BigMarketDao / **bigmarket-dao** Public

Code · Issues · Pull requests · Actions · Projects · Security · Insights

main

bigmarket-dao / contracts / **bigmarket-dao.clar**

Go to file

mijoco-btc attempt to resolve clarinet toml issues around pyth. code is basicall...

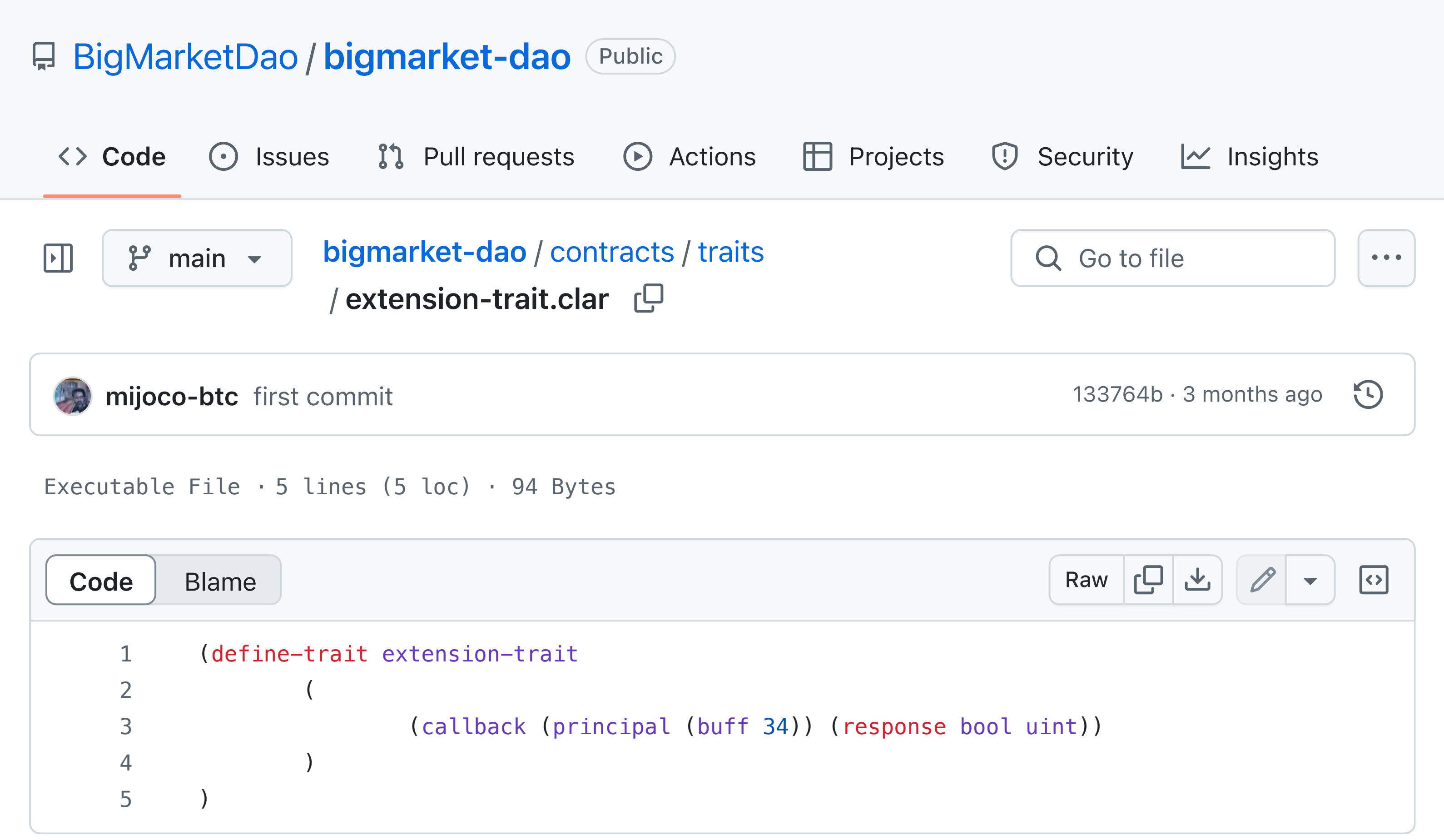
767bb51 · 3 weeks ago

Executable File · 86 lines (69 loc) · 2.72 KB

Code | Blame | Raw

```
;; BigMarket DAO
;; Synopsis:
;; bigmarket-dao is the core of the dao framework.
;; Description:
;; Valid extensions must be registered here. The DAO is bootstrapped
;; by calling construct with a bootstrap proposal.

(use-trait extension-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.extension-trait.extensic
(use-trait proposal-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.proposal-trait.proposal-t

(define-constant err-unauthorised (err u1000))
(define-constant err-already-executed (err u1001))
(define-constant err-invalid-extension (err u1002))

(define-data-var executive principal tx-sender)
(define-map executed-proposals principal uint)
(define-map extensions principal bool)

;; --- Authorisation check

(define-private (is-self-or-extension)
	(ok (asserts! (or (is-eq tx-sender (as-contract tx-sender)) (is-extension contract-ca
)

;; --- Extensions

(define-read-only (is-extension (extension principal))
	(default-to false (map-get? extensions extension))
)

(define-public (set-extension (extension principal) (enabled bool))
	(begin
		(try! (is-self-or-extension))
		(print {event: "extension", extension: extension, enabled: enabled})
		(ok (map-set extensions extension enabled))
	)
)
```

```

(define-private (set-extensions-iter (item {extension: principal, enabled: bool}))
        (begin
                (print {event: "extension", extension: (get extension item), enabled: (get e
                (map-set extensions (get extension item) (get enabled item))
        )
)

(define-public (set-extensions (extension-list (list 200 {extension: principal, enabled: bool
        (begin
                (try! (is-self-or-extension))
                (ok (map set-extensions-iter extension-list))
        )
)

;; --- Proposals

(define-read-only (executed-at (proposal <proposal-trait>))
        (map-get? executed-proposals (contract-of proposal))
)

(define-public (execute (proposal <proposal-trait>) (sender principal))
        (begin
                (try! (is-self-or-extension))
                (asserts! (map-insert executed-proposals (contract-of proposal) stacks-block-
                (print {event: "execute", proposal: proposal})
                (as-contract (contract-call? proposal execute sender))
        )
)

;; --- Bootstrap

(define-public (construct (proposal <proposal-trait>))
        (let ((sender tx-sender))
                (asserts! (is-eq sender (var-get executive)) err-unauthorised)
                (var-set executive (as-contract tx-sender))
                (as-contract (execute proposal sender))
        )
)

;; --- Extension requests

(define-public (request-extension-callback (extension <extension-trait>) (memo (buff 34)))
        (let ((sender tx-sender))
                (asserts! (is-extension contract-caller) err-invalid-extension)
                (asserts! (is-eq contract-caller (contract-of extension)) err-invalid-extensi
                (as-contract (contract-call? extension callback sender memo))
        )
)
```


BigMarketDao / bigmarket-dao Public

Code Issues Pull requests Actions Projects Security Insights

main bigmarket-dao / contracts / traits / extension-trait.clar

Go to file

mijoco-btc first commit 133764b · 3 months ago

Executable File · 5 lines (5 loc) · 94 Bytes

Code Blame Raw

```
(define-trait extension-trait
	(
		(callback (principal (buff 34)) (response bool uint))
	)
)
```

main bigmarket-dao / contracts / traits / governance-token-trait.clar

Go to file

mijoco-btc Renamed contracts, updated heading comments, tweaked the custom major...

79c28af · 2 months ago

Executable File · 12 lines (12 loc) · 490 Bytes

Code Blame Raw

```
(define-trait governance-token-trait
        (
                (bmg-get-balance (principal) (response uint uint))
                (bmg-has-percentage-balance (principal uint) (response bool uint))
                (bmg-transfer (uint principal principal) (response bool uint))
                (bmg-lock (uint principal) (response bool uint))
                (bmg-unlock (uint principal) (response bool uint))
                (bmg-get-locked (principal) (response uint uint))
                (bmg-mint (uint principal) (response bool uint))
                (bmg-burn (uint principal) (response bool uint))
        )
)
```

BigMarketDao / **bigmarket-dao** Public

Code Issues Pull requests Actions Projects Security Insights

main bigmarket-dao / contracts / traits / **prediction-market-trait.clar**

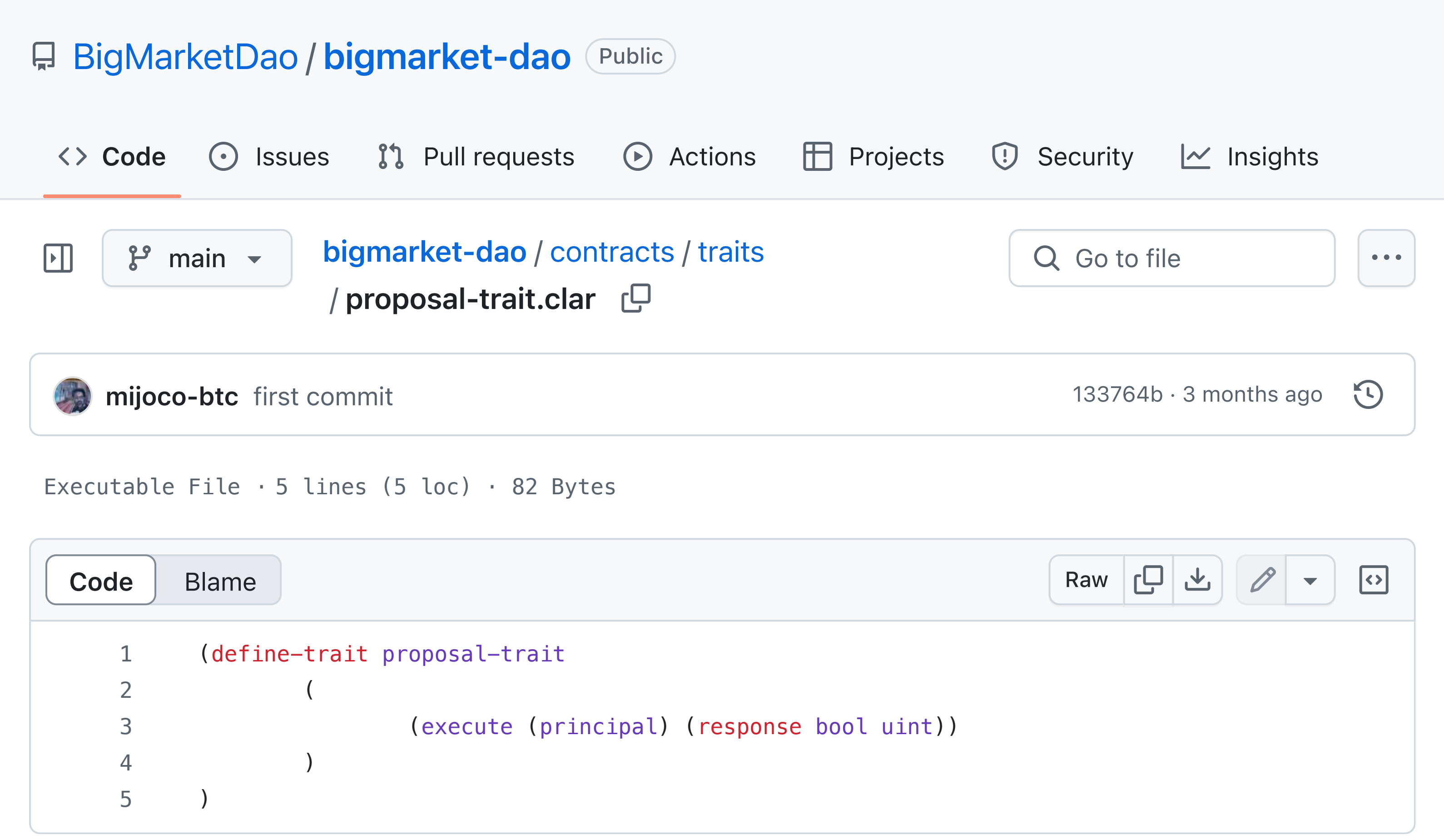
Go to file

mijoco-btc Inclusion of reputation soul bound sft. 32188d4 · last week

6 lines (6 loc) · 169 Bytes

Code Blame Raw

```clarity
(define-trait prediction-market-trait
  (
    (dispute-resolution (uint principal) (response bool uint))
    (resolve-market-vote (uint uint) (response bool uint))
  )
)
```



bigmarket-dao / contracts / traits / proposal-trait.clar

mijoco-btc first commit 133764b · 3 months ago

Executable File · 5 lines (5 loc) · 82 Bytes

```
(define-trait proposal-trait
	(
		(execute (principal) (response bool uint))
	)
)
```

BigMarketDao / **bigmarket-dao** Public

Code · Issues · Pull requests · Actions · Projects · Security · Insights

main

bigmarket-dao / contracts / traits / **sip013-semi-fungible-token-trait.clar**

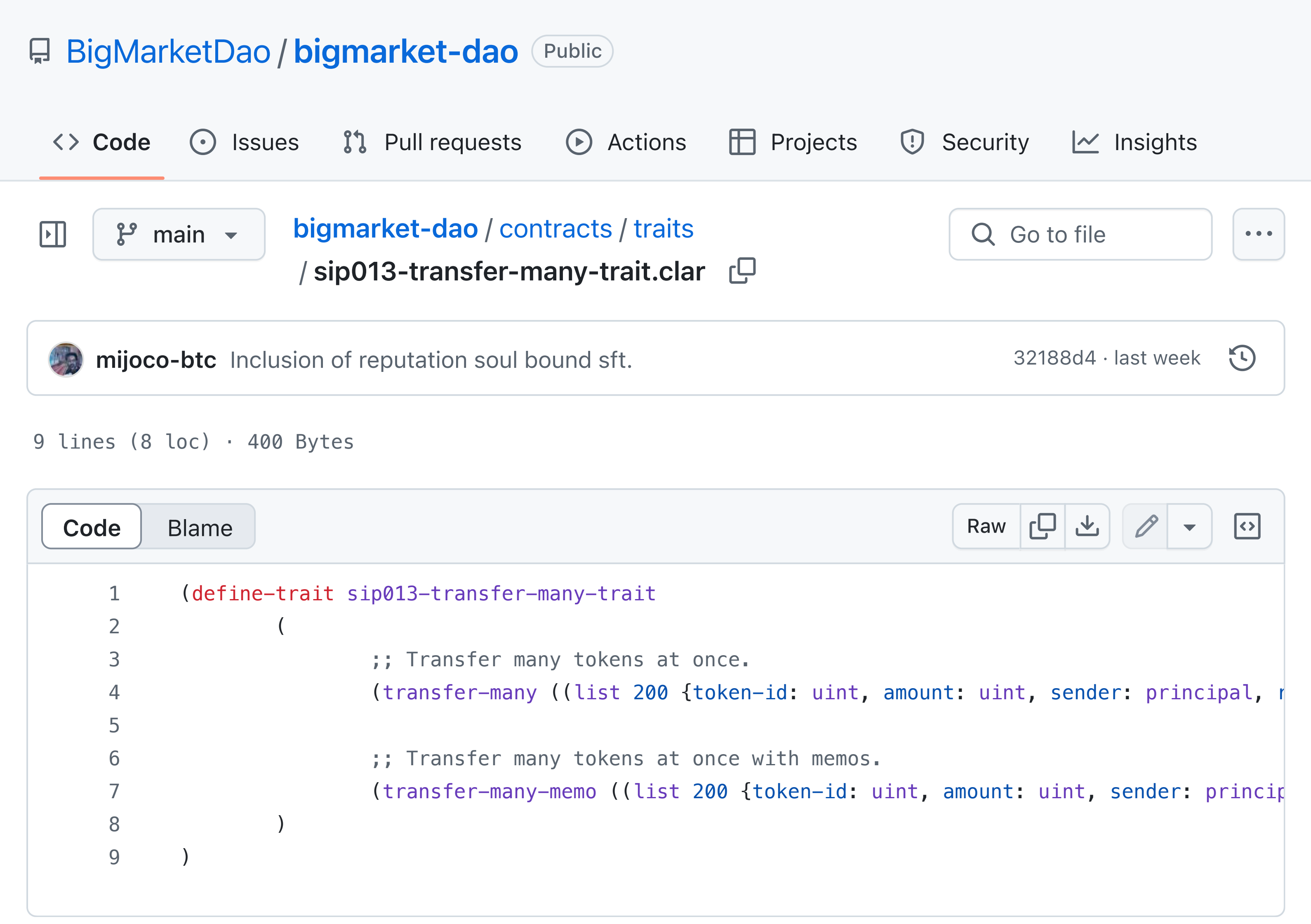
Go to file

mijoco-btc Inclusion of reputation soul bound sft. 32188d4 · last week

27 lines (20 loc) · 961 Bytes

Code Blame Raw

```
(define-trait sip013-semi-fungible-token-trait
        (
                ;; Get a token type balance of the passed principal.
                (get-balance (uint principal) (response uint uint))

                ;; Get the total SFT balance of the passed principal.
                (get-overall-balance (principal) (response uint uint))

                ;; Get the current total supply of a token type.
                (get-total-supply (uint) (response uint uint))

                ;; Get the overall SFT supply.
                (get-overall-supply () (response uint uint))

                ;; Get the number of decimal places of a token type.
                (get-decimals (uint) (response uint uint))

                ;; Get an optional token URI that represents metadata for a specific token.
                (get-token-uri (uint) (response (optional (string-ascii 256)) uint))

                ;; Transfer from one principal to another.
                (transfer (uint uint principal principal) (response bool uint))

                ;; Transfer from one principal to another with a memo.
                (transfer-memo (uint uint principal principal (buff 34)) (response bool uint)
        )
)
```



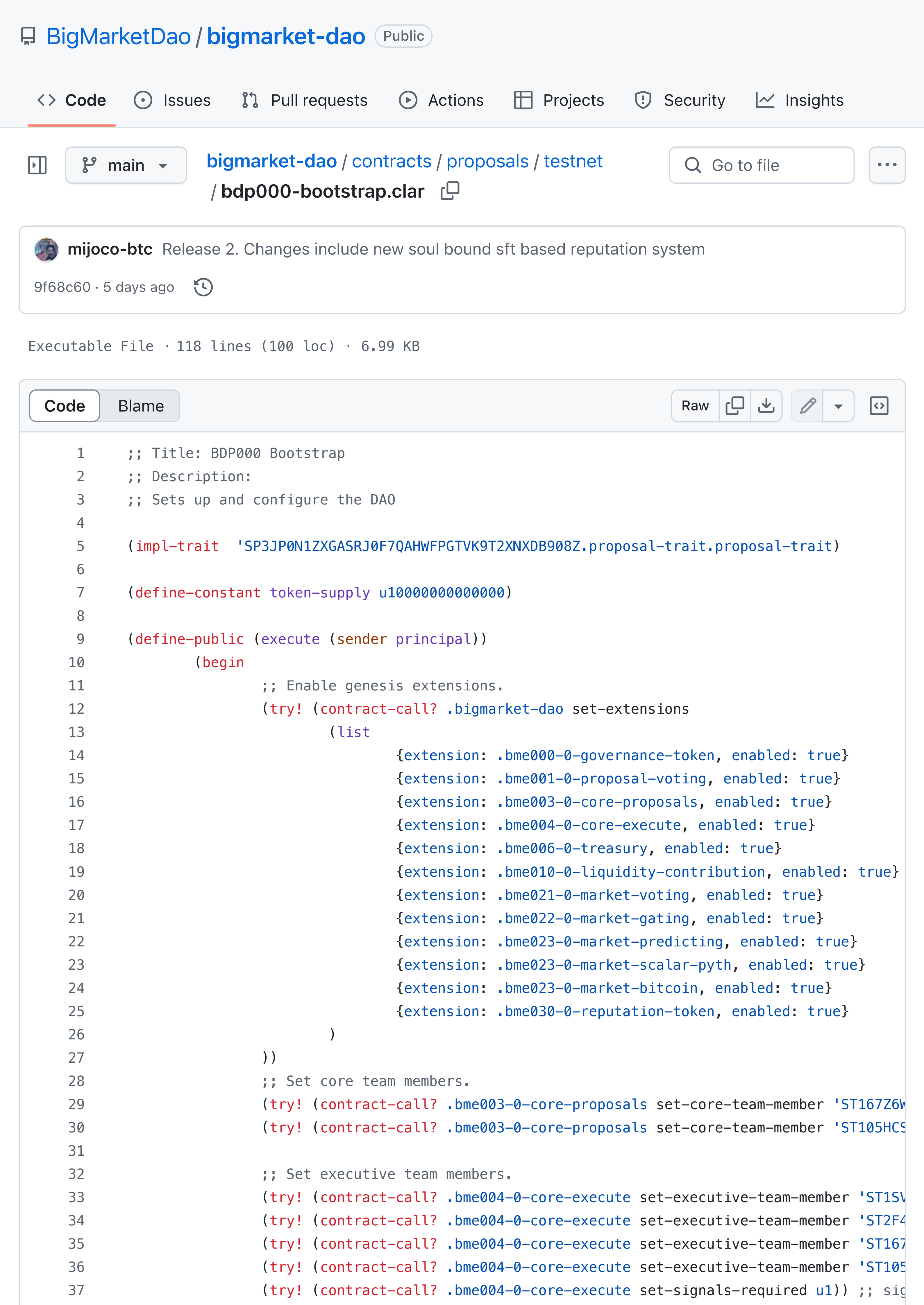
BigMarketDao / **bigmarket-dao** Public

bigmarket-dao / contracts / traits / **sip013-transfer-many-trait.clar**

mijoco-btc Inclusion of reputation soul bound sft. 32188d4 · last week

9 lines (8 loc) · 400 Bytes

```
(define-trait sip013-transfer-many-trait
	(
		;; Transfer many tokens at once.
		(transfer-many ((list 200 {token-id: uint, amount: uint, sender: principal, r

		;; Transfer many tokens at once with memos.
		(transfer-many-memo ((list 200 {token-id: uint, amount: uint, sender: princip
	)
)
```


BigMarketDao / bigmarket-dao Public

bigmarket-dao / contracts / proposals / testnet / bdp000-bootstrap.clar

mijoco-btc Release 2. Changes include new soul bound sft based reputation system

9f68c60 · 5 days ago

Executable File · 118 lines (100 loc) · 6.99 KB

```
;; Title: BDP000 Bootstrap
;; Description:
;; Sets up and configure the DAO

(impl-trait  'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.proposal-trait.proposal-trait)

(define-constant token-supply u10000000000000)

(define-public (execute (sender principal))
        (begin
                ;; Enable genesis extensions.
                (try! (contract-call? .bigmarket-dao set-extensions
                        (list
                                {extension: .bme000-0-governance-token, enabled: true}
                                {extension: .bme001-0-proposal-voting, enabled: true}
                                {extension: .bme003-0-core-proposals, enabled: true}
                                {extension: .bme004-0-core-execute, enabled: true}
                                {extension: .bme006-0-treasury, enabled: true}
                                {extension: .bme010-0-liquidity-contribution, enabled: true}
                                {extension: .bme021-0-market-voting, enabled: true}
                                {extension: .bme022-0-market-gating, enabled: true}
                                {extension: .bme023-0-market-predicting, enabled: true}
                                {extension: .bme023-0-market-scalar-pyth, enabled: true}
                                {extension: .bme023-0-market-bitcoin, enabled: true}
                                {extension: .bme030-0-reputation-token, enabled: true}
                        )
                ))
                ;; Set core team members.
                (try! (contract-call? .bme003-0-core-proposals set-core-team-member 'ST167Z6W
                (try! (contract-call? .bme003-0-core-proposals set-core-team-member 'ST105HCS

                ;; Set executive team members.
                (try! (contract-call? .bme004-0-core-execute set-executive-team-member 'ST1SV
                (try! (contract-call? .bme004-0-core-execute set-executive-team-member 'ST2F4
                (try! (contract-call? .bme004-0-core-execute set-executive-team-member 'ST167
                (try! (contract-call? .bme004-0-core-execute set-executive-team-member 'ST105
                (try! (contract-call? .bme004-0-core-execute set-signals-required u1)) ;; sig
```

```

;; configure prediction markets
;; allowedCreators = ["ST1SV7MYKRKKDG8PHSSKZ0W66DPKRPB5KV8ACN62G", "ST2F4ZBBV
(try! (contract-call? .bme022-0-market-gating set-merkle-root-by-principal .b
(try! (contract-call? .bme022-0-market-gating set-merkle-root-by-principal .b
(try! (contract-call? .bme022-0-market-gating set-merkle-root-by-principal .b

(try! (contract-call? .bme023-0-market-predicting set-resolution-agent 'ST167
(try! (contract-call? .bme023-0-market-predicting set-dev-fund 'ST1EEDB05014J
(try! (contract-call? .bme023-0-market-predicting set-dao-treasury .bme006-0-
(try! (contract-call? .bme023-0-market-predicting set-allowed-token 'ST31A25Y
(try! (contract-call? .bme023-0-market-predicting set-allowed-token .bme000-0
(try! (contract-call? .bme023-0-market-predicting set-allowed-token 'ST1F7QA2

(try! (contract-call? .bme023-0-market-scalar-pyth set-resolution-agent 'ST16
(try! (contract-call? .bme023-0-market-scalar-pyth set-dev-fund 'ST1EEDB05014
(try! (contract-call? .bme023-0-market-scalar-pyth set-dao-treasury .bme006-0
(try! (contract-call? .bme023-0-market-scalar-pyth set-allowed-token 'ST31A25
(try! (contract-call? .bme023-0-market-scalar-pyth set-allowed-token .bme000-
(try! (contract-call? .bme023-0-market-scalar-pyth set-allowed-token 'ST1F7QA

(try! (contract-call? .bme023-0-market-bitcoin set-resolution-agent 'ST167Z6W
(try! (contract-call? .bme023-0-market-bitcoin set-dev-fund 'ST1EEDB05014JVXS
(try! (contract-call? .bme023-0-market-bitcoin set-dao-treasury .bme006-0-tre

(try! (contract-call? .bme010-0-token-sale initialize-ido))

;; core team voting rights unlock over u105120 bitcoin block period
(try! (contract-call? .bme000-0-governance-token set-core-team-vesting
    (list
        {recipient: sender, start-block: burn-block-height, duration:
        {recipient: 'ST2F4ZBBV22RF2WYR424HKX5RDN6XRK19X37YEVGG, start
        {recipient: 'STNNX8QFM2MPJ35V9SNH6BMWYRJF8KVZC3XDZGVZ, start-
        {recipient: 'ST105HCS1RTR7D61EZET8CWNEF24ENEN3V6ARBYBJ, start
        {recipient: 'ST167Z6WFHMV0FZKFCRNWZ33WTB0DFBCW9M1FW3AY, start
    )
))
(try! (contract-call? .bme000-0-governance-token bmg-mint-many
    (list
        {amount: (/ (* u1500 token-supply) u10000), recipient: .bme00
    )
))

;; Entry levels (weight: 1)
(try! (contract-call? .bme030-0-reputation-token set-tier-weight u1 u1))
(try! (contract-call? .bme030-0-reputation-token set-tier-weight u2 u1))
(try! (contract-call? .bme030-0-reputation-token set-tier-weight u3 u1))

;; Contributor levels (weight: 2)
(try! (contract-call? .bme030-0-reputation-token set-tier-weight u4 u2))
(try! (contract-call? .bme030-0-reputation-token set-tier-weight u5 u2))
(try! (contract-call? .bme030-0-reputation-token set-tier-weight u6 u2))

;; Active community (weight: 3)
(try! (contract-call? .bme030-0-reputation-token set-tier-weight u7 u3))
(try! (contract-call? .bme030-0-reputation-token set-tier-weight u8 u3))
```

```
                    (try! (contract-call? .bme030-0-reputation-token set-tier-weight u9 u3))

                    ;; Project leads (weight: 5)
                    (try! (contract-call? .bme030-0-reputation-token set-tier-weight u10 u5))
                    (try! (contract-call? .bme030-0-reputation-token set-tier-weight u11 u5))
                    (try! (contract-call? .bme030-0-reputation-token set-tier-weight u12 u5))

                    ;; Strategic contributors (weight: 8)
                    (try! (contract-call? .bme030-0-reputation-token set-tier-weight u13 u8))
                    (try! (contract-call? .bme030-0-reputation-token set-tier-weight u14 u8))
                    (try! (contract-call? .bme030-0-reputation-token set-tier-weight u15 u8))

                    ;; Core stewards (weight: 13)
                    (try! (contract-call? .bme030-0-reputation-token set-tier-weight u16 u13))
                    (try! (contract-call? .bme030-0-reputation-token set-tier-weight u17 u13))
                    (try! (contract-call? .bme030-0-reputation-token set-tier-weight u18 u13))

                    ;; Founders / exec level (weight: 21)
                    (try! (contract-call? .bme030-0-reputation-token set-tier-weight u19 u21))
                    (try! (contract-call? .bme030-0-reputation-token set-tier-weight u20 u21))

                    (print "BigMarket DAO has risen.")
                    (ok true)
            )
    )
```

mijoco-btc Inclusion of reputation soul bound sft. 32188d4 · last week

Executable File · 268 lines (220 loc) · 8.35 KB

Code Blame Raw

```
;; Title: BME00 Governance Token
;; Synopsis:
;; This extension defines the governance token of BigMarket DAO.
;; Description:
;; The governance token is a simple SIP010-compliant fungible token
;; with some added functions to make it easier to manage by
;; BigMarket DAO proposals and extensions.
;; The operations vesting schedule and recipients can be updated (see current-key and
;; set-core-team-vesting) up till the first claim. If more recipients are added they
;; allocation is proportionally diluted.

(impl-trait .governance-token-trait.governance-token-trait)
(impl-trait 'SP3FBR2AGK5H9QBDH3EEN6DF8EK8JY7RX8QJ5SVTE.sip-010-trait-ft-standard.sip-010-trai
(impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.extension-trait.extension-trait)

(define-fungible-token bmg-token u10000000000000)
(define-fungible-token bmg-token-locked)

(define-constant core-team-max-vesting u1500000000000) ;; 15% of total supply (10,000,000 BIG

(define-constant err-unauthorised (err u3000))
(define-constant err-not-token-owner (err u3001))
(define-constant err-not-core-team (err u3002))
(define-constant err-no-vesting-schedule (err u3003))
(define-constant err-nothing-to-claim (err u3004))
(define-constant err-core-vesting-limit (err u3005))
(define-constant err-cliff-not-reached (err u3006))
(define-constant err-recipients-are-locked (err u3007))
(define-constant err-transfers-blocked (err u3008))

(define-data-var token-name (string-ascii 32) "BigMarket Governance Token")
(define-data-var token-symbol (string-ascii 10) "BIG")
(define-data-var token-uri (optional (string-utf8 256)) none)
(define-data-var token-decimals uint u6)
(define-data-var core-team-size uint u0)

(define-data-var token-price uint u100000)
(define-data-var transfers-active bool false)
```

```
(define-map core-team-vesting-tracker principal uint) ;; Tracks vested amount per recipient

;; ---- Vesting Storage ----
(define-data-var claim-made bool false)
(define-data-var current-key uint u0)
(define-map core-team-vesting {current-key: uint, recipient: principal}
  {total-amount: uint, start-block: uint, duration: uint, claimed: uint}
)

;; --- Authorisation check

(define-public (is-dao-or-extension)
        (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
)

;; ---- Vesting Methods ----

;; --- Vesting logic and sale
(define-public (set-transfers-active (new-transfers-active bool))
  (begin
    (try! (is-dao-or-extension))
    (var-set transfers-active new-transfers-active)
    (ok true)
  )
)
(define-read-only (get-transfers-active) (var-get transfers-active))

(define-public (set-token-price (new-token-price uint))
  (begin
    (try! (is-dao-or-extension))
    (var-set token-price new-token-price)
    (ok true)
  )
)

(define-public (set-core-team-vesting (core-team (list 200 {recipient: principal, start-block
  (begin
        (try! (is-dao-or-extension))
        (asserts! (not (var-get claim-made)) err-recipients-are-locked)
        (var-set current-key (+ u1 (var-get current-key)))
        (var-set core-team-size (len core-team))
        (as-contract (fold set-core-team-vesting-iter core-team (ok true)))
  )
)
(define-private (set-core-team-vesting-iter (item {recipient: principal, start-block: uint, d
        (begin
                (try! previous-result)
                ;;(asserts! (as-contract (contract-call? .bme004-0-core-execute is-executive-
                (let (
                                (amount (/ core-team-max-vesting (var-get core-team-size)))
                        )
                        (map-set core-team-vesting {current-key: (var-get current-key), recip
                                {total-amount: amount, start-block: (get start-block item), d
                        (map-set core-team-vesting-tracker (get recipient item) amount)
                        (print {event: "set-core-team-vesting", amount: amount, start-block:
                        (ok true)
```

```
                    )
            )
    )

    (define-public (core-claim)
      (let
        (
            (vesting (unwrap! (map-get? core-team-vesting {current-key: (var-get current-key), re
            (current-block burn-block-height)
                    (start-block (get start-block vesting))
                    (duration (get duration vesting))
                    (total-amount (get total-amount vesting))
                    (claimed (get claimed vesting))
                    (elapsed (if (> current-block start-block) (- current-block start-block) u0))
                    (vested (if (> elapsed duration) total-amount (/ (* total-amount elapsed) dur

    (define-public (set-token-uri (new-uri (optional (string-utf8 256))))
            (begin
                    (try! (is-dao-or-extension))
                    (ok (var-set token-uri new-uri))
            )
    )

    (define-private (bmg-mint-many-iter (item {amount: uint, recipient: principal}))
            (ft-mint? bmg-token (get amount item) (get recipient item))
    )

    (define-public (bmg-mint-many (recipients (list 200 {amount: uint, recipient: principal})))
    		(begin
    			(try! (is-dao-or-extension))
    			(ok (map bmg-mint-many-iter recipients))
    		)
    )

    ;; --- Public functions

    ;; sip-010-trait

    (define-public (transfer (amount uint) (sender principal) (recipient principal) (memo (optio
    		(begin
    			(asserts! (or (is-eq tx-sender sender) (is-eq contract-caller sender)) err-no
    		(asserts! (or (var-get transfers-active) (unwrap! (is-dao-or-extension) err-unauthori
    			(ft-transfer? bmg-token amount sender recipient)
    		)
    )

    (define-read-only (get-name)
    		(ok (var-get token-name))
    )

    (define-read-only (get-symbol)
    		(ok (var-get token-symbol))
    )

    (define-read-only (get-decimals)
    		(ok (var-get token-decimals))
    )

    (define-read-only (get-balance (who principal))
    		(ok (+ (ft-get-balance bmg-token who) (ft-get-balance bmg-token-locked who)))
    )

    (define-read-only (get-total-supply)
    		(ok (+ (ft-get-supply bmg-token) (ft-get-supply bmg-token-locked)))
    )

    (define-read-only (get-token-uri)
    		(ok (var-get token-uri))
    )

    ;; governance-token-trait

    (define-read-only (bmg-get-balance (who principal))
    		(get-balance who)
    )

    (define-read-only (bmg-has-percentage-balance (who principal) (factor uint))
    		(ok (>= (* (unwrap-panic (get-balance who)) factor) (* (unwrap-panic (get-total-suppl
    )

    (define-read-only (bmg-get-locked (owner principal))
    		(ok (ft-get-balance bmg-token-locked owner))
    )
```

```
;; --- Extension callback

(define-public (callback (sender principal) (memo (buff 34)))
        (ok true)
)
```

BigMarketDao / **bigmarket-dao** Public

Code Issues Pull requests Actions Projects Security Insights

main bigmarket-dao / contracts / extensions / **bme001-0-proposal-voting.clar**

Go to file

mijoco-btc Inclusion of reputation soul bound sft. 32188d4 · last week

248 lines (219 loc) · 9.83 KB

Code Blame Raw

```
;; Title: BME01 Proposal Voting
;; Synopsis:
;; Allows governance token holders to vote on and conclude proposals.
;; Description:
;; Once proposals are submitted, they are open for voting after a lead up time
;; passes. Any token holder may vote on an open proposal, where one token equals
;; one vote. Members can vote until the voting period is over. After this period
;; anyone may trigger a conclusion. The proposal will then be executed if the
;; votes in favour exceed the ones against by the custom majority if set or simple majority
;; otherwise. Votes may additionally be submitted as batched list of signed structured
;; voting messages using SIP-018.
;; The mechanism for voting requires Governance tokens to be burned in exchange for the
;; equivalent number of lock tokens - these can be re-exchanged after the vote is concluded.

(impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.extension-trait.extension-trait)
(use-trait proposal-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.proposal-trait.proposal-t

(define-constant err-unauthorised (err u3000))
(define-constant err-proposal-already-executed (err u3002))
(define-constant err-proposal-already-exists (err u3003))
(define-constant err-unknown-proposal (err u3004))
(define-constant err-proposal-already-concluded (err u3005))
(define-constant err-proposal-inactive (err u3006))
(define-constant err-proposal-not-concluded (err u3007))
(define-constant err-no-votes-to-return (err u3008))
(define-constant err-end-block-height-not-reached (err u3009))
(define-constant err-disabled (err u3010))
(define-constant err-not-majority (err u3011))

(define-constant structured-data-prefix 0x534950303138)
(define-constant message-domain-hash (sha256 (unwrap! (to-consensus-buff?
        {
                name: "BigMarket",
                version: "1.0.0",
                chain-id: chain-id
        }
    ) err-unauthorised)
))
(define-constant custom-majority-upper u10000)
```

```
(define-constant custom-majority-upper u10000)
(define-constant structured-data-header (concat structured-data-prefix message-domain-hash))

(define-map proposals
	principal
	{
		custom-majority: (optional uint), ;; u10000 = 100%
		votes-for: uint,
		votes-against: uint,
		start-burn-height: uint,
		end-burn-height: uint,
		concluded: bool,
		passed: bool,
		proposer: principal
	}
)

(define-map member-total-votes {proposal: principal, voter: principal} uint)

;; --- Authorisation check

(define-public (is-dao-or-extension)
	(ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
)

;; --- Internal DAO functions

;; Proposals

(define-public (add-proposal (proposal <proposal-trait>) (data {start-burn-height: uint, end-
	(begin
		(try! (is-dao-or-extension))
		(asserts! (is-none (contract-call? .bigmarket-dao executed-at proposal)) err-
		(asserts! (match (get custom-majority data) majority (> majority u5000) true)
		(print {event: "propose", proposal: proposal, proposer: tx-sender})
		(ok (asserts! (map-insert proposals (contract-of proposal) (merge {votes-for:
	)
)

;; --- Public functions

;; Proposals

(define-read-only (get-proposal-data (proposal principal))
	(map-get? proposals proposal)
)

;; Votes

(define-read-only (get-current-total-votes (proposal principal) (voter principal))
	(default-to u0 (map-get? member-total-votes {proposal: proposal, voter: voter}))
)

(define-public (vote (amount uint) (for bool) (proposal principal) (reclaim-proposal (optiona
  (process-vote-internal amount for proposal tx-sender reclaim-proposal)
)

```

```
(define-public (batch-vote (votes (list 50 {message: (tuple
                                              (attestation (string-ascii 100))
                                              (proposal principal)
                                              (vote bool)
                                              (voter principal)
                                              (amount uint)
                                              (reclaim-proposal (optional principal))),
                                 signature: (buff 65)})))
  (begin
    (ok (fold fold-vote votes u0))
  )
)

(define-private (fold-vote  (input-vote {message: (tuple
                                              (attestation (string-ascii 100))
```

```
	)
)

;; Conclusion

(define-public (conclude (proposal <proposal-trait>))
	(let
		(
			(proposal-data (unwrap! (map-get? proposals (contract-of proposal)) e
			(passed
				(match (get custom-majority proposal-data)
					majority (> (* (get votes-for proposal-data) custom-m
					(> (get votes-for proposal-data) (get votes-against p
				)
			)
		)
		(asserts! (not (get concluded proposal-data)) err-proposal-already-concluded)
		(asserts! (>= burn-block-height (get end-burn-height proposal-data)) err-end-
		(map-set proposals (contract-of proposal) (merge proposal-data {concluded: tr
		(print {event: "conclude", proposal: proposal, passed: passed})
		(and passed (try! (contract-call? .bigmarket-dao execute proposal tx-sender))
	(try! (contract-call? .bme030-0-reputation-token mint tx-sender u3 u5))
		(ok passed)
	)
)

;; Reclamation

(define-public (reclaim-votes (proposal (optional principal)))
	(let
		(
			(reclaim-proposal (unwrap! proposal err-unknown-proposal))
			(proposal-data (unwrap! (map-get? proposals reclaim-proposal) err-unk
                                (votes (unwrap! (map-get? member-total-votes {proposal: reclaim-propo
                        )
                        (asserts! (get concluded proposal-data) err-proposal-not-concluded)
                        (map-delete member-total-votes {proposal: reclaim-proposal, voter: tx-sender}
                (try! (contract-call? .bme030-0-reputation-token mint tx-sender u5 u3))
                        (contract-call? .bme000-0-governance-token bmg-unlock votes tx-sender)
                )
        )

        ;; --- Extension callback

        (define-public (callback (sender principal) (memo (buff 34)))
                (ok true)
        )

        (define-read-only (verify-signature (hash (buff 32)) (signature (buff 65)) (signer principal)
                (is-eq (principal-of? (unwrap! (secp256k1-recover? hash signature) false)) (ok signer
        )

        (define-read-only (verify-signed-structured-data (structured-data-hash (buff 32)) (signature
                (verify-signature (sha256 (concat structured-data-header structured-data-hash)) signa
        )

        (define-read-only (verify-signed-tuple
            (message-data (tuple
                            (attestation (string-ascii 100))
                            (proposal principal)
                            (vote bool)
                            (voter principal)
                            (amount uint)))
            (signature (buff 65))
            (signer principal))
          (let
            (
              ;; Compute the structured data hash
                (structured-data-hash (sha256 (unwrap! (to-consensus-buff? message-data) err-unauthor
            )
            ;; Verify the signature using the computed hash
            (ok (verify-signed-structured-data structured-data-hash signature signer))
          )
        )
```

BigMarketDao / **bigmarket-dao** Public

Code Issues Pull requests Actions Projects Security Insights

main

bigmarket-dao / contracts / extensions / **bme003-0-core-proposals.clar**

Go to file

mijoco-btc attempt to resolve clarinet toml issues around pyth. code is basicall...

767bb51 · 3 weeks ago

Executable File · 73 lines (59 loc) · 2.53 KB

Code Blame Raw

```
;; Title: BME03 Core Proposals
;; Synopsis:
;; This extension allows for the creation of core proposals by a few trusted
;; principals.
;; Description:
;; Only a list of trusted principals, designated as the
;; "core team", can create core proposals. The core proposal
;; extension has an optional ~3 month sunset period, after which no more core
;; proposals can be made - set it to 0 to disable. The core team members, sunset period, and
;; core vote duration can be changed by means of a future proposal.

(impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.extension-trait.extension-trait)
(use-trait proposal-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.proposal-trait.proposal-t

(define-data-var core-team-sunset-height uint u0) ;; does not expire by default - can be char

(define-constant err-unauthorised (err u3300))
(define-constant err-not-core-team-member (err u3301))
(define-constant err-sunset-height-reached (err u3302))
(define-constant err-sunset-height-in-past (err u3303))

(define-map core-team principal bool)

;; --- Authorisation check

(define-public (is-dao-or-extension)
	(ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
)

;; --- Internal DAO functions

(define-public (set-core-team-sunset-height (height uint))
	(begin
		(try! (is-dao-or-extension))
		(asserts! (> height burn-block-height) err-sunset-height-in-past)
		(ok (var-set core-team-sunset-height height))
	)
```

```
)

(define-public (set-core-team-member (who principal) (member bool))
	(begin
		(try! (is-dao-or-extension))
		(print {event: "set-core-team-member", who: who, member: member})
		(ok (map-set core-team who member))
	)
)

;; --- Public functions

(define-read-only (is-core-team-member (who principal))
	(default-to false (map-get? core-team who))
)

(define-public (core-propose (proposal <proposal-trait>) (start-burn-height uint) (duration u
	(begin
		(asserts! (is-core-team-member tx-sender) err-not-core-team-member)
		(asserts! (or (is-eq (var-get core-team-sunset-height) u0) (< burn-block-heig
		(contract-call? .bme001-0-proposal-voting add-proposal proposal
			{
				start-burn-height: start-burn-height,
				end-burn-height: (+ start-burn-height duration),
				custom-majority: custom-majority,
				proposer: tx-sender ;; change to original submitter
			}
		)
	)
)

;; --- Extension callback

(define-public (callback (sender principal) (memo (buff 34)))
	(ok true)
)
```

BigMarketDao / **bigmarket-dao** Public

Code · Issues · Pull requests · Actions · Projects · Security · Insights

main

bigmarket-dao / contracts / extensions / **bme004-0-core-execute.clar**

Go to file

mijoco-btc attempt to resolve clarinet toml issues around pyth. code is basicall...

767bb51 · 3 weeks ago

Executable File · 99 lines (80 loc) · 3.62 KB

Code Blame Raw

```
;; Title: BME04 Core Execute
;; Synopsis:
;; This extension allows a small number of very trusted principals to immediately
;; execute a proposal once a super majority is reached.
;; Description:
;; An extension meant for the bootstrapping period of a DAO. It temporarily gives
;; some very trusted principals the ability to perform an "executive action";
;; meaning, they can skip the voting process to immediately executive a proposal.
;; The Core execute extension has an optional sunset period of ~1 month from deploy
;; time, set it to 0 to disable. The core executive team, parameters, and sunset period may b
;; by means of a future proposal.

(impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.extension-trait.extension-trait)
(use-trait proposal-trait  'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.proposal-trait.proposal-

(define-data-var executive-team-sunset-height uint u0) ;; does not expire by default - can be

(define-constant err-unauthorised (err u3400))
(define-constant err-not-executive-team-member (err u3401))
(define-constant err-already-executed (err u3402))
(define-constant err-sunset-height-reached (err u3403))
(define-constant err-sunset-height-in-past (err u3404))

(define-map executive-team principal bool)
(define-map executive-action-signals {proposal: principal, team-member: principal} bool)
(define-map executive-action-signal-count principal uint)

(define-data-var executive-signals-required uint u1) ;; signals required for an executive act

;; --- Authorisation check

(define-public (is-dao-or-extension)
        (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
)

;; --- Internal DAO functions

```

```
(define-public (set-executive-team-sunset-height (height uint))
	(begin
		(try! (is-dao-or-extension))
		(asserts! (> height burn-block-height) err-sunset-height-in-past)
		(ok (var-set executive-team-sunset-height height))
	)
)

(define-public (set-executive-team-member (who principal) (member bool))
	(begin
		(try! (is-dao-or-extension))
		(ok (map-set executive-team who member))
	)
)

(define-public (set-signals-required (new-requirement uint))
	(begin
		(try! (is-dao-or-extension))
		(ok (var-set executive-signals-required new-requirement))
	)
)

;; --- Public functions

(define-read-only (is-executive-team-member (who principal))
	(default-to false (map-get? executive-team who))
)

(define-read-only (has-signalled (proposal principal) (who principal))
	(default-to false (map-get? executive-action-signals {proposal: proposal, team-member
)

(define-read-only (get-signals-required)
	(var-get executive-signals-required)
)

(define-read-only (get-signals (proposal principal))
	(default-to u0 (map-get? executive-action-signal-count proposal))
)

(define-public (executive-action (proposal <proposal-trait>))
	(let
		(
			(proposal-principal (contract-of proposal))
			(signals (+ (get-signals proposal-principal) (if (has-signalled propo
		)
		(asserts! (is-executive-team-member tx-sender) err-not-executive-team-member)
		(asserts! (or (is-eq (var-get executive-team-sunset-height) u0) (< burn-block
		(and (>= signals (var-get executive-signals-required))
			(try! (contract-call? .bigmarket-dao execute proposal tx-sender))
		)
		(map-set executive-action-signals {proposal: proposal-principal, team-member:
		(map-set executive-action-signal-count proposal-principal signals)
		(ok signals)
	)
)
```

```

;; --- Extension callback

(define-public (callback (sender principal) (memo (buff 34)))
        (ok true)
)
```

BigMarketDao / **bigmarket-dao** Public

Code · Issues · Pull requests · Actions · Projects · Security · Insights

main

bigmarket-dao / contracts / extensions / **bme006-0-treasury.clar**

Go to file

mijoco-btc attempt to resolve clarinet toml issues around pyth. code is basicall...

767bb51 · 3 weeks ago

Executable File · 155 lines (126 loc) · 4.92 KB

Code Blame Raw

```
;; Title: BME006 Treasury
;; Synopsis:
;; A treasury that can manage STX, SIP009, SIP010, and SIP013 tokens.
;; Description:
;; An extension contract that is meant to hold tokens on behalf of the
;; DAO. It can hold and transfer STX, SIP009, SIP010, and SIP013 tokens.
;; They can be deposited by simply transferring them to the contract.
;; Any extension or executing proposal can trigger transfers.
;; Technically, the ExecutorDAO core can hold and transfer tokens
;; directly. The treasury extension merely adds a bit of separation.

(impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.extension-trait.extension-trait)

(define-constant err-unauthorised (err u3000))

;; --- Transferable traits

(define-trait sip009-transferable
        (
                (transfer (uint principal principal) (response bool uint))
        )
)

(define-trait sip010-transferable
        (
                (transfer (uint principal principal (optional (buff 34))) (response bool uint
        )
)

(define-trait sip013-transferable
        (
                (transfer (uint uint principal principal) (response bool uint))
                (transfer-memo (uint uint principal principal (buff 34)) (response bool uint)
        )
)

(define-trait sip013-transferable-many
    (
        (transfer-many ((list 200 {token-id: uint, amount: uint, sender: principal, r
        (transfer-many-memo ((list 200 {token-id: uint, amount: uint, sender: princip
    )
)

;; --- Authorisation check

(define-public (is-dao-or-extension)
    (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
)

;; --- Internal DAO functions

;; STX

(define-public (stx-transfer (amount uint) (recipient principal) (memo (optional (buff 34))))
    (begin
        (try! (is-dao-or-extension))
        (match memo to-print (print to-print) 0x)
        (as-contract (stx-transfer? amount tx-sender recipient))
    )
)

(define-public (stx-transfer-many (transfers (list 200 {amount: uint, recipient: principal, m
    (begin
        (try! (is-dao-or-extension))
        (as-contract (fold stx-transfer-many-iter transfers (ok true)))
    )
)

;; SIP009

(define-public (sip009-transfer (token-id uint) (recipient principal) (asset <sip009-transfer
    (begin
        (try! (is-dao-or-extension))
        (as-contract (contract-call? asset transfer token-id tx-sender recipient))
    )
)

(define-public (sip009-transfer-many (data (list 200 {token-id: uint, recipient: principal}))
    (begin
        (as-contract (fold sip009-transfer-many-iter data asset))
        (ok true)
    )
)

;; SIP010

(define-public (sip010-transfer (amount uint) (recipient principal) (memo (optional (buff 34)
    (begin
        (try! (is-dao-or-extension))
        (as-contract (contract-call? asset transfer amount tx-sender recipient memo))
    )
)

```

```
(define-public (sip010-transfer-many (data (list 200 {amount: uint, recipient: principal, mem
    (begin
        (as-contract (fold sip010-transfer-many-iter data asset))
        (ok true)
    )
)

;; SIP013

(define-public (sip013-transfer (token-id uint) (amount uint) (recipient principal) (memo (op
    (begin
        (try! (is-dao-or-extension))
        (as-contract (match memo memo-buff
            (contract-call? asset transfer-memo token-id amount tx-sender recipie
            (contract-call? asset transfer token-id amount tx-sender recipient)
        ))
    )
)

(define-public (sip013-transfer-many (transfers (list 200 {token-id: uint, amount: uint, send
    (begin
        (try! (is-dao-or-extension))
        (as-contract (contract-call? asset transfer-many transfers))
    )
)

(define-public (sip013-transfer-many-memo (transfers (list 200 {token-id: uint, amount: uint,
    (begin
        (try! (is-dao-or-extension))
        (as-contract (contract-call? asset transfer-many-memo transfers))
    )
)

;; --- Iterator functions

(define-private (stx-transfer-many-iter (data {amount: uint, recipient: principal, memo: (opt
    (begin
        (try! previous-result)
        (match (get memo data) to-print (print to-print) 0x)
        (stx-transfer? (get amount data) tx-sender (get recipient data))
    )
)

(define-private (sip009-transfer-many-iter (data {token-id: uint, recipient: principal}) (ass
    (begin
        (unwrap-panic (contract-call? asset transfer (get token-id data) tx-sender (g
        asset
    )
)

(define-private (sip010-transfer-many-iter (data {amount: uint, recipient: principal, memo: (
    (begin
        (unwrap-panic (contract-call? asset transfer (get amount data) tx-sender (get
        asset
    )
)
```

```

;; --- Extension callback

(define-public (callback (sender principal) (memo (buff 34)))
	(ok true)
)
```

BigMarketDao / **bigmarket-dao** Public

Code Issues Pull requests Actions Projects Security Insights

main bigmarket-dao / contracts / extensions / **bme010-0-liquidity-contribution.clar** Go to file

mijoco-btc Release 2. Changes include new soul bound sft based reputation system

9f68c60 · 5 days ago

Executable File · 62 lines (51 loc) · 1.96 KB

Code Blame Raw

```
;; Title: BME010 Reputation-Gated Liquidity Contribution
;; Synopsis:
;; Accept STX from contributors and reward them with BIGR reputation tokens
;; Description:
;; Users are rewarded with BIGR by contributing STX to the DAO treasury.
;; BIGR is used to claim BIG through the main reputation contract.
;; The rate is set by the DAO and can be updated as needed.

(impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.extension-trait.extension-trait)

;; Constants and Errors
(define-constant err-unauthorised (err u5000))
(define-constant err-zero-amount (err u5001))

;; Contract variables
(define-data-var stx-to-bigr-rate uint u10) ;; Default: 1 STX = 10 BIGR

(define-map stx-contributions {who: principal} uint)

;; Authorization check
(define-public (is-dao-or-extension)
  (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-extens
)

;; DAO can update the reward rate
(define-public (set-liquidity-reward-rate (new-rate uint))
  (begin
    (try! (is-dao-or-extension))
    (var-set stx-to-bigr-rate new-rate)
    (ok true)
  )
)
(define-read-only (get-liquidity-reward-rate)
        (var-get stx-to-bigr-rate)
)

(define-public (contribute-stx (amount uint))
  (let (
        (user tx-sender)
        (rate (var-get stx-to-bigr-rate))
        (bigr-earned (* amount rate))
        (existing (default-to u0 (map-get? stx-contributions {who: user})))
    )
    (asserts! (> amount u0) err-zero-amount)

    ;; Transfer STX to the DAO treasury
    (try! (stx-transfer? amount user .bme006-0-treasury))

    ;; Record contribution
    (map-set stx-contributions {who: user} (+ existing amount))

    ;; Mint BIGR to the contributor
    (try! (contract-call? .bme030-0-reputation-token mint user u7 bigr-earned))
    (print {event: "liquidity_contribution", from: user, amount: amount, bigr: bigr-earned})
    (ok bigr-earned)
  )
)

;; Extension trait callback stub
(define-public (callback (sender principal) (memo (buff 34)))
  (ok true)
)
```

mijoco-btc Inclusion of reputation soul bound sft. 32188d4 · last week

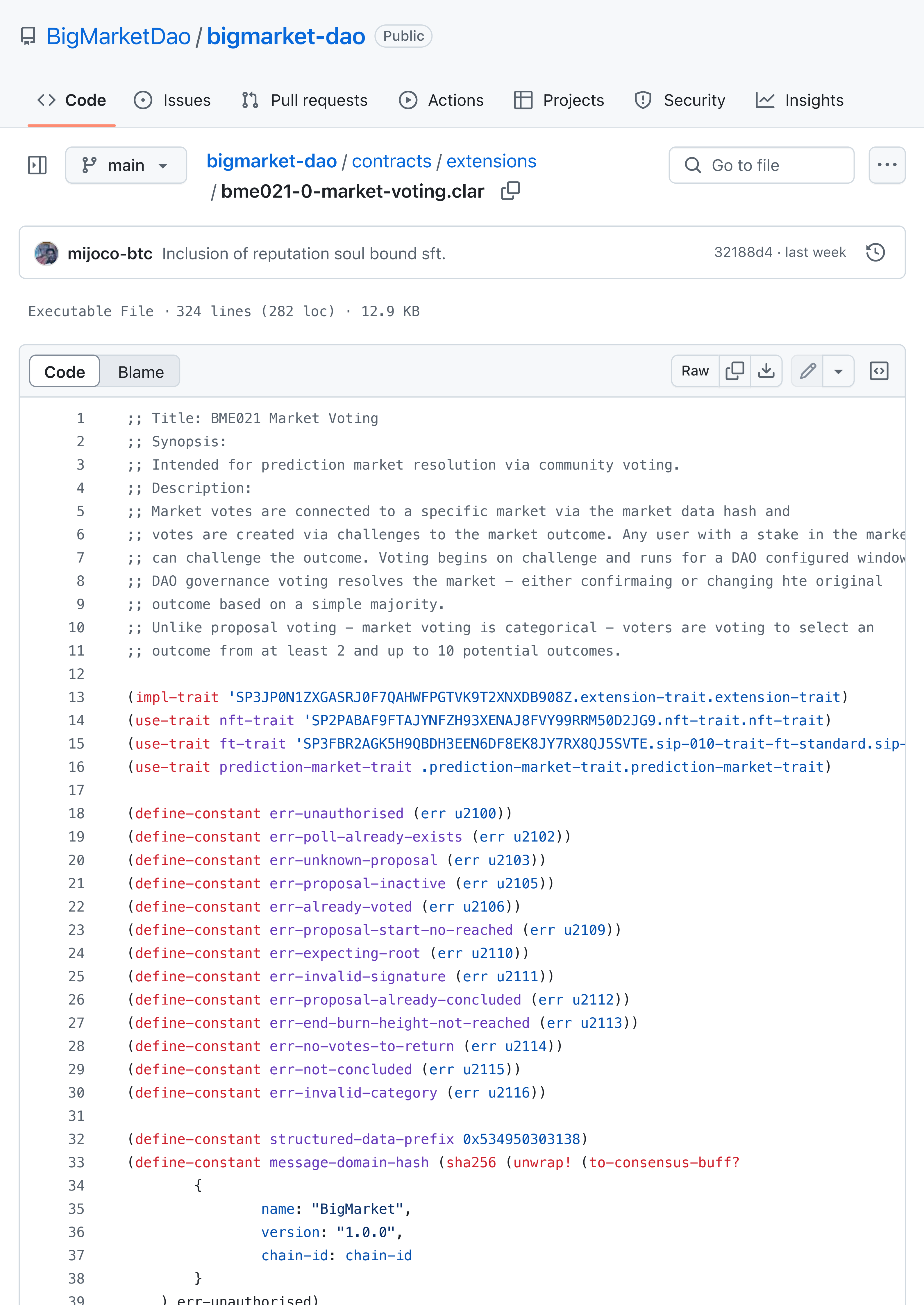
Executable File · 175 lines (142 loc) · 6.56 KB

Code Blame Raw

```
;; Title: BME010 Token Sale
;; Synopsis:
;; Enables token sale for govenernance tokens.
;; Description:
;; Allows to token sale over 6 stages with token price set at each stage by the current DAO.
;; Contract allows any stage to be cancelled and for tokens to be reclaimed.
;; Listing via a DEX is not supported but can be enabled at any stage
;;

(impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.extension-trait.extension-trait)

(define-constant err-unauthorised (err u5000))
(define-constant err-invalid-stage (err u5001))
(define-constant err-stage-sold-out (err u5002))
(define-constant err-nothing-to-claim (err u5003))
(define-constant err-no-more-stages (err u5005))
(define-constant err-already-cancelled (err u5006))
(define-constant err-no-purchase (err u5007))
(define-constant err-stage-not-cancelled (err u5008))
(define-constant err-stage-cancelled (err u5009))

(define-data-var current-stage uint u1) ;; IDO starts at Stage 1
(define-data-var current-stage-start uint burn-block-height) ;; Tracks burn-block-height wher

(define-map ido-stage-details uint
  {price: uint, max-supply: uint, tokens-sold: uint, cancelled: bool})
(define-map ido-purchases {stage: uint, buyer: principal} uint) ;; Tracks purchases

;; --- Authorisation check

(define-public (is-dao-or-extension)
        (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
)

(define-read-only (get-ido-stages)
  (list
    (map-get? ido-stage-details u1)
    (map-get? ido-stage-details u2)
    (map-get? ido-stage-details u3)
    (map-get? ido-stage-details u3)
    (map-get? ido-stage-details u4)
    (map-get? ido-stage-details u5)
    (map-get? ido-stage-details u6)
  )
)

(define-read-only (get-ido-user-for-stage (stage uint) (who principal))
  (map-get? ido-purchases {stage: stage, buyer: who})
)

(define-read-only (get-ido-user (who principal))
  (list
    (match (map-get? ido-purchases {stage: u1, buyer: who}) value value u0 )
    (match (map-get? ido-purchases {stage: u2, buyer: who}) value value u0 )
    (match (map-get? ido-purchases {stage: u3, buyer: who}) value value u0 )
    (match (map-get? ido-purchases {stage: u4, buyer: who}) value value u0 )
    (match (map-get? ido-purchases {stage: u5, buyer: who}) value value u0 )
  )
)

;; --- Internal DAO functions

(define-public (initialize-ido)
  (begin
    (try! (is-dao-or-extension))

    ;; Set up each stage
    (map-set ido-stage-details u1 {price: u5, max-supply: u600000000000, tokens-sold: u0, car
    (map-set ido-stage-details u2 {price: u6, max-supply: u833333000000, tokens-sold: u0, car
    (map-set ido-stage-details u3 {price: u7, max-supply: u1071429000000, tokens-sold: u0, ca
    (map-set ido-stage-details u4 {price: u8, max-supply: u1250000000000, tokens-sold: u0, ca
    (map-set ido-stage-details u5 {price: u10, max-supply: u1500000000000, tokens-sold: u0, c
    (map-set ido-stage-details u6 {price: u20, max-supply: u1000000000000, tokens-sold: u0, c

    (print {event: "ido-initialized"})
    (ok true)
  )
)

(define-public (buy-ido-tokens (stx-amount uint))
  (let (
    (stage (var-get current-stage))
    (stage-info (unwrap! (map-get? ido-stage-details stage) err-invalid-stage))
    (bmg-price (get price stage-info))
    (max-supply (get max-supply stage-info))
    (tokens-sold (get tokens-sold stage-info))
    (sender tx-sender)
                (cancelled (get cancelled stage-info))
    (current-stake (default-to u0 (map-get? ido-purchases {stage: stage, buyer: tx-sender})))
    (tokens-to-buy (* stx-amount bmg-price))
        )

    ;; Ensure enough supply remains
    (asserts! (<= (+ tokens-sold tokens-to-buy) max-supply) err-stage-sold-out)
    (asserts! (not cancelled) err-stage-cancelled)

```

```
      ;; Accept STX payment
      (try! (stx-transfer? stx-amount tx-sender .bme006-0-treasury))

      ;; Mint tokens directly to the buyer
      (try! (as-contract (contract-call? .bme000-0-governance-token bmg-mint tokens-to-buy send

      ;; Update stage details
      (map-set ido-stage-details stage (merge stage-info {tokens-sold: (+ tokens-sold tokens-to
      (map-set ido-purchases {stage: stage, buyer: tx-sender} (+ current-stake tokens-to-buy))

      (print {event: "ido-purchase", buyer: tx-sender, stage: stage, tokens: tokens-to-buy, stx

      (ok tokens-to-buy)
    )
  )

  (define-public (advance-ido-stage)
    (begin
      (try! (is-dao-or-extension))
      (let (
        (stage (var-get current-stage))
        (stage-info (unwrap! (map-get? ido-stage-details stage) err-invalid-stage))
      )

      (asserts! (not (get cancelled stage-info)) err-already-cancelled) ;; Ensure not already c
      (asserts! (< stage u6) err-no-more-stages) ;; Can't go past stage 6
      (var-set current-stage (+ u1 stage)) ;; Move to the next stage

      ;; Use burn-block-height to track when the stage starts
      (var-set current-stage-start burn-block-height)

      (print {event: "ido-stage-advanced", new-stage: (var-get current-stage), burn-start: burr
      (ok stage)
      )

    )
  )

  (define-public (cancel-ido-stage)
    (begin
      (try! (is-dao-or-extension))

      (let ((stage (var-get current-stage))
            (stage-info (unwrap! (map-get? ido-stage-details stage) err-invalid-stage)))

        (asserts! (not (get cancelled stage-info)) err-already-cancelled) ;; Ensure not already

        ;; Update the stage's cancelled flag
        (map-set ido-stage-details stage (merge stage-info {cancelled: true}))

        (print {event: "cancel-ido-stage", stage: stage})
        (ok true)
      )
    )
  )

```

```
(define-public (claim-ido-refund)
  (let ((stage (var-get current-stage))
        (purchase-amount (unwrap! (map-get? ido-purchases {stage: stage, buyer: tx-sender}) e
        (stage-info (unwrap! (map-get? ido-stage-details stage) err-invalid-stage))
        (price (get price stage-info))
        (sender tx-sender)
        )
    ;; Ensure stage is actually cancelled
    (asserts! (get cancelled stage-info) err-stage-not-cancelled)
    ;; Transfer STX back to the buyer / burn the bdg
    (try! (as-contract (contract-call? .bme006-0-treasury stx-transfer (* purchase-amount pri
    (try! (as-contract (contract-call? .bme000-0-governance-token bmg-burn purchase-amount se
    ;; Remove the purchase record
    (map-delete ido-purchases {stage: stage, buyer: tx-sender})
    (print {event: "ido-refund", buyer: tx-sender, refunded: purchase-amount, stage: stage})
    (ok purchase-amount)
  )
)

;; --- Extension callback

(define-public (callback (sender principal) (memo (buff 34)))
        (ok true)
)
```


BigMarketDao / bigmarket-dao Public
Code Issues Pull requests Actions Projects Security Insights
main
bigmarket-dao / contracts / extensions / bme021-0-market-voting.clar
Go to file
mijoco-btc Inclusion of reputation soul bound sft. 32188d4 · last week
Executable File · 324 lines (282 loc) · 12.9 KB
Code Blame Raw

```
;; Title: BME021 Market Voting
;; Synopsis:
;; Intended for prediction market resolution via community voting.
;; Description:
;; Market votes are connected to a specific market via the market data hash and
;; votes are created via challenges to the market outcome. Any user with a stake in the marke
;; can challenge the outcome. Voting begins on challenge and runs for a DAO configured window
;; DAO governance voting resolves the market - either confirmaing or changing hte original
;; outcome based on a simple majority.
;; Unlike proposal voting - market voting is categorical - voters are voting to select an
;; outcome from at least 2 and up to 10 potential outcomes.

(impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.extension-trait.extension-trait)
(use-trait nft-trait 'SP2PABAF9FTAJYNFZH93XENAJ8FVY99RRM50D2JG9.nft-trait.nft-trait)
(use-trait ft-trait 'SP3FBR2AGK5H9QBDH3EEN6DF8EK8JY7RX8QJ5SVTE.sip-010-trait-ft-standard.sip-
(use-trait prediction-market-trait .prediction-market-trait.prediction-market-trait)

(define-constant err-unauthorised (err u2100))
(define-constant err-poll-already-exists (err u2102))
(define-constant err-unknown-proposal (err u2103))
(define-constant err-proposal-inactive (err u2105))
(define-constant err-already-voted (err u2106))
(define-constant err-proposal-start-no-reached (err u2109))
(define-constant err-expecting-root (err u2110))
(define-constant err-invalid-signature (err u2111))
(define-constant err-proposal-already-concluded (err u2112))
(define-constant err-end-burn-height-not-reached (err u2113))
(define-constant err-no-votes-to-return (err u2114))
(define-constant err-not-concluded (err u2115))
(define-constant err-invalid-category (err u2116))

(define-constant structured-data-prefix 0x534950303138)
(define-constant message-domain-hash (sha256 (unwrap! (to-consensus-buff?
        {
                name: "BigMarket",
                version: "1.0.0",
                chain-id: chain-id
        }
    ) err-unauthorised)
```

```
))

(define-constant structured-data-header (concat structured-data-prefix message-domain-hash))

(define-data-var voting-duration uint u288)

(define-map resolution-polls
    {market: principal, market-id: uint}
    {
  votes: (list 10 uint), ;; votes for each category. NB with 2 categories the votes at 0 ar
        end-burn-height: uint,
        proposer: principal,
        concluded: bool,
  num-categories: uint,
  winning-category: (optional uint),
    }
)
(define-map member-total-votes {market-id: uint, voter: principal} uint)

;; --- Authorisation check

(define-public (is-dao-or-extension)
    (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
)

;; --- Internal DAO functions
(define-public (set-voting-duration (new-duration uint))
 (begin
  (try! (is-dao-or-extension))
  (var-set voting-duration new-duration)
  (ok true)
 )
)

;; called by a staker in a market to begin dispute resolution process
(define-public (create-market-vote
  (market <prediction-market-trait>)
  (market-id uint)
  (empty-votes (list 10 uint))
  (num-categories uint)
 )
 (let
  (
   (original-sender tx-sender)
  )
  (asserts! (is-none (map-get? resolution-polls {market-id: market-id, market: (contract-of
  (asserts! (is-eq (len empty-votes) num-categories) err-poll-already-exists)
        ;; a user with stake can propose but only for a market in the correct state i
  (try! (as-contract (contract-call? market dispute-resolution market-id original-sender)))

  ;; Register the poll
  (map-set resolution-polls {market-id: market-id, market: (contract-of market)}
   {
   votes: empty-votes,
   end-burn-height: (+ burn-block-height (var-get voting-duration)),
   proposer: tx-sender,
        concluded: false,
        num-categories: num-categories,
        winning-category: none}
      )

      ;; Emit an event for the new poll
      (print {event: "create-market-vote", market-id: market-id, proposer: tx-sender, market: m
      (ok true)
    )
  )

  ;; --- Public functions

  (define-read-only (get-poll-data (market principal) (market-id uint))
        (map-get? resolution-polls {market-id: market-id, market: market})
```

```
)

(define-read-only (verify-signed-structured-data (structured-data-hash (buff 32)) (signature
        (verify-signature (sha256 (concat structured-data-header structured-data-hash)) signa
)

;; Conclusion

(define-read-only (get-poll-status (market principal) (market-id uint))
    (let
        (
            (poll-data (unwrap! (map-get? resolution-polls {market-id: market-id, market: mar
            (is-active (< burn-block-height (get end-burn-height poll-data)))
    )
    (ok {active: is-active, concluded: (get concluded poll-data), votes: (get votes poll-
  )
)


(define-public (conclude-market-vote (market <prediction-market-trait>) (market-id uint))
    (let
        (
   (poll-data (unwrap! (map-get? resolution-polls {market-id: market-id, market: (contract
   (votes (get votes poll-data))
   (winning-category (get max-index (find-max-category votes)))
   (total-votes (fold + votes u0))
   (winning-votes (unwrap! (element-at? votes winning-category) err-already-voted))
   (result (try! (contract-call? market resolve-market-vote market-id winning-category)))
        )
        (asserts! (not (get concluded poll-data)) err-proposal-already-concluded)
        (asserts! (>= burn-block-height (get end-burn-height poll-data)) err-end-burn
        (map-set resolution-polls {market-id: market-id, market: (contract-of market)
        (print {event: "conclude-market-vote", market-id: market-id, winning-category
  (try! (contract-call? .bme030-0-reputation-token mint tx-sender u3 u3))
        (ok winning-category)
    )
)

(define-public (reclaim-votes (market principal) (id (optional uint)))
    (let
        (
            (market-id (unwrap! id err-unknown-proposal))
   (poll-data (unwrap! (map-get? resolution-polls {market: market, market-id: market-id})
            (votes (unwrap! (map-get? member-total-votes {market-id: market-id, v
        )
        (asserts! (get concluded poll-data) err-not-concluded)
        (map-delete member-total-votes {market-id: market-id, voter: tx-sender})
        (contract-call? .bme000-0-governance-token bmg-unlock votes tx-sender)
    )
)

;; --- Extension callback
(define-public (callback (sender principal) (memo (buff 34)))
    (ok true)
)


(define-private (find-max-category (votes (list 10 uint)))
 (fold find-max-iter votes {max-votes: u0, max-index: u0, current-index: u0})
)

(define-private (find-max-iter (current-votes uint) (acc (tuple (max-votes uint) (max-index u
 (let
  (
   (max-votes (get max-votes acc))  ;; Extract highest vote count so far
   (max-index (get max-index acc))  ;; Extract category index with highest votes
   (current-index (get current-index acc))  ;; Track current category index
  )
  (if (> current-votes max-votes)
          (tuple (max-votes current-votes) (max-index current-index) (current-index (+ current-in
          (tuple (max-votes max-votes) (max-index max-index) (current-index (+ current-index u1))
        )
      )
    )
```

mijoco-btc attempt to resolve clarinet toml issues around pyth. code is basicall...

767bb51 · 3 weeks ago

Executable File · 201 lines (176 loc) · 7.96 KB

Code Blame Raw

```
;; Title: BME021 Market Gating
;; Synopsis:
;; Efficient access control using merkle proofs.
;; Description:
;; Provides gating functionality based on account (can-access-by-account) and
;; ownership (can-access-by-ownership). The map of keys / roots are DAO managed.
;; Keys can by any data hash or a specific contract id hash. For ownership the user
;; must pass either an NFT or FT token and a merkle proof of ownership. For access
;; by account the account principal is passed along with the proof.

;; Define the SIP-009 and SIP-010 traits
(use-trait nft-trait 'SP2PABAF9FTAJYNFZH93XENAJ8FVY99RRM50D2JG9.nft-trait.nft-trait)
(use-trait ft-trait 'SP3FBR2AGK5H9QBDH3EEN6DF8EK8JY7RX8QJ5SVTE.sip-010-trait-ft-standard.sip-
(impl-trait 'SP3JP0N1ZXGASRJ0F7QAHWFPGTVK9T2XNXDB908Z.extension-trait.extension-trait)

(define-constant err-unauthorised (err u2200))
(define-constant err-either-sip9-or-sip10-required (err u2201))
(define-constant err-token-contract-invalid (err u2202))
(define-constant err-token-ownership-invalid (err u2203))
(define-constant err-expecting-nft-contract (err u2204))
(define-constant err-expecting-ft-contract (err u2205))
(define-constant err-expecting-token-id (err u2206))
(define-constant err-not-nft-owner (err u2207))
(define-constant err-not-ft-owner (err u2208))
(define-constant err-expecting-nft-buffer (err u2209))
(define-constant err-expecting-ft-buffer (err u2210))
(define-constant err-expecting-valid-merkle-proof (err u2211))
(define-constant err-expecting-merkle-root-for-poll (err u2212))
(define-constant err-expecting-an-owner (err u2213))
(define-constant err-account-proof-invalid (err u2214))
(define-constant err-ownership-proof-invalid (err u2215))

;; Merkle roots for gated data
(define-map merkle-roots
  (buff 32)
  {
              merkle-root: (buff 32)
  }
)

(define-public (is-dao-or-extension)
      (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
)

(define-public (set-merkle-root (hashed-id (buff 32)) (root (buff 32)))
  (begin
    ;; Ensure only dao can set the root
    (try! (is-dao-or-extension))
    (map-set merkle-roots hashed-id { merkle-root: root})
    (print {event: "merkle-root", hashed-id: hashed-id, merkle-root: (map-get? merkle-roots h
    (ok true)
  )
)

(define-public (set-merkle-root-by-principal (contract-id principal) (root (buff 32)))
  (let
      (
        ;; construct the key from the contract-id
        (principal-contract (unwrap! (principal-destruct? contract-id) (err u1001)))
        (contract-bytes (get hash-bytes principal-contract))
        ;; panics if not contract principal
        (contract-name (unwrap! (to-consensus-buff? (unwrap! (get name principal-contract) er
        (contract-key (sha256 (concat contract-bytes contract-name )))
    )
      ;; Ensure only dao can set the root
      (try! (is-dao-or-extension))
      (map-set merkle-roots contract-key { merkle-root: root})
      (print {event: "set-merkle-root-by-principal", contract-id: contract-id, contract-name:
      (ok true)
))

(define-read-only (get-merkle-root (hashed-id (buff 32)))
    (map-get? merkle-roots hashed-id)
)

;; Verify a Merkle proof
(define-private (calculate-hash (hash1 (buff 32)) (hash2 (buff 32)) (position bool))
  (if position
      (sha256 (concat hash2 hash1))
      (sha256 (concat hash1 hash2))
  )
)

 (define-private (process-proof-step (proof-step (tuple (position bool) (hash (buff 32)))) (c
  (let ((position (get position proof-step))
        (hash (get hash proof-step)))
    (calculate-hash current hash position)
  )
)

(define-private (verify-merkle-proof
    (leaf (buff 32))              ;; The leaf hash (token hash)
    (proof (list 10 (tuple (position bool) (hash (buff 32)))))
    (root (buff 32))
  )
  (let
    (
      (calculated-root
        (fold process-proof-step proof leaf)
      )
    )
    (ok (is-eq calculated-root root))
  )
)


(define-private (verify-nft-ownership
    (nft-contract <nft-trait>) ;; NFT contract
    (voter principal)          ;; Voter's principal
    (token-id uint)            ;; Token ID
```

```
    )
    (ok (>= balance quantity))
  ))


;; Validate proof of access
(define-public (can-access-by-ownership
    (market-data-hash (buff 32))                 ;; The hashed ID
    (nft-contract (optional <nft-trait>)) ;; Optional NFT contract
    (ft-contract (optional <ft-trait>))   ;; Optional FT contract
    (token-id (optional uint))         ;; Token ID for NFTs
    (proof (list 10 (tuple (position bool) (hash (buff 32)))))       ;; The Merkle proof
    (quantity uint)                    ;; Required token quantity
  )
  (let
    (
      ;; Determine if this is an NFT or FT contract
      (is-nft-contract (is-some nft-contract))

      ;; Fetch the Merkle root for the poll
      (root (unwrap! (map-get? merkle-roots market-data-hash) err-expecting-merkle-root-for

      ;; Compute the Merkle proof leaf
```

```
            (contract-id (if is-nft-contract
                             (unwrap! (to-consensus-buff? (as-contract (unwrap! nft-contract err-
                             (unwrap! (to-consensus-buff? (as-contract (unwrap! ft-contract err-e
            (leaf (sha256 contract-id))

            ;; Verify the Merkle proof
            (proof-valid (unwrap! (verify-merkle-proof leaf proof (get merkle-root root)) err-exp

            ;; Verify ownership or balance
            (ownership-valid
              (if is-nft-contract
                  (unwrap! (verify-nft-ownership (unwrap! nft-contract err-expecting-nft-contract
                  (unwrap! (verify-ft-balance (unwrap! ft-contract err-expecting-ft-contract) tx-
          )
        ;; Ensure both conditions are satisfied
        (asserts! proof-valid err-ownership-proof-invalid)
        (asserts! ownership-valid err-token-ownership-invalid)
        (ok true)
      ))

    (define-public (can-access-by-account
        (sender principal)
        (proof (list 10 (tuple (position bool) (hash (buff 32)))))       ;; The Merkle proof
      )
      (let
          (
            ;; Fetch the Merkle root for the poll
            (principal-contract (unwrap! (principal-destruct? tx-sender) (err u1001)))
            (contract-bytes (get hash-bytes principal-contract))
            (contract-name (unwrap! (to-consensus-buff? (unwrap! (get name principal-contract) er
            (contract-key (sha256 (concat contract-bytes contract-name )))
            (root (unwrap! (map-get? merkle-roots contract-key) err-expecting-merkle-root-for-pol

            ;; Compute the Merkle proof leaf
            (principal-data (unwrap! (principal-destruct? sender) (err u1001)))
            (leaf (sha256 (get hash-bytes principal-data)))

            ;; Verify the Merkle proof
            (proof-valid (unwrap! (verify-merkle-proof leaf proof (get merkle-root root)) err-exp
        )
        ;; Ensure both conditions are satisfied
        (asserts! proof-valid err-account-proof-invalid)
        (print {event: "can-access-by-account", contract-key: contract-key, contract-name: contra
        (ok true)
      ))


      ;; --- Extension callback
    (define-public (callback (sender principal) (memo (buff 34)))
            (ok true)
    )
```

BigMarketDao / bigmarket-dao Public

Code Issues Pull requests Actions Projects Security Insights

main bigmarket-dao / contracts / extensions / bme023-0-market-bitcoin.clar

Go to file

mijoco-btc Inclusion of reputation soul bound sft. 32188d4 · last week

653 lines (589 loc) · 24.6 KB

Code Blame Raw

```clarity
;; Title: BME023 Market bitcoin predictions
;; Synopsis:
;; Implements prediciton markets for bitcoin users (see also bme023-0-market-predicting).
;; Description:
;; Provide binary and categorical prediction markets with
;; bitcoin only transactions - no stx needed for gas. Works with
;; clarity-bitcoin-lib-v5 for bitcoin catamaran swaps into markets.

(impl-trait  .prediction-market-trait.prediction-market-trait)

(define-constant min-stake u100000) ;; Example: 100,000 satoshis (0.001 BTC)

;; ---------------- CONSTANTS & TYPES ----------------
(define-constant MARKET_TYPE u3) ;; bitcoin tx market
(define-constant token 'SM3VDXK3WZZSA84XXFKAFAF15NNZX32CTSG82JFQ4.sbtc-token)

(define-constant RESOLUTION_OPEN u0)
(define-constant RESOLUTION_RESOLVING u1)
(define-constant RESOLUTION_DISPUTED u2)
(define-constant RESOLUTION_RESOLVED u3)

(define-constant err-unauthorised (err u10000))
(define-constant err-invalid-market-type (err u10001))
(define-constant err-amount-too-low (err u10002))
(define-constant err-wrong-market-type (err u10003))
(define-constant err-already-concluded (err u10004))
(define-constant err-market-not-found (err u10005))
(define-constant err-user-not-winner-or-claimed (err u10006))
(define-constant err-user-not-staked (err u10008))
(define-constant err-market-not-concluded (err u10009))
(define-constant err-insufficient-balance (err u10011))
(define-constant err-insufficient-contract-balance (err u10012))
(define-constant err-user-share-is-zero (err u10013))
(define-constant err-dao-fee-bips-is-zero (err u10014))
(define-constant err-disputer-must-have-stake (err u10015))
(define-constant err-dispute-window-elapsed (err u10016))
(define-constant err-market-not-resolving (err u10017))
(define-constant err-market-not-open (err u10018))
(define-constant err-dispute-window-not-elapsed (err u10019))
```

```
(define-constant err-dispute-window-not-elapsed (err u10019))
(define-constant err-market-wrong-state (err u10020))
(define-constant err-invalid-token (err u10021))
(define-constant err-max-market-fee-bips-exceeded (err u10022))
(define-constant err-category-not-found (err u10023))
(define-constant err-too-few-categories (err u10024))
(define-constant err-element-expected (err u10025))
(define-constant err-winning-stake-not-zero (err u10026))
(define-constant err-losing-stake-is-zero (err u10027))
(define-constant err-transaction-segwit (err u10028))
(define-constant err-transaction-legacy (err u10029))
(define-constant err-transaction (err u1030))
(define-constant err-market-wallet (err u1031))
(define-constant err-transfer-forbidden (err u1032))

(define-data-var market-counter uint u0)
(define-data-var dispute-window-length uint u144)
(define-data-var dev-fee-bips uint u200)
(define-data-var dao-fee-bips uint u200)
(define-data-var market-fee-bips-max uint u1000)
(define-data-var market-create-fee uint u100000000)
(define-data-var dev-fund principal tx-sender)
(define-data-var resolution-agent principal tx-sender)
(define-data-var dao-treasury principal tx-sender)
(define-data-var creation-gated bool true)
(define-data-var market-wallet { version: (buff 1), hashbytes: (buff 32) } { version: 0x00, h
(define-data-var resolution-timeout uint u1000) ;; 1000 blocks (~9 days)

;; Data structure for each Market
;; outcome: winning category
(define-map markets
  uint
  {
                market-data-hash: (buff 32),
    treasury: principal,
    creator: principal,
    market-fee-bips: uint,
    resolution-state: uint, ;; "open", "resolving", "disputed", "concluded"
    resolution-burn-height: uint,
    categories: (list 10 (string-ascii 64)), ;; List of available categories
    stakes: (list 10 uint), ;; Total staked per category
    outcome: (optional uint),
    concluded: bool
  }
)

(define-map stake-balances
  { market-id: uint, user: principal }
  (list 10 uint)
)

;; ---------------- access control ----------------
(define-public (is-dao-or-extension)
        (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
)

;; ---------------- getters / setters ----------------
```

```

(define-public (set-dispute-window-length (length uint))
  (begin
    (try! (is-dao-or-extension))
    (var-set dispute-window-length length)
    (ok true)
  )
)

(define-public (set-creation-gated (gated bool))
  (begin
    (try! (is-dao-or-extension))
    (var-set creation-gated gated)
    (ok true)
  )
```

```
            (market-fee-bips (get market-fee-bips md))
            (user-share (if (> winning-pool u0) (/ (* user-stake total-pool) winning-pool) u0))
            (daofee (/ (* user-share (var-get dao-fee-bips)) u10000))
            (marketfee (/ (* user-share market-fee-bips) u10000))
            (user-share-net (- user-share (+ daofee marketfee)))
        )
        (begin
          ;; Ensure inputs are valid
          (asserts! (> winning-pool u0) err-amount-too-low)
          (asserts! (> user-share-net u0) err-user-share-is-zero)
          (asserts! (> daofee u0) err-dao-fee-bips-is-zero)

          ;; Perform transfers
          (as-contract
            (begin
              ;; Transfer user share, capped by initial contract balance

              ;;(try! (stx-transfer? user-share-net tx-sender original-sender))
              ;;(try! (stx-transfer? daofee tx-sender (var-get dao-treasury)))

              (if (> user-share-net u0)
                (try! (contract-call? token transfer user-share-net tx-sender original-sender nor
                true
              )
              (if (> daofee u0)
                (try! (contract-call? token transfer daofee tx-sender (var-get dao-treasury) none
                true
              )
              (if (> marketfee u0)
                (try! (contract-call? token transfer marketfee tx-sender treasury none))
                true
              )
            )
          )

          ;; Zero out user stake
          (map-set stake-balances { market-id: market-id, user: tx-sender } (list u0 u0 u0 u0 u0

          ;; Log and return user share
          (try! (contract-call? .bme030-0-reputation-token mint tx-sender u6 u2))
          (print {event: "claim-winnings", market-id: market-id, index-won: index-won, claimer: t
          (ok user-share-net)
        )
      )
    )

    ;; the funds have arrived on bitcoin - so the sender here is the big market sbtc liquidity pc
    (define-private (process-stake-transfer (amount uint))
      (let (
            ;;(sender-balance (stx-get-balance tx-sender))
            (sender-balance (unwrap! (contract-call? token get-balance .bme023-0-market-bitcoin)
            (fee (calculate-fee amount (var-get dev-fee-bips)))
            (transfer-amount (- amount fee))
           )
        (begin
          ;; Ensure amount is valid
          (asserts! (>= amount u100) err-amount-too-low)
          ;; Check tx-sender's balance
          (asserts! (>= sender-balance amount) err-insufficient-balance)

          ;; assume here the contract has the funds to cover payouts.
          ;; in fact the liquidity will come from direct sbtc into this contract from the bitcoir
          ;; (try! (contract-call? token transfer transfer-amount tx-sender .bme023-0-market-prec
          (try! (as-contract (contract-call? token transfer fee .bme023-0-market-bitcoin (var-get

          (ok transfer-amount)
        )
      )
    )
    (define-private (calculate-fee (amount uint) (fee-bips uint))
      (let ((fee (/ (* amount fee-bips) u10000)))
        fee
      )
    )
```

BigMarketDao / **bigmarket-dao** Public

Code · Issues · Pull requests · Actions · Projects · Security · Insights

main

bigmarket-dao / contracts / extensions / **bme023-0-market-predicting.clar**

Go to file

mijoco-btc Inclusion of reputation soul bound sft. 32188d4 · last week

495 lines (445 loc) · 19.5 KB

Code Blame Raw

```
;; Title: BME023 Market Predicting
;; Synopsis:
;; Implements binary and categorical prediciton markets.
;; Description:
;; Market creation allows a new binary or categorical market to be set up.
;; Off chain market data is verifiable via the markets data hash.
;; Markets run in a specific token (stx, sbtc, bmg etc) the market is created
;; with an allowed token. Allowed tokens are controlled by the DAO.
;; Market creation can be gated via market proof and a market creator can
;; set their own fee up to a max fee amount determined by the DAO.
;; Anyone with the required token can stake as many times as they wish and for any choice
;; of outcome. Resolution process begins via a call gated to the DAO controlled resolution ag
;; address. The resolution can be challenged by anyone with a stake in the market
;; If a challenge is made the dispute resolution process begins which requires a DAO vote
;; to resolve - the outcome of the vote resolve the market and sets the outcome.
;; If the dispute window passes without challenge or once the vote concludes the market is fu
;; resolved and claims can then be made.

(use-trait ft-token 'SP3FBR2AGK5H9QBDH3EEN6DF8EK8JY7RX8QJ5SVTE.sip-010-trait-ft-standard.sip-
(impl-trait  .prediction-market-trait.prediction-market-trait)

;; ---------------- CONSTANTS & TYPES ----------------
;; Market Types (1 => categorical market)
(define-constant MARKET_TYPE u1)

(define-constant RESOLUTION_OPEN u0)
(define-constant RESOLUTION_RESOLVING u1)
(define-constant RESOLUTION_DISPUTED u2)
(define-constant RESOLUTION_RESOLVED u3)

(define-constant err-unauthorised (err u10000))
(define-constant err-invalid-market-type (err u10001))
(define-constant err-amount-too-low (err u10002))
(define-constant err-wrong-market-type (err u10003))
(define-constant err-already-concluded (err u10004))
(define-constant err-market-not-found (err u10005))
(define-constant err-user-not-winner-or-claimed (err u10006))
(define-constant err-user-not-staked (err u10008))
(define-constant err-market-not-concluded (err u10009))
```

```
(define-constant err-insufficient-balance (err u10011))
(define-constant err-insufficient-contract-balance (err u10012))
(define-constant err-user-share-is-zero (err u10013))
(define-constant err-dao-fee-bips-is-zero (err u10014))
(define-constant err-disputer-must-have-stake (err u10015))
(define-constant err-dispute-window-elapsed (err u10016))
(define-constant err-market-not-resolving (err u10017))
(define-constant err-market-not-open (err u10018))
(define-constant err-dispute-window-not-elapsed (err u10019))
(define-constant err-market-wrong-state (err u10020))
(define-constant err-invalid-token (err u10021))
(define-constant err-max-market-fee-bips-exceeded (err u10022))
(define-constant err-category-not-found (err u10023))
(define-constant err-too-few-categories (err u10024))
(define-constant err-element-expected (err u10025))
(define-constant err-winning-stake-not-zero (err u10026))
(define-constant err-losing-stake-is-zero (err u10027))

(define-data-var market-counter uint u0)
(define-data-var dispute-window-length uint u144)
(define-data-var dev-fee-bips uint u200)
(define-data-var dao-fee-bips uint u200)
(define-data-var market-fee-bips-max uint u1000)
(define-data-var market-create-fee uint u100000000)
(define-data-var dev-fund principal tx-sender)
(define-data-var resolution-agent principal tx-sender)
(define-data-var dao-treasury principal tx-sender)
(define-data-var creation-gated bool true)
(define-data-var resolution-timeout uint u1000) ;; 1000 blocks (~9 days)

;; Data structure for each Market
;; outcome: winning category
(define-map markets
  uint
  {
        market-data-hash: (buff 32),
    token: principal,
    treasury: principal,
    creator: principal,
    market-fee-bips: uint,
    resolution-state: uint, ;; "open", "resolving", "disputed", "concluded"
    resolution-burn-height: uint,
    categories: (list 10 (string-ascii 64)), ;; List of available categories
    stakes: (list 10 uint), ;; Total staked per category
    outcome: (optional uint),
    concluded: bool
  }
)

(define-map stake-balances
  { market-id: uint, user: principal }
  (list 10 uint)
)
(define-map allowed-tokens principal bool)

;; ---------------- access control ----------------
```

```
(define-public (is-dao-or-extension)
	(ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
)

;; ---------------- getters / setters ----------------
(define-public (set-allowed-token (token principal) (enabled bool))
	(begin
		(try! (is-dao-or-extension))
		(print {event: "allowed-token", token: token, enabled: enabled})
		(ok (map-set allowed-tokens token enabled))
	)
)
(define-read-only (is-allowed-token (token principal))
	(default-to false (map-get? allowed-tokens token))
)
```

```
    (user-stake uint)
    (winning-pool uint)
    (total-pool uint)
    (index-won uint) (token <ft-token>))
    (let (
          (md (unwrap! (map-get? markets market-id) err-market-not-found))
          (original-sender tx-sender)
          (treasury (get treasury md))
          (market-fee-bips (get market-fee-bips md))
          (user-share (if (> winning-pool u0) (/ (* user-stake total-pool) winning-pool) u0))
          (daofee (/ (* user-share (var-get dao-fee-bips)) u10000))
    (marketfee (/ (* user-share market-fee-bips) u10000))
    (user-share-net (- user-share (+ daofee marketfee)))
  )
  (begin
   ;; Ensure inputs are valid
   (asserts! (> winning-pool u0) err-amount-too-low)
   (asserts! (> user-share-net u0) err-user-share-is-zero)

   ;; Perform transfers
   (as-contract
    (begin
     ;; Transfer user share, capped by initial contract balance
     (if (> user-share-net u0)
      (try! (contract-call? token transfer user-share-net tx-sender original-sender nor
      true
     )
     (if (> daofee u0)
      (try! (contract-call? token transfer daofee tx-sender (var-get dao-treasury) none
      true
     )
     (if (> marketfee u0)
      (try! (contract-call? token transfer marketfee tx-sender treasury none))
      true
     )
    )
   )

   ;; Zero out user stake
   (map-set stake-balances { market-id: market-id, user: tx-sender } (list u0 u0 u0 u0 u0

   ;; Log and return user share
   (try! (contract-call? .bme030-0-reputation-token mint tx-sender u6 u2))
   (print {event: "claim-winnings", market-id: market-id, index-won: index-won, claimer: t
   (ok user-share-net)
  )
 )
)

(define-private (process-stake-transfer (amount uint) (token <ft-token>))
 (let (
    ;;(sender-balance (stx-get-balance tx-sender))
    (sender-balance (unwrap! (contract-call? token get-balance tx-sender) err-insufficier
    (fee (calculate-fee amount (var-get dev-fee-bips)))
    (transfer-amount (- amount fee))
   )
  (begin
   ;; Ensure amount is valid
   (asserts! (>= amount u100) err-amount-too-low)
   ;; Check tx-sender's balance
   (asserts! (>= sender-balance amount) err-insufficient-balance)

   (try! (contract-call? token transfer transfer-amount tx-sender .bme023-0-market-predict
   (try! (contract-call? token transfer fee tx-sender (var-get dev-fund) none))

   (ok transfer-amount)
  )
  )
)
(define-private (calculate-fee (amount uint) (fee-bips uint))
  (let ((fee (/ (* amount fee-bips) u10000)))
    fee
  )
)
```

BigMarketDao / **bigmarket-dao** Public

Code · Issues · Pull requests · Actions · Projects · Security · Insights

main

bigmarket-dao / contracts / extensions / **bme023-0-market-scalar-dia.clar**

Go to file

mijoco-btc Inclusion of reputation soul bound sft. 32188d4 · last week

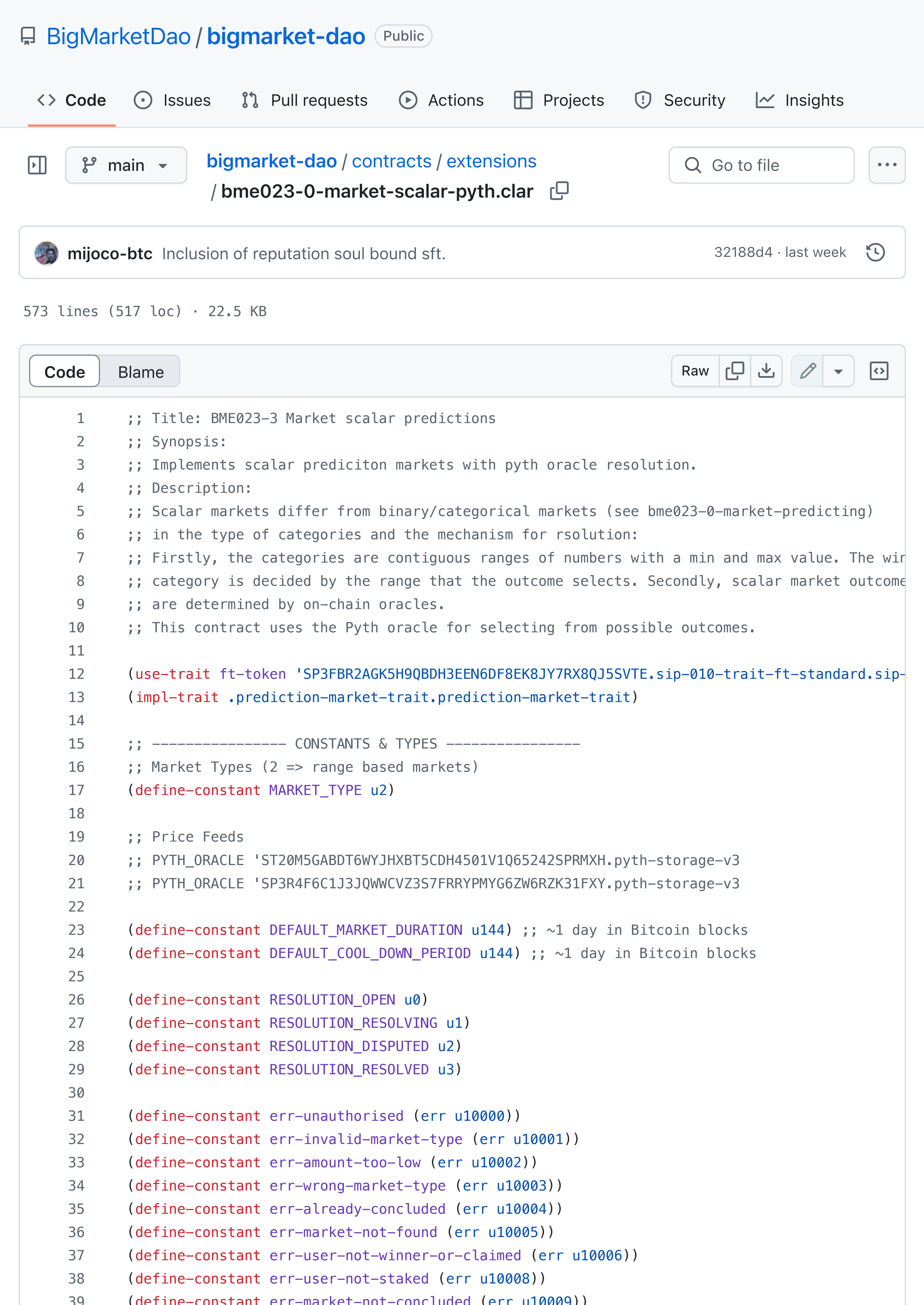
594 lines (536 loc) · 23.4 KB

Code Blame Raw

```
;; Title: BME023 Market scalar predictions
;; Synopsis:
;; Implements scalar prediciton markets (see also bme023-0-market-predicting).
;; Description:
;; Scalar markets differ from binary/categorical markets (see bme023-0-market-predicting)
;; in the type of categories and the mechanism for rsolution:
;; Firstly, the categories are contiguous ranges of numbers with a min and max value. The wir
;; category is decided by the range that the outcome selects. Secondly, scalar market outcome
;; are determined by on-chain oracles. This contract uses the DIA oracle for selecting from
;; possible outcomes.

(use-trait ft-token 'SP3FBR2AGK5H9QBDH3EEN6DF8EK8JY7RX8QJ5SVTE.sip-010-trait-ft-standard.sip-
(impl-trait .prediction-market-trait.prediction-market-trait)

;; ---------------- CONSTANTS & TYPES ----------------
;; Market Types (2 => range based markets)
(define-constant MARKET_TYPE u2)

;; Price Feeds
;; DIA_ORACLE 'SP1G48FZ4Y7JY8G2Z0N51QTCYGBQ6F4J43J77BQC0.dia-oracle
;; DIA_ORACLE 'ST3Q982CNNQ00E3FH6853EMTA5FPF1M3ENJTHB8PY.dia-oracle
(define-constant STX_USD_FEED_ID 0xe62df6c8b4a85fe1a67db44dc12de5db330f7ac66b72dc658afedf0f4a

(define-constant DEFAULT_MARKET_DURATION u144) ;; ~1 day in Bitcoin blocks
(define-constant DEFAULT_COOL_DOWN_PERIOD u144) ;; ~1 day in Bitcoin blocks

(define-constant RESOLUTION_OPEN u0)
(define-constant RESOLUTION_RESOLVING u1)
(define-constant RESOLUTION_DISPUTED u2)
(define-constant RESOLUTION_RESOLVED u3)

(define-constant err-unauthorised (err u10000))
(define-constant err-invalid-market-type (err u10001))
(define-constant err-amount-too-low (err u10002))
(define-constant err-wrong-market-type (err u10003))
(define-constant err-already-concluded (err u10004))
(define-constant err-market-not-found (err u10005))
(define-constant err-user-not-winner-or-claimed (err u10006))
(define-constant err-user-not-staked (err u10008))
```

```
(define-constant err-market-not-concluded (err u10009))
(define-constant err-insufficient-balance (err u10011))
(define-constant err-insufficient-contract-balance (err u10012))
(define-constant err-user-share-is-zero (err u10013))
(define-constant err-dao-fee-bips-is-zero (err u10014))
(define-constant err-disputer-must-have-stake (err u10015))
(define-constant err-dispute-window-elapsed (err u10016))
(define-constant err-market-not-resolving (err u10017))
(define-constant err-market-not-open (err u10018))
(define-constant err-dispute-window-not-elapsed (err u10019))
(define-constant err-market-wrong-state (err u10020))
(define-constant err-invalid-token (err u10021))
(define-constant err-max-market-fee-bips-exceeded (err u10022))
(define-constant err-category-not-found (err u10023))
(define-constant err-too-few-categories (err u10024))
(define-constant err-element-expected (err u10025))
(define-constant err-winning-stake-not-zero (err u10026))
(define-constant err-losing-stake-is-zero (err u10027))
(define-constant err-unknown-stacks-block (err u10028))

(define-data-var market-counter uint u0)
(define-data-var dispute-window-length uint u144)
(define-data-var dev-fee-bips uint u200)
(define-data-var dao-fee-bips uint u200)
(define-data-var market-fee-bips-max uint u1000)
(define-data-var market-create-fee uint u100000000)
(define-data-var dev-fund principal tx-sender)
(define-data-var resolution-agent principal tx-sender)
(define-data-var dao-treasury principal tx-sender)
(define-data-var creation-gated bool true)
(define-data-var resolution-timeout uint u1000) ;; 1000 blocks (~9 days)

;; Data structure for each Market
;; outcome: winning category
(define-map markets
  uint
  {
        market-data-hash: (buff 32),
    token: principal,
    treasury: principal,
    creator: principal,
    market-fee-bips: uint,
    resolution-state: uint, ;; "open", "resolving", "disputed", "concluded"
    categories: (list 10 {min: uint, max: uint}), ;; Min (inclusive) and Max (exclusive)
    stakes: (list 10 uint), ;; Total staked per category
    outcome: (optional uint),
    concluded: bool,
    market-start: uint,
    market-duration: uint,
    cool-down-period: uint,
    price-feed-id: (string-ascii 32), ;; DIA price feed ID (custom per market)
    price-outcome: (optional uint)
  }
)

(define-map stake-balances
  { market-id: uint, user: principal }
  (list 10 uint)
)
(define-map allowed-tokens principal bool)

;; ---------------- access control ----------------
(define-public (is-dao-or-extension)
        (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
)

;; ---------------- getters / setters ----------------
(define-public (set-allowed-token (token principal) (enabled bool))
        (begin
                (try! (is-dao-or-extension))
                (print {event: "allowed-token", token: token, enabled: enabled})
```

```
        (md (unwrap! (map-get? markets market-id) err-market-not-found))
        (original-sender tx-sender)
        (treasury (get treasury md))
        (market-fee-bips (get market-fee-bips md))
        (user-share (if (> winning-pool u0) (/ (* user-stake total-pool) winning-pool) u0))
        (daofee (/ (* user-share (var-get dao-fee-bips)) u10000))
        (marketfee (/ (* user-share market-fee-bips) u10000))
        (user-share-net (- user-share (+ daofee marketfee)))
    )
    (begin
      ;; Ensure inputs are valid
      (asserts! (> winning-pool u0) err-amount-too-low)
      (asserts! (> user-share-net u0) err-user-share-is-zero)
      (asserts! (> daofee u0) err-dao-fee-bips-is-zero)

      ;; Perform transfers
      (as-contract
        (begin
          ;; Transfer user share, capped by initial contract balance

          ;;(try! (stx-transfer? user-share-net tx-sender original-sender))
          ;;(try! (stx-transfer? daofee tx-sender (var-get dao-treasury)))

          (if (> user-share-net u0)
            (try! (contract-call? token transfer user-share-net tx-sender original-sender nor
            true
          )
          (if (> daofee u0)
            (try! (contract-call? token transfer daofee tx-sender (var-get dao-treasury) none
            true
          )
          (if (> marketfee u0)
            (try! (contract-call? token transfer marketfee tx-sender treasury none))
            true
          )
        )
      )

      ;; Zero out user stake
      (map-set stake-balances { market-id: market-id, user: tx-sender } (list u0 u0 u0 u0 u0

      ;; Log and return user share
      (try! (contract-call? .bme030-0-reputation-token mint tx-sender u6 u2))
      (print {event: "claim-winnings", market-id: market-id, index-won: index-won, claimer: t
      (ok user-share-net)
    )
  )
)

(define-private (process-stake-transfer (amount uint) (token <ft-token>))
  (let (
        ;;(sender-balance (stx-get-balance tx-sender))
        (sender-balance (unwrap! (contract-call? token get-balance tx-sender) err-insufficier
        (fee (calculate-fee amount (var-get dev-fee-bips)))
        (transfer-amount (- amount fee))
       )
    (begin
      ;; Ensure amount is valid
      (asserts! (>= amount u100) err-amount-too-low)
      ;; Check tx-sender's balance
      (asserts! (>= sender-balance amount) err-insufficient-balance)

      (try! (contract-call? token transfer transfer-amount tx-sender .bme023-0-market-scalar-
      (try! (contract-call? token transfer fee tx-sender (var-get dev-fund) none))

      (ok transfer-amount)
    )
  )
)
(define-private (calculate-fee (amount uint) (fee-bips uint))
  (let ((fee (/ (* amount fee-bips) u10000)))
    fee
  )
)
```


BigMarketDao / bigmarket-dao Public
Code Issues Pull requests Actions Projects Security Insights
main
Go to file

bigmarket-dao / contracts / extensions / bme023-0-market-scalar-pyth.clar

mijoco-btc Inclusion of reputation soul bound sft. 32188d4 · last week

573 lines (517 loc) · 22.5 KB

```
;; Title: BME023-3 Market scalar predictions
;; Synopsis:
;; Implements scalar prediciton markets with pyth oracle resolution.
;; Description:
;; Scalar markets differ from binary/categorical markets (see bme023-0-market-predicting)
;; in the type of categories and the mechanism for rsolution:
;; Firstly, the categories are contiguous ranges of numbers with a min and max value. The wir
;; category is decided by the range that the outcome selects. Secondly, scalar market outcome
;; are determined by on-chain oracles.
;; This contract uses the Pyth oracle for selecting from possible outcomes.

(use-trait ft-token 'SP3FBR2AGK5H9QBDH3EEN6DF8EK8JY7RX8QJ5SVTE.sip-010-trait-ft-standard.sip-
(impl-trait .prediction-market-trait.prediction-market-trait)

;; ---------------- CONSTANTS & TYPES ----------------
;; Market Types (2 => range based markets)
(define-constant MARKET_TYPE u2)

;; Price Feeds
;; PYTH_ORACLE 'ST20M5GABDT6WYJHXBT5CDH4501V1Q65242SPRMXH.pyth-storage-v3
;; PYTH_ORACLE 'SP3R4F6C1J3JQWWCVZ3S7FRRYPMYG6ZW6RZK31FXY.pyth-storage-v3

(define-constant DEFAULT_MARKET_DURATION u144) ;; ~1 day in Bitcoin blocks
(define-constant DEFAULT_COOL_DOWN_PERIOD u144) ;; ~1 day in Bitcoin blocks

(define-constant RESOLUTION_OPEN u0)
(define-constant RESOLUTION_RESOLVING u1)
(define-constant RESOLUTION_DISPUTED u2)
(define-constant RESOLUTION_RESOLVED u3)

(define-constant err-unauthorised (err u10000))
(define-constant err-invalid-market-type (err u10001))
(define-constant err-amount-too-low (err u10002))
(define-constant err-wrong-market-type (err u10003))
(define-constant err-already-concluded (err u10004))
(define-constant err-market-not-found (err u10005))
(define-constant err-user-not-winner-or-claimed (err u10006))
(define-constant err-user-not-staked (err u10008))
(define-constant err-market-not-concluded (err u10009))
```

```
(define-constant err-insufficient-balance (err u10011))
(define-constant err-insufficient-contract-balance (err u10012))
(define-constant err-user-share-is-zero (err u10013))
(define-constant err-dao-fee-bips-is-zero (err u10014))
(define-constant err-disputer-must-have-stake (err u10015))
(define-constant err-dispute-window-elapsed (err u10016))
(define-constant err-market-not-resolving (err u10017))
(define-constant err-market-not-open (err u10018))
(define-constant err-dispute-window-not-elapsed (err u10019))
(define-constant err-market-wrong-state (err u10020))
(define-constant err-invalid-token (err u10021))
(define-constant err-max-market-fee-bips-exceeded (err u10022))
(define-constant err-category-not-found (err u10023))
(define-constant err-too-few-categories (err u10024))
(define-constant err-element-expected (err u10025))
(define-constant err-winning-stake-not-zero (err u10026))
(define-constant err-losing-stake-is-zero (err u10027))
(define-constant err-unknown-stacks-block (err u10028))

(define-data-var market-counter uint u0)
(define-data-var dispute-window-length uint u144)
(define-data-var dev-fee-bips uint u200)
(define-data-var dao-fee-bips uint u200)
(define-data-var market-fee-bips-max uint u1000)
(define-data-var market-create-fee uint u100000000)
(define-data-var dev-fund principal tx-sender)
(define-data-var resolution-agent principal tx-sender)
(define-data-var dao-treasury principal tx-sender)
(define-data-var creation-gated bool true)
(define-data-var resolution-timeout uint u1000) ;; 1000 blocks (~9 days)

;; Data structure for each Market
;; outcome: winning category
(define-map markets
  uint
  {
                market-data-hash: (buff 32),
    token: principal,
    treasury: principal,
    creator: principal,
    market-fee-bips: uint,
    resolution-state: uint, ;; "open", "resolving", "disputed", "concluded"
    categories: (list 10 {min: uint, max: uint}), ;; Min (inclusive) and Max (exclusive)
    stakes: (list 10 uint), ;; Total staked per category
    outcome: (optional uint),
    concluded: bool,
    market-start: uint,
    market-duration: uint,
    cool-down-period: uint,
    price-feed-id: (buff 32), ;; Pyth price feed ID
    price-outcome: (optional uint)
  }
)

(define-map stake-balances
  { market-id: uint, user: principal }
  (list 10 uint)
)
(define-map allowed-tokens principal bool)

;; ---------------- access control ----------------
(define-public (is-dao-or-extension)
        (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
)

;; ---------------- getters / setters ----------------
(define-public (set-allowed-token (token principal) (enabled bool))
        (begin
                (try! (is-dao-or-extension))
                (print {event: "allowed-token", token: token, enabled: enabled})
                (ok (map-set allowed-tokens token enabled))
```

```
      (md (unwrap! (map-get? markets market-id) err-market-not-found))
      (original-sender tx-sender)
      (treasury (get treasury md))
      (market-fee-bips (get market-fee-bips md))
      (user-share (if (> winning-pool u0) (/ (* user-stake total-pool) winning-pool) u0))
      (daofee (/ (* user-share (var-get dao-fee-bips)) u10000))
      (marketfee (/ (* user-share market-fee-bips) u10000))
      (user-share-net (- user-share (+ daofee marketfee)))
    )
    (begin
     ;; Ensure inputs are valid
     (asserts! (> winning-pool u0) err-amount-too-low)
     (asserts! (> user-share-net u0) err-user-share-is-zero)
     (asserts! (> daofee u0) err-dao-fee-bips-is-zero)

     ;; Perform transfers
     (as-contract
      (begin
       ;; Transfer user share, capped by initial contract balance

       ;;(try! (stx-transfer? user-share-net tx-sender original-sender))
       ;;(try! (stx-transfer? daofee tx-sender (var-get dao-treasury)))

       (if (> user-share-net u0)
        (try! (contract-call? token transfer user-share-net tx-sender original-sender nor
        true
       )
       (if (> daofee u0)
        (try! (contract-call? token transfer daofee tx-sender (var-get dao-treasury) none
        true
       )
       (if (> marketfee u0)
        (try! (contract-call? token transfer marketfee tx-sender treasury none))
        true
       )
      )
     )

     ;; Zero out user stake
     (map-set stake-balances { market-id: market-id, user: tx-sender } (list u0 u0 u0 u0 u0

     ;; Log and return user share
     (try! (contract-call? .bme030-0-reputation-token mint tx-sender u6 u2))
     (print {event: "claim-winnings", market-id: market-id, index-won: index-won, claimer: t
     (ok user-share-net)
```

```
        )
      )
    )

    (define-private (process-stake-transfer (amount uint) (token <ft-token>))
      (let (
            ;;(sender-balance (stx-get-balance tx-sender))
            (sender-balance (unwrap! (contract-call? token get-balance tx-sender) err-insufficier
            (fee (calculate-fee amount (var-get dev-fee-bips)))
            (transfer-amount (- amount fee))
           )
        (begin
          ;; Ensure amount is valid
          (asserts! (>= amount u100) err-amount-too-low)
          ;; Check tx-sender's balance
          (asserts! (>= sender-balance amount) err-insufficient-balance)

          (try! (contract-call? token transfer transfer-amount tx-sender .bme023-0-market-scalar-
          (try! (contract-call? token transfer fee tx-sender (var-get dev-fund) none))

          (ok transfer-amount)
        )
      )
    )
    (define-private (calculate-fee (amount uint) (fee-bips uint))
      (let ((fee (/ (* amount fee-bips) u10000)))
        fee
      )
    )
```

BigMarketDao / **bigmarket-dao** Public

Code Issues Pull requests Actions Projects Security Insights

main bigmarket-dao / contracts / extensions / **bme030-0-reputation-token.clar**

Go to file

mijoco-btc Release 2. Changes include new soul bound sft based reputation system

9f68c60 · 5 days ago

293 lines (253 loc) · 10.4 KB

Code Blame Raw

```
;; Title: BME030 Reputation Token
;; Synopsis:
;; Wraps reputation scheme within a non-transferable soulbound semi fungible token (see sip-0
;; Description:
;; The reputation token is a SIP-013 compliant token that is controlled by active DAO extensi
;; It facilitates hierarchical reputation and rewards based on engagements across a number of
;; BigMarket DAO features and use cases.

(impl-trait 'SPDBEG5X8XD50SPM1JJH0E5CTXGDV5NJTKAKKR5V.sip013-semi-fungible-token-trait.sip013
(impl-trait 'SPDBEG5X8XD50SPM1JJH0E5CTXGDV5NJTKAKKR5V.sip013-transfer-many-trait.sip013-trans

(define-constant err-unauthorised (err u30001))
(define-constant err-already-minted (err u30002))
(define-constant err-soulbound (err u30003))
(define-constant err-insufficient-balance (err u30004))
(define-constant err-zero-amount (err u30005))
(define-constant err-claims-old-epoch (err u30006))
(define-constant err-claims-zero-rep (err u30007))
(define-constant err-claims-zero-total (err u30008))
(define-constant err-invalid-tier (err u30009))

(define-constant max-tier u20)

(define-data-var token-name (string-ascii 32) "BigMarket Reputation Token")
(define-data-var token-symbol (string-ascii 10) "BIGR")

(define-fungible-token bigr-token)
(define-non-fungible-token bigr-id { token-id: uint, owner: principal })

(define-map balances { token-id: uint, owner: principal } uint)
(define-map supplies uint uint)
(define-map last-claimed-epoch { who: principal } uint)
(define-map tier-weights uint uint)

(define-data-var reward-per-epoch uint u100000000) ;; 1000 BIG (in micro units)
(define-data-var overall-supply uint u0)

```

```
;; ------------------------
;; DAO Control Check
;; ------------------------
(define-public (is-dao-or-extension)
        (ok (asserts! (or (is-eq tx-sender .bigmarket-dao) (contract-call? .bigmarket-dao is-
)

;; ------------------------
;; Trait Implementations
;; ------------------------
(define-read-only (get-balance (token-id uint) (who principal))
  (ok (default-to u0 (map-get? balances { token-id: token-id, owner: who })))
)

(define-read-only (get-symbol)
        (ok (var-get token-symbol))
)

(define-read-only (get-name)
        (ok (var-get token-name))
)

(define-read-only (get-overall-balance (who principal))
  (ok (ft-get-balance bigr-token who))
)

(define-read-only (get-total-supply (token-id uint))
  (ok (default-to u0 (map-get? supplies token-id)))
)

(define-read-only (get-overall-supply)
  (ok (var-get overall-supply))
)

(define-read-only (get-decimals (token-id uint)) (ok u0))

(define-read-only (get-token-uri (token-id uint))
  (ok none)
)

(define-public (set-reward-per-epoch (new-reward uint))
  (begin
    (try! (is-dao-or-extension))
    (var-set reward-per-epoch new-reward)
    (ok true)
  )
)
(define-public (set-tier-weight (token-id uint) (weight uint))
  (begin
    (try! (is-dao-or-extension))
    (map-set tier-weights token-id weight)
    (ok true)
  )
)

;; ------------------------
;; Mint / Burn
;; ------------------------
(define-public (mint (recipient principal) (token-id uint) (amount uint))
  (begin
    (try! (is-dao-or-extension))
    (asserts! (> amount u0) err-zero-amount)
    (asserts! (and (> token-id u0) (<= token-id max-tier)) err-invalid-tier)
    (try! (ft-mint? bigr-token amount recipient))
    (try! (tag-nft { token-id: token-id, owner: recipient }))
    (map-set balances { token-id: token-id, owner: recipient }
      (+ amount (default-to u0 (map-get? balances { token-id: token-id, owner: recipient }))))
    (map-set supplies token-id (+ amount (default-to u0 (map-get? supplies token-id))))
    (var-set overall-supply (+ (var-get overall-supply) amount))
    (print { event: "sft_mint", token-id: token-id, amount: amount, recipient: recipient })
    (ok true)
  )
)
```

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

  (define-read-only (get-epoch)
          (/ burn-block-height u4000)
  )

  (define-read-only (get-last-claimed-epoch)
          (default-to u0 (map-get? last-claimed-epoch { who: tx-sender }))
  )

  ;; ------------------------
  ;; Helpers
  ;; ------------------------
  (define-private (tag-nft (nft-token-id { token-id: uint, owner: principal }))
    (begin
      (if (is-some (nft-get-owner? bigr-id nft-token-id))
        (try! (nft-burn? bigr-id nft-token-id (get owner nft-token-id)))
        true)
      (nft-mint? bigr-id nft-token-id (get owner nft-token-id))
    )
  )

  (define-private (transfer-many-iter (item { token-id: uint, amount: uint, sender: principal,
    (match prev ok-prev (transfer (get token-id item) (get amount item) (get sender item) (get
  )

  (define-private (transfer-many-memo-iter (item { token-id: uint, amount: uint, sender: princi
    (match prev ok-prev (transfer-memo (get token-id item) (get amount item) (get sender item)
  )

  ;; dynamic weighted totals for user
  (define-read-only (get-weighted-rep (user principal))
    (let (
          (tiers (list u1 u2 u3 u4 u5 u6 u7 u8 u9 u10))
          (result (fold add-weighted-rep-for-user tiers (tuple (acc u0) (user user))))
    )
      (ok (get acc result))
    )
```

```
)

(define-private (add-weighted-rep-for-user (token-id uint) (state (tuple (acc uint) (user pri
  (let (
    (acc (get acc state))
    (user (get user state))
    (bal-at-tier (default-to u0 (map-get? balances {token-id: token-id, owner: user})))
    (weight-at-tier (default-to u1 (map-get? tier-weights token-id)))
  )
    (tuple (acc (+ acc (* bal-at-tier weight-at-tier))) (user user))
  )
)

;; dynamic weighted totals for overall supply pool
(define-read-only (get-weighted-supply)
  (let (
    (tiers (list u1 u2 u3 u4 u5 u6 u7 u8 u9 u10))
    (result (fold add-weighted-supply-for-tier tiers u0))
  )
    (ok result)
  )
)

(define-private (add-weighted-supply-for-tier (token-id uint) (acc uint))
  (let (
    (tier-supply (default-to u0 (map-get? supplies token-id)))
    (weight (default-to u1 (map-get? tier-weights token-id)))
  )
    (+ acc (* tier-supply weight))
  )
)
```