Single tx swap - a trustless multisignature escrow service - Specification

Daniel Oravec

9. 3. 2023

1 Overview

Single-tx-swap is a trustless escrow service built for the Ergo blockchain. A swap of assets between two parties happens in a single transaction. The service works as a GUI multisignature transaction builder for participants who would like to exchange some of their assets. Both parties involved have to sign the same transaction in order for the swap to take place.

Users that *single-tx-swap* aims at are mostly people who like trading NFTs for other NFTs and fungible tokens on Discord or other communication platforms. Discord channels of NFT projects usually have a dedicated channel where members can post their NFTs with information about what they would like to trade them for. Once two people interested in a trade agree on the contents of the trade, they need an escrow service in order for them to not get tricked by each other.

An escrow service utilizing a smart contract already exists. However, it has several shortcomings that we would like to solve.

- Two transactions are needed for the swap.
- Tokens can only be exchanged for one token in one swap.
- Users need to either understand the contract's code or trust the service provider.
- Users might be scared of locking their funds in a smart contract.

Therefore, we propose a solution using multisignature transactions that solves all of these problems. However, it requires users to perform the swap at the same time, which is a disadvantage against the smart contract solution. We only consider this disadvantage marginal for our use-case.

2 Requirements

There are two main user roles:

1. Trading session **host**
2. Trading session **guest**

Besides that, everyone has a **visitor** role. Anyone who connected their wallet has an **authenticated user** role. Once two parties agree on the swap, they decide which one of them will be the session host. The host's role is to visit our website, connect their wallet and create a private trading session. After that, they will send a private invite link to the other party, who will be the session guest.

The host will wait until the guest joins the trading session and connect their wallet. Once that happens, The host is presented with wallet contents of both themselves and the guest. The host selects NFTs to swap from both wallets. Besides that, they specify amounts of fungible tokens, including ERG that they would like to swap too. While the host is building the swap, the guest is asked to wait.

After the host is done building the swap, they are asked to partially sign the transaction. By doing this, they allow spending their inputs in such a transaction. Inputs of the guest are still unsigned and therefore, the host can't submit the transaction to the network, as it would get rejected by the validator. This partially signed transaction is then sent to the guest and the host is asked to wait until the guest also signs the transaction. Once the guest does that, the transaction is fully signed and is submitted to the network.

If the transaction is accepted by the Ergo blockchain, the swap is performed. If the transaction is rejected, nothing happens and participants do not lose any funds. If both participants carefully review the transaction in their wallet before signing it, there is no risk for users.

3 User stories

1. As a **visitor**, I can connect my wallet, so that I become an **authenticated user**.
2. As an **authenticated user**, I can disconnect my wallet, so that I become a **visitor**.
3. As an **authenticated user**, I can push a "start trading session" button, so that I become a session **host**.
4. As an **authenticated user**, I can visit a private trading session by clicking a link received by the session **host**, so that I become a session **guest**.
5. As a session **host**, I can send an invite link to someone, wait for them to become a **guest**, then be presented with wallet contents of myself and the **guest** and then select tokens to swap, so that I am presented with a prompt to review and partially sign a swap transaction.
6. As a **guest**, after I enter the trading session, I can wait until I'm presented with a transaction partially signed by the **host** and then review it and finalize the signing process if I agree with it, so the transaction is submitted to the network.
7. As an **authenticated user**, I can visit my profile and fill my contact info there, so that this info will be visible on my profile for other **visitors**.
8. As a **visitor**, I can search for the contact info of a specific NFT holder, so that I can contact them outside of the platform if I want to buy that NFT.
9. As an **authenticated user**, I can send a message to any other **authenticated user**.
10. As an **authenticated user**, I can send a message to a holder of any specific NFT based on their address, regardless of whether they have a profile or not.
11. As a **visitor**, I can visit any profile, so that I can view wallet contents, history and statistics of that profile.

4 Data model

A description of all needed database tables follows.

sessions
`id: INTEGER;`

- Surrogate key

`secret: STRING;`

- A unique 16-byte unpredictable ID of the session, hex encoded.

`creator_addr: STRING;`

- An address of the session host.

`creator_assets_json: JSON;`

- JSON of all assets held by the creator's address at the time of creating the session. This data will be fetched by the BE based on `creator_addr`, so that the host cannot lie about their assets.

`creator_nano_erg: BIGINT;`

- Amount of ERG (Ergo's native token) held by the creator times 10^9, similar to `creator_assets_json`.

`guest_addr: STRING;`

- An address of the session guest.

`guest_assets_json: JSON;`

- Like creator_assets_json, but for the guest.

`guest_nano_erg: BIGINT;`

- Like creator_nano_erg, but for the guest.

`unsigned_tx: JSON;`

- A swap transaction with no witnesses yet.

```
unsigned_tx_added_on: DATE;
signed_inputs_creator: JSON;
```

- A list of signed creator's transaction inputs. This will be used to assemble the final signed transaction.

`tx_input_indices_creator: ARRAY(INTEGER);`

- Which transaction inputs are creator's. This is so that we can place `signed_inputs_creator` to correct places in the final transaction's inputs.

`tx_input_indices_guest: ARRAY(INTEGER);`

- Like `tx_input_indices_creator`, but for the guest.

`tx_id: STRING;`

- Transaction id of the swap transaction (a has of the serialized transaction).

```
submitted_at: DATE;
created_at: DATE;
```

users

`address: STRING`

- PK, wallet address, not home address.

`username: STRING; email: STRING; discord: STRING;`

- Discord username

`twitter: STRING;`

- Twitter username

`allow_messages: BOOL;`

- Whether receiving messages is allowed.

assets

`token_id: STRING;`

- PK, unique token ID on the blockchain.

`decimals: INTEGER;`

- To keep all computations in whole numbers.

`is_verified: BOOL;`

- Whether the asset is authentic.

user_session_stats

`user_address: STRING;`

- PK, address of the respective user.

`sessions_hosted: INTEGER;`

- Number of sessions hosted by this user.

`sessions_visited: INTEGER;`

- Number of sessions where this user was a guest.

user_asset_stats
`user_address: STRING;`

- Address of the respective user.

`token_id: STRING;`

- Asset's unique ID on the blockchain.

`amount_bought: BIGINT;`
`amount_sold: BIGINT;`

follows
`user_address: STRING; followed_address: STRING;`

- Address that is followed by the user specified in `user_address`.

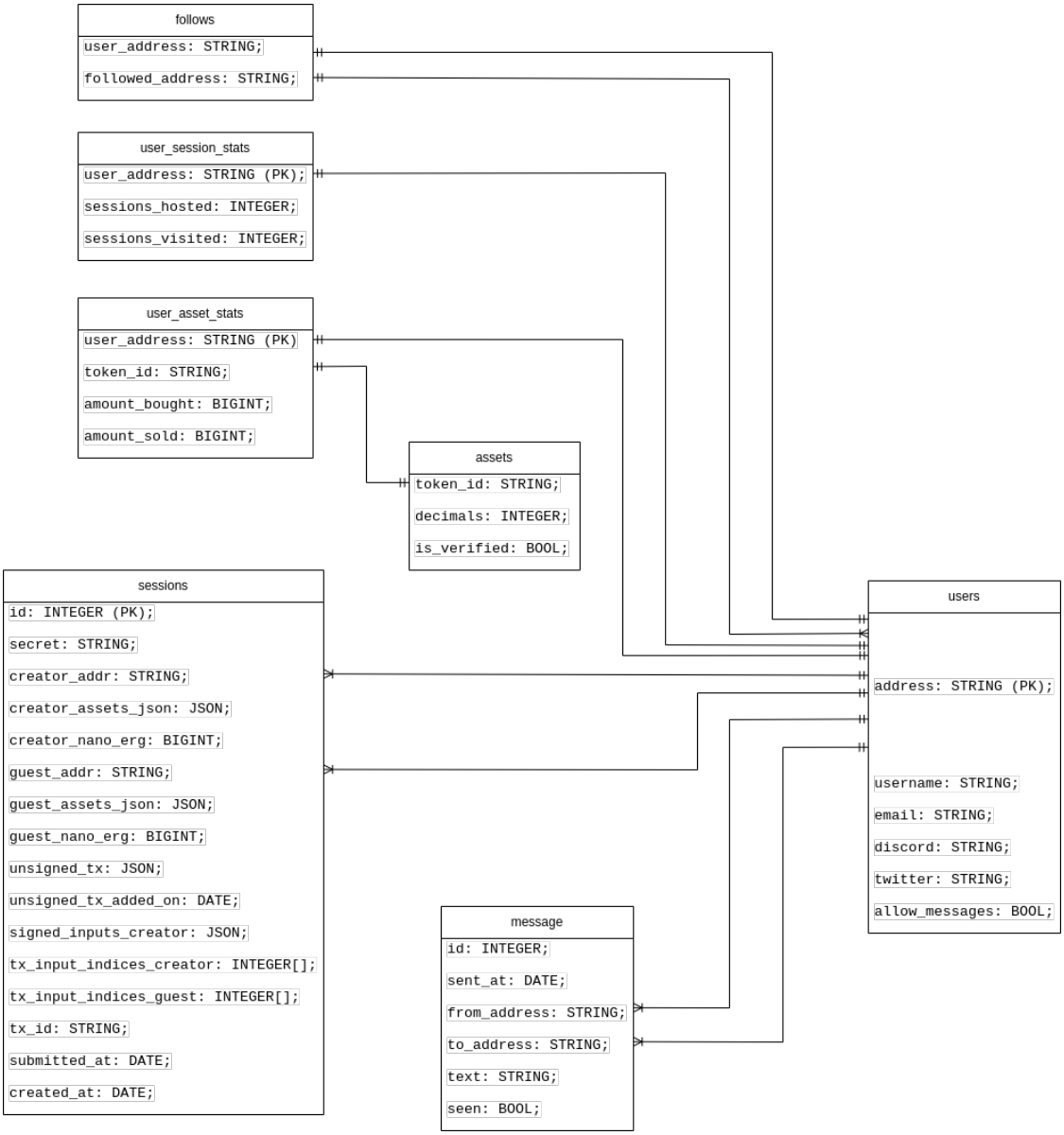
messages
`id: INTEGER;`
`sent_at: DATE;`
`from_address: STRING;`

- Blockchain address of the sender.

`to_address: STRING;`

- Blockchain address of the receiver.

`text: STRING;`
`seen: BOOL;`



5 Technological requirements

This will be a Typescript Next.js(13.1.6) React(18.2.0) application using styled-components(5.3.6) for styling. For state management, such as handling wallets, Zustand(4.3.2) will be used.

Transactions will be built using FleetSDK(>=0.1.0-alpha19). In case more complicated computations will be needed, ergo-lib will be used.

As this is a Next.js app, we chose Vercel for hosting it.

The backend will be written in Typescript (4.9.5) using Express.js(4.18.2). FleetSDK(>=0.1.0-alpha19) will be used for validating transactions received from users. Our backend will provide a REST API.

Postgres(13.7) is a database system of our choice.

The backend will be hosted on AWS Elastic Beanstalk with AWS Codebuild for CI/CD.

Major browsers such as Chrome (+Brave) and Firefox should be supported.

6 Time plan

25. 3. 2023: Working POC

- Homepage and swap UI design in Figma

- Wallet connection
- Implement footer, navbar
- Theming (dark, light)
- `[POST] /tx/register`
- `[GET] /tx`
- `[GET] /tx/partial`
- `[POST] /tx/partial/register`
- `[POST] /session/create`
- `[POST] /session/enter`
- Landing page implementations (with FAQs)
- Swap UI and logic implementation
- Parse assets according to EIP-0004 to get image links and display images
- Add SVG icons for verified fungible tokens
- Setup domains, Vercel, Elastic Beanstalk and Codebuild

1. 4. 2023: Profile section and database improvements

- Design and partially implement profile page
- List of assets (no history yet)
- Edit profile
- Add contact info (no profile picture yet)
- Use migrations

11. 4. 2023: Finalize profile section, add messaging

- Implement messaging system
- Design and implement a page for viewing messages
- Also delete, mark as unread
- Add notification bell to navbar for messages

30. 4. 2023: Statistics

- Implement statistics (sessions created, visited, amount traded too if possible) in profile section
- Implement swap history in profile section
- [TBD] Global rankings page

2. 5. 2023: Add search users functionality

- Design and implement a page for searching a user by tokenID that they hold
- Contact info will be displayed if such user is found