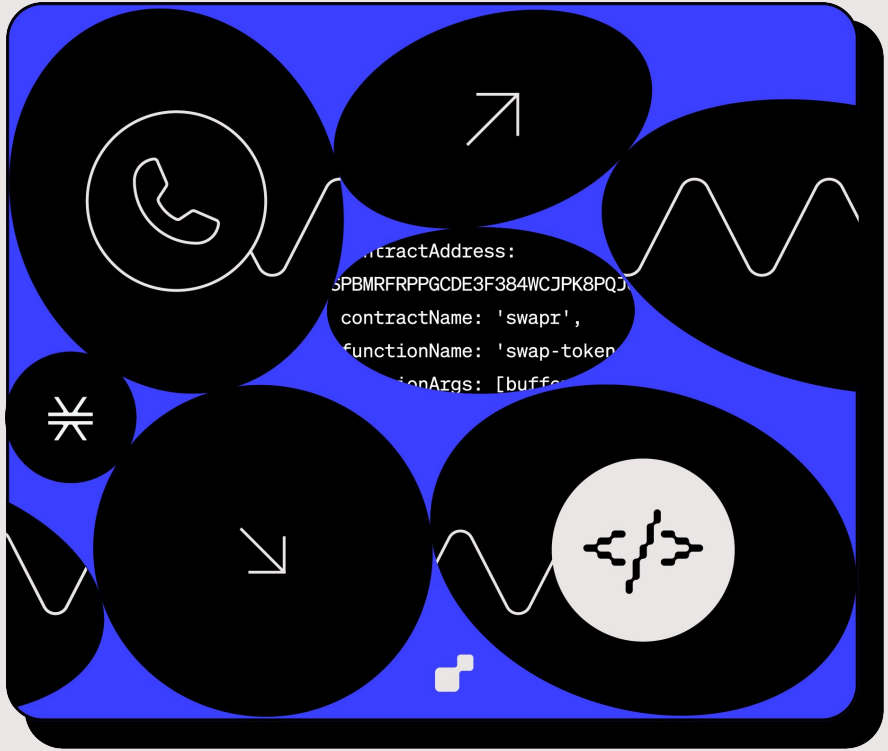
Hiro Monthly Developer Calls

#5 – August 25, 2022



Hiro Developers

Purpose

Hiro is
developer obsessed

Strive to
engage
with the developer community directly

Invite and
inspire
developers of diverse backgrounds to Stacks & Hiro

Listen
to your needs carefully & improve the DevX

Shoutouts & Thanks

August 2022 Contributors

Thank you!

Clarinet

@csgui
@dabuchera
@moodmosaic
@LNow

API

@wileyj
@dngrhm

Web Wallet

@aulneau
@sjc5

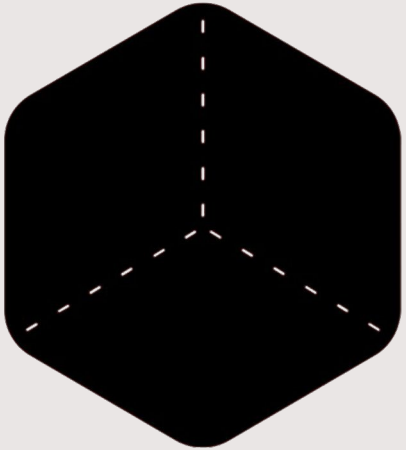
Community Topics

Community Topics

Topic 1: MultiSafe Demo w/ Chris Castig and Talha Bugra

PROBLEM: In order for Stacks and Bitcoin to grow, we need a scalable and secure way to store BTC, STX, SIP-009s (NFTs), SIP-010s (fungible tokens).

- MultiSafe is an open source, shared crypto safe for Stacks (STX).
  - "Gnosis-Safe" for Bitcoin DAOs.
- Audit completed with CoinFabrik.
- Can be leveraged by crypto communities and DAOs for treasury management.



https://linktr.ee/multisafe

https://github.com/Trust-Machines/multisafe

Community Topics

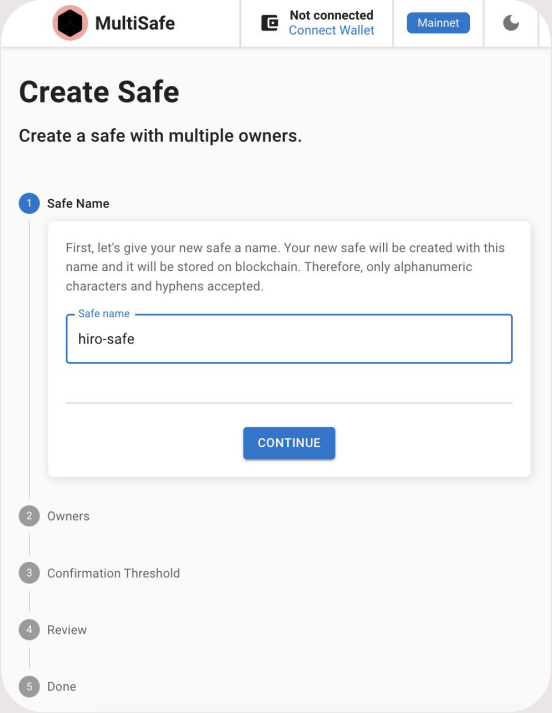
Topic 1: MultiSafe Demo w/ Chris Castig and Talha Bugra

Features

- Store, send, and receive Stacks (STX)
- Support for up to 20 different owners of the safe
- Configure confirmation threshold for executing transactions

Roadmap

- User-friendly app
- SIP-009 Non-fungible token and SIP-010 Fungible token support
- Native Bitcoin (BTC) support

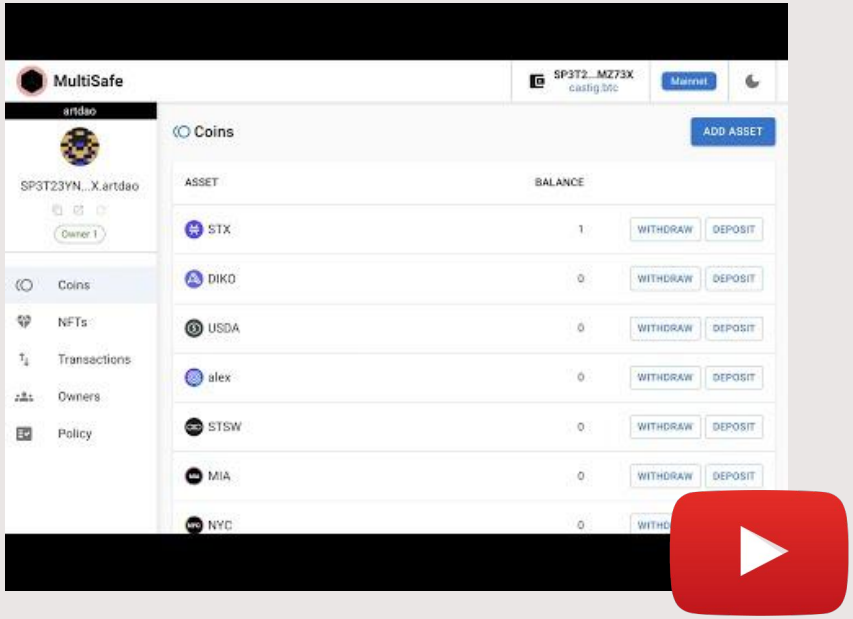


https://linktr.ee/multisafe

https://github.com/Trust-Machines/multisafe

Community Topics

Topic 1: MultiSafe Demo w/ Chris Castig and Talha Bugra



How to Use MultiSafe // A MultiSig for Stacks and Bitcoin DAOs

https://youtu.be/6Eash60pLJ0

https://linktr.ee/multisafe

https://github.com/Trust-Machines/multisafe

Community Topics

Topic 2: Docs Talk with Kenny Rogers

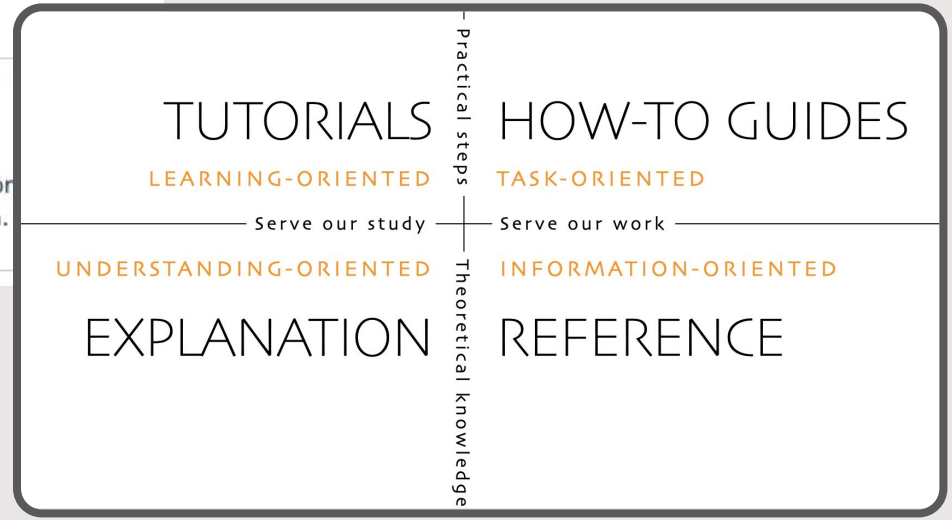
KennyRogers.btc
@KenTheRogers

Learning to build on Stacks should be
easy and straightforward

https://twitter.com/KenTheRogers/status/1551965346633895936?s=20&t=QUtpC30Zmy7hZvpPuu0Eug

Community Topics

Topic 2: Docs Talk with Kenny Rogers



⇒ Proposed Stacks Documentation Structure

Smart Contract of the Month

Topic 3: Smart Contract of the Month

Ryder Community Handles

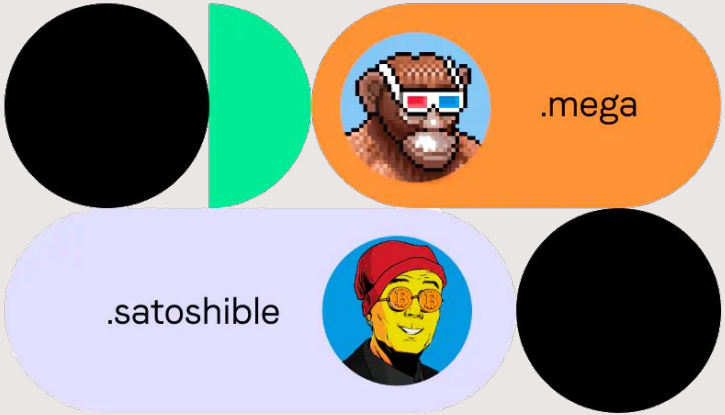
.mega
.crashpunk
.satoshible
.stacksparrot
.citycoins
.fren
.bitcoinmonkey

- Leveraging the culture and community in crypto to establish handles as identity within the Stacks ecosystem and beyond.

- Your web3 “handle”
  - Sign in to applications and with crypto
  - Send and receive transactions
  - Add some of your own personality to your crypto!

- Handles are registered through a smart contract on Stacks.

- A portion of the handle sales goes back to the crypto communities



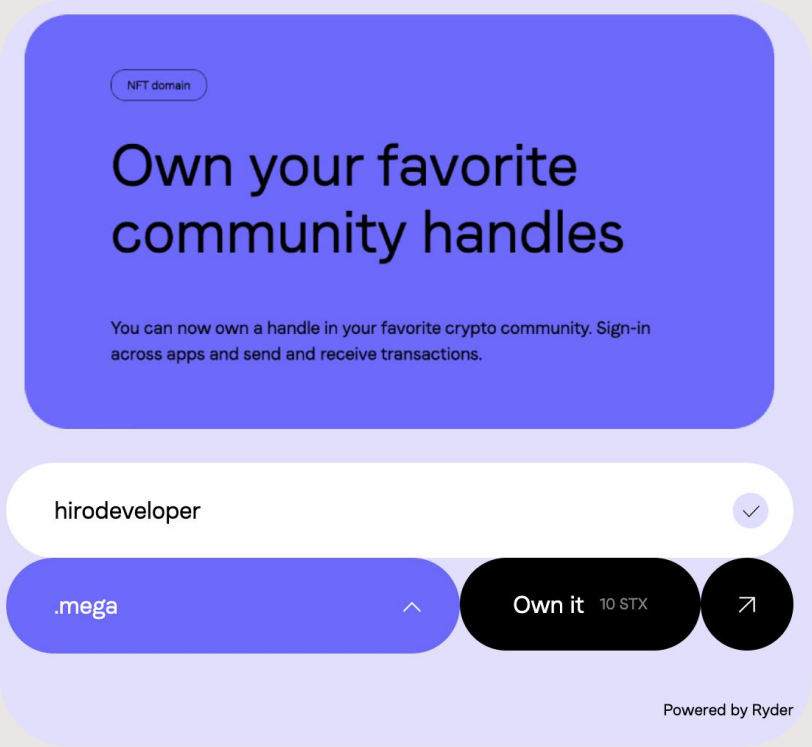
⇒ Link to Smart Contract in Stacks Explorer

https://handles.ryder.id/

Topic 3: Smart Contract of the Month

Ryder Community Handles

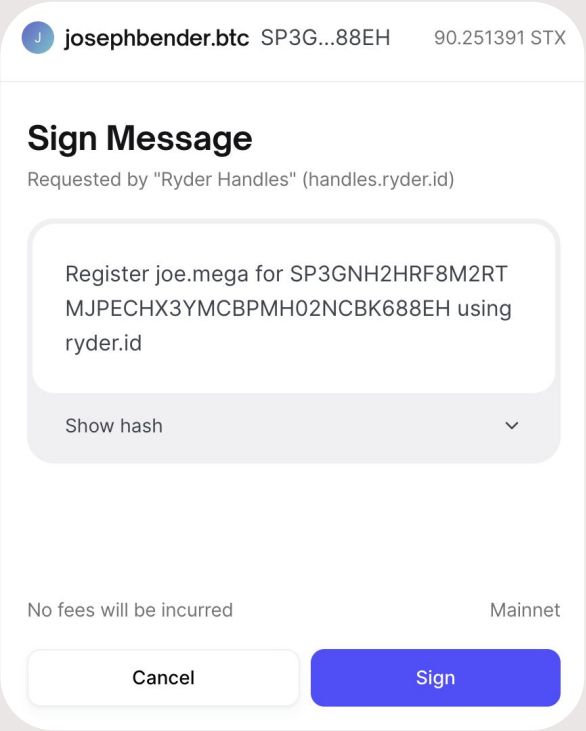
- Each handle costs ~$5 USD (in STX), plus transaction fees. It needs to be repurchased **once a year** to stay valid.
- Users can only own **one handle per account.**
- Ryder handles are community-centric in nature. It provides a protocol which allows communities to implement rules they should adhere to.
- A handle can range from **3-32 characters.**
- Currently, Ryder handles only support **Stacks (STX)** and **Bitcoin (BTC)**, but will support more cryptocurrencies in the future.
- Handles are also **NFTs** that can be sold on any marketplace.
- The protocol for handles is based on Stacks' **Blockchain Name System (BNS)**



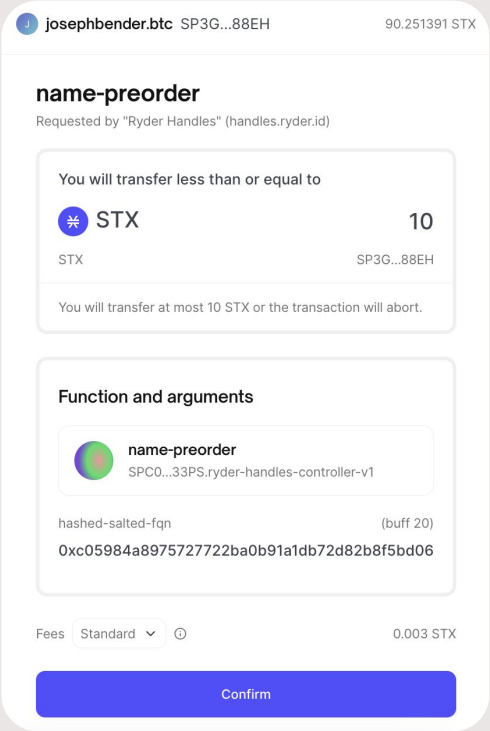
Topic 3: Smart Contract of the Month

ryder-handles-controller-v1

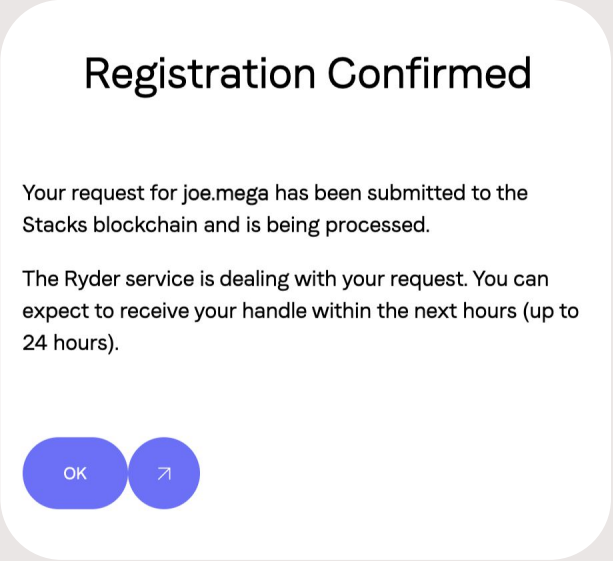
- 13 functions
- 14 variables
- 3 maps
- 0 tokens



1. Sign Message for Ryder to authenticate your wallet



2. Publish pre-order of the name to the Stacks blockchain



3. Registration complete

Community Topics

Defining Variables

Clarity smart contract on the Stacks blockchain responsible for handling name registrations

contract-owner: Sets the owner of the contract to the principal that deployed the Clarity file.

community-treasuries map: Creates a map that holds the principals of community treasuries.

approval-pubkey: Configures an empty buffer to hold a public key for approving operations.

price-in-ustx: Sets a default price for registering names

name-preorder-claimability-ttl: Variable for ensuring there in <24 hours between preorder and registration

name-preorders map: map that holds the salted hash, buyers principal, preorder creation time, claimed status, and price.

renewal-signatures map: map that holds users' renewal signatures.

```
(define-data-var contract-owner principal tx-sender)
(define-map community-treasuries (buff 20) principal)

(define-data-var approval-pubkey (buff 33) 0x00)
(define-data-var price-in-ustx uint u9999999)

;; between name-preorder and name-register must be less
(define-constant name-preorder-claimability-ttl u144)

(define-map name-preorders
  { hashed-salted-fqn: (buff 20), buyer: principal }
  { created-at: uint, claimed: bool, price: uint })

(define-map renewal-signatures (buff 65) bool)
```

name-preorder Function

User preorders desired name by registering a hash of the salted name

- Passes in a salted hash (**hashed-salted-fqn**) of buffer length 20 bytes for the name
- Pulls price from **price-in-ustx** variable
- Checks the **name-preorders** map to ensure former preorders have expired
  - Throws error if preorder exists
- Data validation on the salted hash
  - Verifies buffer length 20 bytes
- Confirm user payment with **pay-fees**
- Store the name preorder in the map:
  - hashed-salted-fqn
  - tx-sender principal
  - block-height
  - Claimed status
  - Price

Salting is adding randomness to the hashing process to force uniqueness and increase complexity, without increasing user requirements

```
;; preorder a name by registering a hash of the salted name
;; tx-sender has to pay registration fees here
;; returns the blockheight before the name has to be revealed
(define-public (name-preorder (hashed-salted-fqn (buff 20)))
  (let ((price (var-get price-in-ustx))
        (former-preorder
            (map-get?
 name-preorders { hashed-salted-fqn: hashed-salted-fqn, buyer: tx-sender })))
    ;; ensure eventual former pre-order expired
    (asserts!
      (or (is-none former-preorder)
          (≥ block-height (+ name-preorder-claimability-ttl
                              (unwrap-panic (get created-at former-preorder)))))
      err-preorder-already-exists)
    ;; ensure that the hashed fqn is 20 bytes long
    (asserts! (is-eq (len hashed-salted-fqn) u20) err-hash-malformated)
    ;; ensure that user will be paying. First to escrow, then to community on rev
    (try! (pay-fees price none))
    ;; store the pre-order
    (map-set name-preorders
      { hashed-salted-fqn: hashed-salted-fqn, buyer: tx-sender }
      { created-at: block-height, claimed: false, price: price })
    (ok (+ block-height name-preorder-claimability-ttl))))
```

name-register Function

Passes in:

- Namespace
- Desired name registration
- Salted hash
- Approval signature from user
- Principal of name purchaser
- Zonefile hash

Pulls preorder from **name-preorder** map

Ensures preorder entry is unclaimed, and less than 24 hours have passed since preorder.

The user (tx-sender) burns 1 STX

Community-handles burns 1 STX

NFT of name is then transferred to tx-sender

- **pay-fees-from-escrow**
- **stx-transfer?**
- Contract call v2 registration contract

```
;; @desc register an ordered name, this is the second tx of the registration flow
;; @event: tx-sender sends 1 stx
;; @event: community-handles burns 1 stx
;; @event: community-handles sends name nft to tx-sender
(define-public (name-register (namespace (buff 20))
                              (name (buff 48))
                              (salt (buff 20))
                              (approval-signature (buff 65))
                              (owner principal)
                              (zonefile-hash (buff 20)))
  (let ((hashed-salted-fqn (hash160 (concat (concat (concat name 0x2e) namespace) salt)))
        (preorder (unwrap!
          (map-get? name-preorders { hashed-salted-fqn: hashed-salted-fqn, buyer: owner })
          err-not-found))
        (hash (sha256 (concat (concat (concat name 0x2e) namespace) salt))))
    ;; Name must be approved by current approver
    (asserts! (secp256k1-verify hash approval-signature (var-get approval-pubkey))
 err-not-authorized)
    ;; The preorder entry must be unclaimed
    (asserts!
        (not (get claimed preorder))
        err-name-already-claimed)
    ;; Less than 24 hours must have passed since the name was preordered
    (asserts!
        (< block-height (+ (get created-at preorder) name-preorder-claimability-ttl))
        err-name-claimability-expired)
    (map-set renewal-signatures approval-signature true)
    (try! (pay-fees-from-escrow (get price preorder) namespace))
    (try! (stx-transfer? u1 tx-sender (as-contract tx-sender)))
    (try! (as-contract (contract-call?
 .community-handles-v2 name-register namespace name owner zonefile-hash)))
    (ok true)))
```

Admin Functions

set-price: Sets the default price of a name registration

set-community-treasury: Sets the designated principal to act as a particular namespace's treasury

set-approval-pubkey: Configures a new public key for approvals

set-contract-owner: Update the principal that should be recognized as the owner of the contract

set-namespace-controller: Transfers namespace ownership

Checks if user calling function is contract owner

```
(define-public (set-price (amount-in-ustx uint))
    (begin
        (try! (is-contract-owner))
        (var-set price-in-ustx amount-in-ustx)
        (ok true)))

(define-public (set-community-treasury (namespace (buff 20)) (new-treasury principal))
   (begin
        (try! (is-contract-owner))
        (map-set community-treasuries namespace new-treasury)
        (ok true)))

(define-public (set-approval-pubkey (new-pubkey (buff 33)))
   (begin
        (try! (is-contract-owner))
        (var-set approval-pubkey new-pubkey)
        (ok true)))


(define-public (set-contract-owner (new-owner principal))
    (begin
        (try! (is-contract-owner))
        (var-set contract-owner new-owner)
        (ok true)))

;; hand over control of namespace to new controller
;; can only be called by contract owner of this contract
(define-public (set-namespace-controller (namespace (buff 20)) (new-controller principal))
    (begin
        (try! (is-contract-owner))
        (try! (as-contract (contract-call?
 .community-handles-v2 set-namespace-controller namespace new-controller)))
        (ok true)))
```

Claim Fees Function

Allows the namespace controller to claim unused fees from escrow

```
;; @desc retrieve unused fees from escrow
;; If name-register wasn't called successfully the community amount is in escrow
;; and can be claimed by the controller-admin
(define-public (claim-fees (hashed-salted-fqn (buff 20)) (owner principal))
    (let ((preorder (unwrap!
            (map-get? name-preorders { hashed-salted-fqn: hashed-salted-fqn, buyer: owner }
)
            err-not-found))
          (price (get price preorder))
          (amount-controller-admin (/ (* price u70) u100))
          (amount-community (- price amount-controller-admin)))
      (asserts! (not (get claimed preorder)) err-invalid-claim)
      (asserts! (> block-height (+ (get created-at preorder) name-preorder-claimability-ttl
)) err-too-early)

      (and (> amount-community u0)
           (try! (as-contract (stx-transfer? amount-community tx-sender (var-get
 contract-owner)))))
      (ok true)))
```

Developers, we're listening!
Tell us about your experience using Hiro products

survey.hiro.so

Hiro

Are you looking for the roadmap, timelines, open feature requests, or report problems?

github.com/hirosystems

docs.hiro.so/roadmap

Hiro Developers | Documentation | Stacks API | Tutorials | Example Apps | Roadmap | Get Updates

Product Roadmap

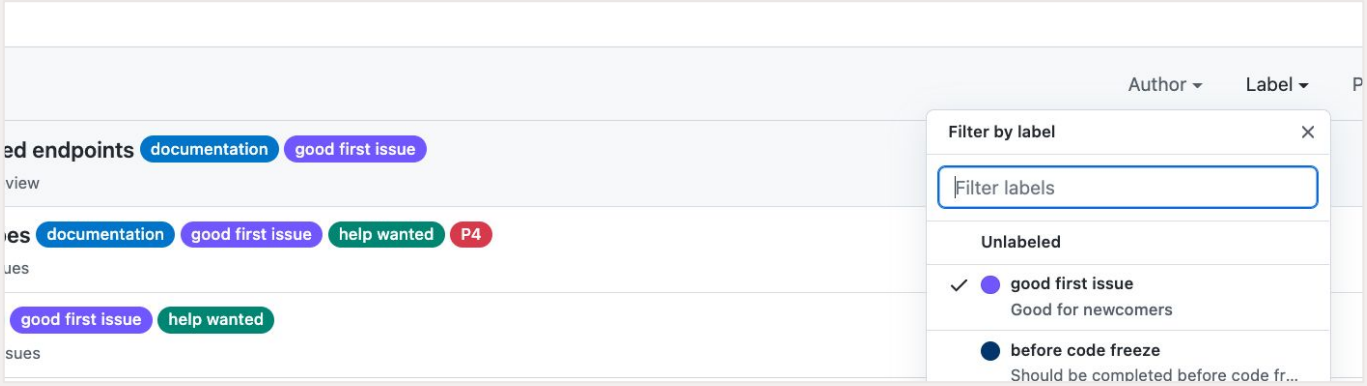
At Hiro, we want to make development on Stacks a fast and painless experience for developers. To achieve that goal, Hiro's product teams will focus on the following high-level themes in Q3 2022.

Product	Q3-2022
Clarinet	hirosystems/clarinet hirosystems/clarity-lsp hirosystems/orchestra
Explorer	hirosystems/explorer
Stacks Blockchain API	hirosystems/stacks-blockchain-api
Stacks.js	hirosystems/stacks.js
Subnets	hirosystems/stacks-subnets

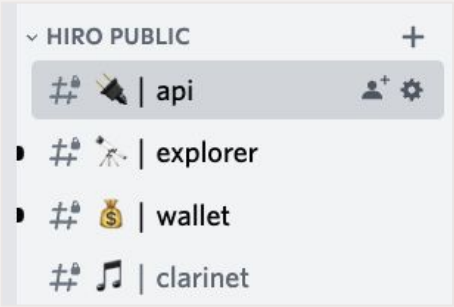
Ask questions, share your feedback, or probe for clarification by commenting on the issues associated with each project's GitHub milestones listed above.

New to Hiro, and are looking for ways to contribute?

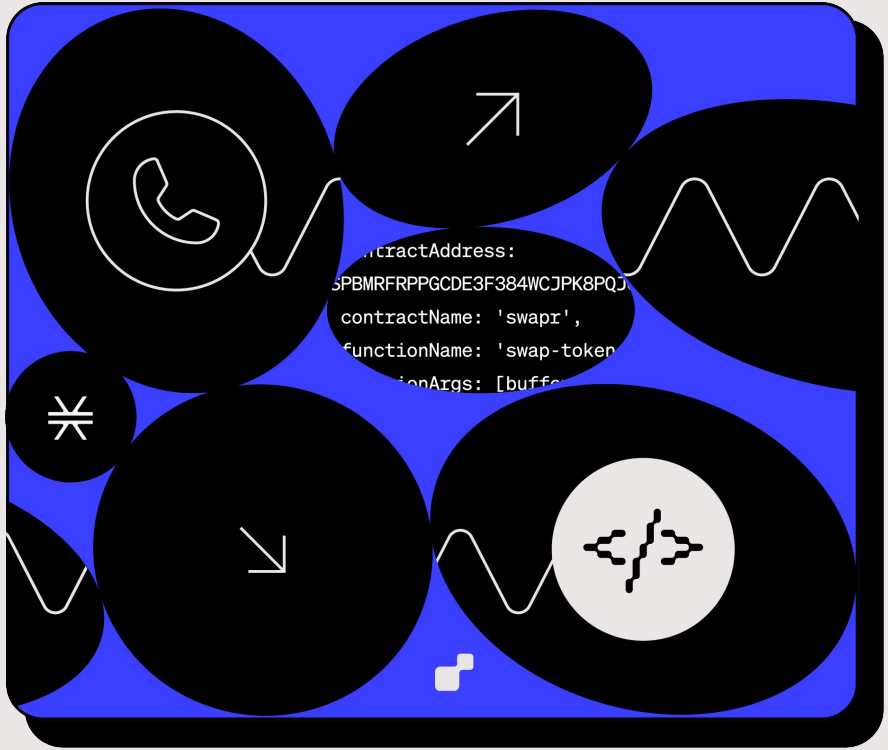
github.com/hirosystems



Are you wondering how to engage with Hiro or ask questions?



Thank You



Hiro Developers