Purpose

- The wallet canister:
    - Queries two external NFT canisters (using their contract addresses) to count NFTs held by users' principals.
    - Stores token balances for users, updated by the payout canister after payouts.
    - Provides query methods for NFT counts and balances.

Assumptions

- The two NFT collections are hosted in separate canisters with known canister IDs (contract addresses).
- These NFT canisters support a method to query ownership (e.g., an ICRC-7-like `getOwnedNFTs` function).
- The payout canister will handle token distribution; the wallet canister only tracks balances and NFT counts.

Requirements

- **DFX CLI**: Installed and configured.
- **Motoko**: For canister development.
- **NFT Canister IDs**: The contract addresses (canister IDs) of the two NFT collections.
- **NFT Interface**: A method to query NFT ownership (e.g., `getOwnedNFTs(principal)` returning an array of NFT IDs).

Steps to Set Up the Wallet Canister

1. **Create the Wallet Canister File**
    - In your DFX project (e.g., `nft_payout_system`), create `src/wallet/main.mo`.
2. **Define the Canister Logic**
    - Here's the updated Motoko code:
    - `motoko`

```
import Principal "mo:base/Principal";
import HashMap "mo:base/HashMap";
import Nat "mo:base/Nat";
import ICRC7 "mo:icrc7"; // Hypothetical ICRC-7 interface for NFTs

actor Wallet {
  // Map of user principal to total NFT count across two collections
  private stable var nftCounts = HashMap.HashMap<Principal, Nat>(10, Principal.equal, Principal.hash);
  // Map of user principal to token balance
  private stable var tokenBalances = HashMap.HashMap<Principal, Nat>(10, Principal.equal,
Principal.hash);
```

```
// Define the two NFT contract addresses (canister IDs)
let nftCanister1 = actor ("<NFT_CONTRACT_ADDRESS_1>") : ICRC7.NFT;
let nftCanister2 = actor ("<NFT_CONTRACT_ADDRESS_2>") : ICRC7.NFT;

// Update NFT count for a given user
public shared func updateNFTCount(user : Principal) : async Nat {
  // Query ownership from first NFT contract
  let ownedNFTs1 = await nftCanister1.getOwnedNFTs(user);
  let count1 = ownedNFTs1.size();

  // Query ownership from second NFT contract
  let ownedNFTs2 = await nftCanister2.getOwnedNFTs(user);
  let count2 = ownedNFTs2.size();

  // Total NFT count
  let totalCount = count1 + count2;
  nftCounts.put(user, totalCount);
  totalCount
};

// Get total NFT count for a user (query)
public query func getNFTCount(user : Principal) : async Nat {
  switch (nftCounts.get(user)) {
    case (?count) count;
    case null 0;
  }
};

// Get NFT count per contract (optional, for transparency)
public shared func getNFTCountByContract(user : Principal) : async { contract1 : Nat; contract2 : Nat } {
  let ownedNFTs1 = await nftCanister1.getOwnedNFTs(user);
  let ownedNFTs2 = await nftCanister2.getOwnedNFTs(user);
  {
    contract1 = ownedNFTs1.size();
    contract2 = ownedNFTs2.size();
  }
};

// Update token balance (called by payout canister)
public shared ({ caller }) func updateBalance(user : Principal, amount : Nat) : async () {
  // Restrict to payout canister
  assert(caller == Principal.fromText("<PAYOUT_CANISTER_ID>"));
  let current = switch (tokenBalances.get(user)) {
    case (?bal) bal;
    case null 0;
  };
  tokenBalances.put(user, current + amount);
};
```

```
  // Get token balance (query)
  public query func getBalance(user : Principal) : async Nat {
    switch (tokenBalances.get(user)) {
      case (?bal) bal;
      case null 0;
    }
  };

  // Stable storage for upgrades
  system func preupgrade() {
    // nftCounts and tokenBalances are stable
  };

  system func postupgrade() {
    // No additional initialization
  };
```

- };
- **Key Changes**:
  - `updateNFTCount` now takes a `user` parameter (Principal) to query any user's wallet, not just the caller.
  - Removed collection addresses within a canister, assuming each contract address (canister ID) represents one collection.
  - Replace `<NFT_CONTRACT_ADDRESS_1>`, `<NFT_CONTRACT_ADDRESS_2>`, and `<PAYOUT_CANISTER_ID>` with actual canister IDs.
  - Adjust `getOwnedNFTs` to match the NFT canisters' actual interface (e.g., it might be `balanceOf` or similar).

3. **Configure** `dfx.json`
   - Ensure the wallet canister is listed:
   - `json`

```
{
  "canisters": {
    "wallet": {
      "main": "src/wallet/main.mo",
      "type": "motoko"
    }
  }
```

- }

4. **Deploy Locally**
   - Start the local IC replica:
   - `bash`
   - dfx start --background

- Deploy:
- `bash`
- dfx deploy wallet
- Record the canister ID.

5. **Test the Wallet Canister**
   - **Update NFT Count**:
   - `bash`
   - dfx canister call wallet updateNFTCount '(principal "<USER_PRINCIPAL>")'
     - Replace `<USER_PRINCIPAL>` with a test user's principal.
   - **Query NFT Count**:
   - `bash`
   - dfx canister call wallet getNFTCount '(principal "<USER_PRINCIPAL>")'
   - **Query NFT Count by Contract**:
   - `bash`
   - dfx canister call wallet getNFTCountByContract '(principal "<USER_PRINCIPAL>")'
   - **Simulate Balance Update** (after payout canister setup):
   - `bash`
   - dfx canister call wallet updateBalance '(principal "<USER_PRINCIPAL>", 100)'
     - Requires payout canister deployment for real testing.
6. **Integrate with NFT Canisters**
   - Confirm the NFT canisters' interface. Example:
     - If they use `getOwnedNFTs(principal) : async [Nat]`, the code works as is.
     - If they use `balanceOf(principal) : async Nat`, modify `updateNFTCount` and `getNFTCountByContract` to call `balanceOf` instead.
   - Test with mock NFT canisters locally if the real ones aren't available:
   - `motoko`

```
actor MockNFT {
  public query func getOwnedNFTs(user : Principal) : async [Nat] {
    if (user == Principal.fromText("<TEST_PRINCIPAL>")) {
      return [1, 2, 3]; // Simulate 3 NFTs
    } else {
      return [];
    }
  };
```

   - };
7. **Deploy to Mainnet**
   - Create canister:
   - `bash`

    - dfx canister create wallet --network ic
    - Deploy:
    - `bash`
    - dfx deploy wallet --network ic
    - Fund with cycles.
8. **Verify Functionality**
    - Call `updateNFTCount` for a user with NFTs in either contract.
    - Check `getNFTCount` and `getNFTCountByContract` for accuracy.
    - Test `updateBalance` with the payout canister.

Integration with Payout Canister

The payout canister (from my first response) needs slight adjustments to work with this wallet canister:

- Update the wallet canister interface:
- `motoko`

```
let walletCanister = actor ("<WALLET_CANISTER_ID>") : actor {
  getNFTCount : (Principal) -> async Nat;
  updateBalance : (Principal, Nat) -> async ();
```

- };
- In `processPayouts`, call `updateNFTCount` before calculating:
- `motoko`

```
private func processPayouts() : async () {
  for (user in users.keys()) {
    ignore await walletCanister.updateNFTCount(user); // Refresh NFT count
    let nftCount = await walletCanister.getNFTCount(user);
    let payout = calculatePayout(nftCount);
    if (payout > 0) {
      await tokenCanister.transfer(user, payout);
      await walletCanister.updateBalance(user, payout);
    };
  };
```

- };
- This ensures the NFT count is fresh for each payout cycle.

Example Workflow

1. User A (Principal: `abcde-...`) holds 2 NFTs in Contract 1 and 1 NFT in Contract 2.
2. Payout canister calls `walletCanister.updateNFTCount("abcde-...")`:
    - Contract 1: 2 NFTs.
    - Contract 2: 1 NFT.
    - Total = 3 NFTs (stored in `nftCounts`).
3. Payout canister queries `getNFTCount("abcde-...")` → 3.
4. Calculates payout:
    - Total value = 3 * 1000 = 3000.
    - Annual payout = 3000 * 0.10 = 300.
    - Per 5-day payout = 300 / 73 ≈ 4 tokens.
5. Transfers 4 tokens to User A and calls `updateBalance("abcde-...", 4)`.

Notes

- **NFT Contract Interface**: If the NFT canisters don't support `getOwnedNFTs`, adapt the code to their specific methods (e.g., `ownerOf`, `tokensOf`).
- **Efficiency**: Querying NFT ownership for many users could be slow; consider batching or caching counts if the NFT canisters support subscriptions or events.
- **Security**: Ensure only the payout canister can call `updateBalance`.