High-Level Tokenomics Concept

MY HOPE OF ALL HOPES IS THAT THIS PLAN IS CHALLENGED. TELL MY WHY IT'S STUPID AND HOW IT CAN BE BETTER. THIS IS NOT THE ONLY WAY.

Here we'll show the inflows and outflows.

Expenses and who pays them.

(1) Book Hosting: Paid by users in ICP via Canisters (Non-Custodial).
(2) Search Engine Hosting: Paid by users in ICP via IC_Akash Canister (Non-Custodial).
(3) Storage of User Data/Selected Content: Paid by our canisters/me (Blackholed if dedicated to storage).

That's all of our project's expenses, so pretty much to an infinite scale, salaries are the only major expense.

Incoming Revenue and who pays it.

Users who want to save stuff, and possibly other interactions, will do so in ICP.

For now, this will just be with bookmarking, where a user pays 2 cents or something to store a piece of a book.

Later they will pay on a per-token basis for use of AI with those bookmarked snippets.

Rewards Distribution and Who Gets it.

Simply put, the incoming revenue is split 50/50.

Half goes (directly) to the people in the first section. The ones paying the expenses.

Half goes (indirectly) to the people in the second section. The one's paying the revenue.

HOW?

The first half is simple in principle, complex in the details: The 2 cents used to bookmark a snippet/source card/search result (these are all the same, I need a fitting name), is owed in ICP to the owner of the book (NFT) that it came from .

The second half does not go directly to the one who spent the money. Instead, it just goes to stakers of $UCG. The reason this has earning potential for people who pay the 2 cents to save a bookmark, is if others like it they earn UCG.

HOW in detail:

The first half: Since the ICP transaction fee is significant enough to make the 2-cent micropayments unbearable, we'll need to use an intermediary token. I'm calling these "GLYPHs" because they're used to engrave a symbol permanently in cypherspace.

Put simply, users who wish to save their search results and use other advanced features will have to send ICP to the glyph canister, which mints glyphs and sends it to their account. When a user bookmarks something, it's account automatically burns a glyph by sending it to the minting canister.

Later we'll adapt that canister to actively manage and disperse the collected ICP to Book Owners and UCG Stakers. We'll also attach the bookmarks to the user principal, and when other users 'like' their bookmarks, UCG emitted into the circulating supply will go to them. But I'm not worrying about any of this right now.

All I want working now is a set of canisters that gets and holds ICP, and in exchange mints and burns GLYPHs. After that we can worry a mechanism to distribute that ICP to book owners (perhaps they are swapped into RUNEs and given to book owners who can exchange for their share of ICP, or we make our own ledger tracking/claim system).

Similar Work

- A Canister that can Transfer Tokens from a Canister: https://github.com/dfinity/examples/tree/master/rust/token_transfer
- Canister that can transfer on behalf of other accounts with ICRC-2: https://github.com/dfinity/examples/tree/master/rust/token_transfer_from
- Canister that can Transfer ICP: https://github.com/dfinity/examples/tree/master/rust/icp_transfer
- Optional approach if I want to do a direct token swap, ICP:GLYPHs: The problem is it's based on a 2-yr old from Psychedelic (who are no longer arond), and with the DIP-20 standard, so not ICRC. https://github.com/dfinity/examples/tree/master/rust/defi
- This is a great Motoko only example of swapping 1 ICRC token to another with the approve/withdraw/etc. Extensibility of ICRC-2. https://github.com/dfinity/examples/tree/master/motoko/icrc2-swap
-

Key Problems:

- You cannot trigger events easily from a token transfer, so a user can send a ICP to the swap/minting canister, but it's not easy to use that as a trigger to mint and send GLYPHs.

~~I can't seem to be too retarted to Get ICRC tokens minting properly because I'm retarted.~~
Figured out the minting function. Whooho. Will play around with doing this stuff manually before deciding how to automate. Now we just need to get the canister to be the caller with some trigger.

General plan given the problems:
~~Deploy the Token Canister and figure out minting.~~
Deploy the ICP custodian canister. And the GLYPH
Figure out how to link the minting and dispersing with the ICP send/deposit.

I already deployed these canisters at separate times. It's simple but I don't know how to add the logic to link them.

Fresh Plan:

I'm getting lost in mixing all these examples so I have to start fresh.

I'm going to set up the ICP transfer Canister, get it live on the network.
In the same repo, I'm going to deploy the GLYPH canister.

Once on the network, I'm going to test what it's like to manually send and receive ICP; and mint, send, and burn GLYPH.

Then I'm going to use the ICRC1 Token-Tansfer Repo to hold and send the GLYPH.

So 3 canisters in the Tokenomics repo.

ReadMe & Notes of Current Working Repo

Simple Tansfer Repo (Canister is Custodian)

```
dfx canister call icp_transfer_backend transfer "(record { amount = record { e8s = 100_000_000 }; to_principal = principal \"$(dfx identity --identity default get-principal)\"})"
```

OWNER (default)=mxtax-xmovu-wu5th-gdf4k-vfkdn-ffsxn-e67ju-sidls-4dr2i-3mqoe-tae
ALICE=67y6u-aw7zf-px4fv-7sesq-tnpcm-gozwp-ojr5j-ansxk-rkwwo-32ahb-mqe
BOB=2iebt-4nyhk-hw3oq-wisaz-picsh-6h76z-urotm-ud7bo-ayafs-hlk36-oae

Token Canisters (deployed ICRC-2)

GLYPHs, we'll call the token glyphs, since their purpose is taking symbols to engrain forever in cypherspace.

(1) cd src/icp_ledger_canister
(2) dfx identity use minter
(3) export MINTER_ACCOUNT_ID=$(dfx ledger account-id)
(4) dfx identity use default
(5) export DEFAULT_ACCOUNT_ID=$(dfx ledger account-id)
(6) dfx canister create GLYPH --specified-id hdtfn-naaaa-aaaam-aciva-cai
(7) Deploy the GLYPH token

```
dfx deploy GLYPH --argument '
 (variant {
   Init = record {
     token_name = "GLYPHs";
     token_symbol = "GLYPH";
     minting_account = record {
       owner = principal "'${DEFAULT}'";
     };
     initial_balances = vec {
       record {
         record {
           owner = principal "'${MINTER}'";
         };
         100_000_000_000;
       };
     };
     metadata = vec {};
     transfer_fee = 10_000;
     archive_options = record {
       trigger_threshold = 2000;
       num_blocks_to_archive = 1000;
       controller_id = principal "'${DEFAULT}'";
     };
     feature_flags = opt record {
       icrc2 = true;
     };
   }
```

```
  })
'
```

(8) Deploy the ledger canister.

```
dfx deploy --specified-id ryjl3-tyaaa-aaaaa-aaaba-cai icp_ledger_canister --argument "
  (variant {
    Init = record {
      minting_account = \"$MINTER_ACCOUNT_ID\";
      initial_values = vec {
        record {
          \"$DEFAULT_ACCOUNT_ID\";
          record {
            e8s = 10_000_000_000 : nat64;
          };
        };
      };
      send_whitelist = vec {};
      transfer_fee = opt record {
        e8s = 10_000 : nat64;
      };
      token_symbol = opt \"LICP\";
      token_name = opt \"Local ICP\";
    }
  })
"
```

(9) Dfx deploy (to deploy the main backend canister).

(10) Determine the address of the backend canister:

```
TOKENS_TRANSFER_ACCOUNT_ID="$(dfx ledger account-id --of-canister
icp_transfer_backend)"
TOKENS_TRANSFER_ACCOUNT_ID_BYTES="$(python3 -c 'print("vec{" + ";".join([str(b) for b
in bytes.fromhex("'$TOKENS_TRANSFER_ACCOUNT_ID'")]) + "}")')"
```

(11) Transfer ICP Funds to the Canister:

```
dfx canister call icp_ledger_canister transfer "(record { to =
${TOKENS_TRANSFER_ACCOUNT_ID_BYTES}; memo = 1; amount = record { e8s =
2_00_000_000 }; fee = record { e8s = 10_000 }; })"
```

(12) Transfer GLYPH Funds to the Canister:

```
dfx canister call GLYPH icrc2_transfer_from "(record { to =
${TOKENS_TRANSFER_ACCOUNT_ID_BYTES}; memo = 1; amount = record { e8s =
2_00_000_000 }; fee = record { e8s = 10_000 }; })"
```

(13) MINT GLYPHs!!!

```
evan@Henry:~/icp_transfer$ dfx canister call GLYPH icrc1_transfer '(
  record {                                                  to = record {
      owner = principal "ie5gv-y6hbb-ll73p-q66aj-4oyzt-tbcuh-odt6h-xkpl7-bwssd-lgzgw-5qe";
      subaccount = opt vec {
        0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
        0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      };
    };

    from = record {
      owner = principal "mxtax-xmovu-wu5th-gdf4k-vfkdn-ffsxn-e67ju-sidls-4dr2i-3mqoe-tae";
      subaccount = opt vec {
        0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
        0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
      };
    };

    amount = 10000;
    fee = null;
  }
)'
```

```
export TOKEN_A=$(dfx canister id token_a)
export TOKEN_B=$(dfx canister id token_b)

dfx deploy --network ic swap --argument '
  record {
    token_a = (principal "qhbym-qaaaa-aaaaa-aaafq-cai");s
    token_b = (principal "hdtfn-naaaa-aaaam-aciva-cai");
  }
'
```

Swap Canister ID= ju4sh-3yaaa-aaaap-ahapa-cai
Swap Canister Deployment:
https://a4gq6-oaaaa-aaaab-qaa4q-cai.raw.icp0.io/?id=ju4sh-3yaaa-aaaap-ahapa-cai

Working Transfer Command (NOT REALLY):

```
dfx canister call --network local icp_transfer_backend swap "(record { amount = record { e8s = 10_000_000 }; to_principal = principal \"$(dfx identity get-principal)\"})"
```

Check the balance when done:

```
dfx canister call GLYPH icrc1_balance_of '(record { owner = principal "mxtax-xmovu-wu5th-gdf4k-vfkdn-ffsxn-e67ju-sidls-4dr2i-3mqoe-tae" })'
```

Fresh Plan (Tokenomics Repo) Readme