Breakdown for Setting Up the Payout Canister

Purpose

- The payout canister:
    - Registers users who want to receive payouts.
    - Periodically (every 5 days) queries the wallet canister for each user's NFT count across two NFT collections.
    - Calculates the payout (10% APY, divided into 5-day intervals) based on the NFT count.
    - Transfers tokens to users via an ICRC-1 token canister and updates their balances in the wallet canister.

Assumptions

- The wallet canister is deployed with the interface from the previous response.
- An ICRC-1 compliant token canister exists for payouts.
- Each NFT (from either collection) contributes 1000 units to the payout calculation.
- Payouts occur every 5 days, with NFT counts refreshed weekly at payout time.

Requirements

- **DFX CLI**: Installed and configured.
- **Motoko**: For canister development.
- **Wallet Canister ID**: From the deployed wallet canister.
- **Token Canister ID**: From the ICRC-1 token canister.
- **Cycles**: For deployment and execution on the Internet Computer.

Steps to Set Up the Payout Canister

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

```
import Principal "mo:base/Principal";
import HashMap "mo:base/HashMap";
import Nat "mo:base/Nat";
import Time "mo:base/Time";
import Float "mo:base/Float";
import ICRC1 "mo:icrc1"; // Hypothetical ICRC-1 token interface

actor Payout {
```

```
  // Interface to the wallet canister
  let walletCanister = actor ("<WALLET_CANISTER_ID>") : actor {
    updateNFTCount : (Principal) -> async Nat;
    getNFTCount : (Principal) -> async Nat;
    updateBalance : (Principal, Nat) -> async ();
  };

  // Interface to the token canister
  let tokenCanister = actor ("<TOKEN_CANISTER_ID>") : ICRC1.Token;

  // Constants
  let BASE_VALUE_PER_NFT : Nat = 1000; // Each NFT = 1000 units
  let APY : Float = 0.10; // 10% annual percentage yield
  let SECONDS_PER_YEAR : Nat = 31_536_000; // 365 days in seconds
  let PAYOUT_INTERVAL : Nat = 432_000; // 5 days in seconds
  let PAYOUTS_PER_YEAR : Nat = SECONDS_PER_YEAR / PAYOUT_INTERVAL; // 73 payouts per
year

  // Stable variables for persistence across upgrades
  private stable var lastPayout : Time.Time = 0;
  private stable var usersEntries : [(Principal, ())] = [];
  private var users = HashMap.HashMap<Principal, ()>(10, Principal.equal, Principal.hash);

  // Initialize users map from stable storage
  system func postupgrade() {
    users := HashMap.fromIter(usersEntries.vals(), 10, Principal.equal, Principal.hash);
  };

  system func preupgrade() {
    usersEntries := Iter.toArray(users.entries());
  };

  // Register a user for payouts
  public shared ({ caller }) func register() : async () {
    users.put(caller, ());
  };

  // Calculate payout amount per user based on NFT count
  private func calculatePayout(nftCount : Nat) : Nat {
    let totalValue = nftCount * BASE_VALUE_PER_NFT;
    let annualPayout = Float.fromInt(totalValue) * APY;
    let perPayout = annualPayout / Float.fromInt(PAYOUTS_PER_YEAR);
    // Convert to Nat, rounding down (could use a fixed-point library for precision)
    Nat.fromFloat(perPayout)
  };

  // Heartbeat for periodic payouts (runs every few seconds)
  system func heartbeat() : async () {
    let now = Time.now();
```

```
  if (now >= lastPayout + PAYOUT_INTERVAL * 1_000_000_000) { // Convert seconds to nanoseconds
    await processPayouts();
    lastPayout := now;
  };
};

// Process payouts for all registered users
private func processPayouts() : async () {
  for (user in users.keys()) {
    // Refresh NFT count for the user
    ignore await walletCanister.updateNFTCount(user);
    let nftCount = await walletCanister.getNFTCount(user);
    let payout = calculatePayout(nftCount);
    if (payout > 0) {
      // Transfer tokens to the user
      let transferResult = await tokenCanister.transfer(user, payout);
      switch (transferResult) {
        case (#Ok(_)) {
          // Update balance in wallet canister
          await walletCanister.updateBalance(user, payout);
        };
        case (#Err(e)) {
          // Log or handle transfer failure (not implemented here)
        };
      };
    };
  };
};

// Manual trigger for testing
public shared func triggerPayout() : async () {
  await processPayouts();
};

// Query last payout time (for debugging)
public query func getLastPayoutTimeTime() : async Time.Time {
  lastPayout
};
```

    - };
    - **Key Features**:
        - **Wallet Integration**: Calls `updateNFTCount` and `getNFTCount` to refresh and retrieve NFT counts, and `updateBalance` to record payouts.
        - **Token Transfer**: Uses an ICRC-1 `transfer` method to send tokens.
        - **Heartbeat**: Checks every few seconds if 5 days have passed, then triggers payouts.
        - **Stable Storage**: Persists `lastPayout` and `users` across upgrades.

        - Replace `<WALLET_CANISTER_ID>` and `<TOKEN_CANISTER_ID>` with actual canister IDs.
3. **Configure** `dfx.json`
    - Add the payout canister to `dfx.json`:
    - `json`

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

    - }
4. **Deploy Locally**
    - Start the local IC replica:
    - `bash`
    - dfx start --background
    - Deploy:
    - `bash`
    - dfx deploy payout
    - Record the canister ID.
5. **Test the Payout Canister**
    - **Register a User**:
    - `bash`
    - dfx canister call payout register
    - **Trigger Payout Manually**:
    - `bash`
    - dfx canister call payout triggerPayout
    - **Check Last Payout Time**:
    - `bash`
    - dfx canister call payout getLastPayoutTime
    - Ensure the wallet canister is deployed and populated with test data (e.g., NFT counts).
6. **Integrate with Wallet and Token Canisters**
    - **Wallet Canister**: Already compatible; ensure it's deployed and its ID is updated in the code.
    - **Token Canister**: Verify it supports `transfer(principal, amount) : async Result`. Example mock:
    - `motoko`

```
actor MockToken {
  public shared func transfer(to : Principal, amount : Nat) : async Result.Result<Nat, Text> {
    #Ok(amount)
  };
```

    - };
7. **Deploy to Mainnet**
    - Create canister:
    - `bash`
    - dfx canister create payout --network ic
    - Deploy:
    - `bash`
    - dfx deploy payout --network ic
    - Fund with cycles via the cycles wallet.
8. **Verify Functionality**
    - Register a test user with NFTs in either contract.
    - Trigger a payout manually and check the wallet canister's `getBalance` for the user.
    - Wait 5 days (or simulate with a shorter interval locally) to confirm heartbeat triggers.

Payout Calculation Recap

- **Formula**:
    - Total value = `NFT count * 1000`.
    - Annual payout = `Total value * 0.10`.
    - Per 5-day payout = `Annual payout / 73`.
- **Example**:
    - User has 3 NFTs (2 from Contract 1, 1 from Contract 2).
    - Total value = 3 * 1000 = 3000.
    - Annual payout = 3000 * 0.10 = 300.
    - Per payout = 300 / 73 ≈ 4 tokens (rounded down).

Example Workflow

1. User A (Principal: `abcde-...`) registers with `register()`.

2. After 5 days, heartbeat triggers `processPayouts()`:
    - Calls `walletCanister.updateNFTCount("abcde-...")` → Updates to 3 NFTs.
    - Queries `walletCanister.getNFTCount("abcde-...")` → 3.
    - Calculates payout = 4 tokens.
    - Transfers 4 tokens via `tokenCanister.transfer("abcde-...", 4)`.
    - Updates balance with `walletCanister.updateBalance("abcde-...", 4)`.
3. User A's balance in the wallet canister increases by 4.

Additional Notes

- **Precision**: The `Nat.fromFloat` conversion rounds down; use a fixed-point library (e.g., `mo:base/Float` with scaling) for exact amounts if needed.
- **Error Handling**: Add logging or retries for failed token transfers.
- **Scalability**: For many users, consider batching `processPayouts` or using a worker canister.
- **Testing**: Simulate time passage locally by reducing `PAYOUT_INTERVAL` (e.g., to 60 seconds).

This payout canister should now work perfectly with the updated wallet canister. Let me know if you need help testing or further refinements!
Think harder
Explain ICRC-1 standard
Explore ICRC-2 standard