmount(50500000000n, 'USDC')  // "$50,500.00 USDC"
formatStablecoinAmount(1250000n, 'xUSD')      // "$1.25 xUSD"
```

Appendix B: Stablecoin Contract Interfaces

B.1 SIP-010 Fungible Token Standard

All stablecoins must implement SIP-010:

```
;; Get balance
(define-read-only (get-balance (account principal))
  (response uint uint))

;; Transfer
(define-public (transfer
    (amount uint)
    (sender principal)
    (recipient principal)
    (memo (optional (buff 34))))
  (response bool uint))

;; Get name
(define-read-only (get-name)
  (response (string-ascii 32) uint))

;; Get symbol
(define-read-only (get-symbol)
  (response (string-ascii 10) uint))

;; Get decimals
(define-read-only (get-decimals)
```

```
  (response uint uint))

;; Get total supply
(define-read-only (get-total-supply)
  (response uint uint))
```

B.2 Calling Stablecoin Contracts

```
;; Get bidder's USDA balance
(let ((balance (unwrap!
                 (contract-call? .usda-token get-balance tx-sender)
                 ERR_BALANCE_CHECK_FAILED)))
  (asserts! (>= balance bid-amount) ERR_INSUFFICIENT_STABLECOIN)
)

;; Transfer USDC from lender to borrower
(try! (contract-call? .usdc-token transfer
  loan-amount
  lender-address
  borrower-address
  none))

;; Dynamic call based on loan's stablecoin
(let ((stablecoin-contract (get borrow-asset loan)))
  (try! (contract-call? stablecoin-contract transfer
    amount
    from
    to
    none))
)
```

End of Technical Specification Document - Phase 1

Document Version: 1.2 (Multi-Stablecoin Support)
Last Updated: January 12, 2026

This technical specification provides complete implementation details for Phase 1 with comprehensive multi-stablecoin support (USDA, USDC, xUSD) integrated throughout the protocol.