# Token Wallet on ICP Blockchain

```
use ic_cdk::export::candid::{CandidType, Deserialize};
use ic_cdk_macros::*;
use std::collections::HashMap;

// Define a struct for the wallet
#[derive(Clone, Debug, CandidType, Deserialize)]
struct Wallet {
    balance: u64,
}

// Define a struct for the smart contract
#[derive(Default)]
struct TokenContract {
    wallets: HashMap<String, Wallet>,
}

// Initialize the contract state
thread_local! {
    static TOKEN_CONTRACT: std::cell::RefCell<TokenContract> =
std::cell::RefCell::new(TokenContract::default());
}

// Add tokens to a wallet (for testing purposes)
#[update]
pub fn mint(address: String, amount: u64) {
    TOKEN_CONTRACT.with(|contract| {
        let mut contract = contract.borrow_mut();
        let wallet = contract.wallets.entry(address.clone()).or_insert(Wallet { balance: 0 });
        wallet.balance += amount;
    });
}

// Transfer tokens from one wallet to another
#[update]
pub fn transfer(from: String, to: String, amount: u64) -> Result<String, String> {
    TOKEN_CONTRACT.with(|contract| {
        let mut contract = contract.borrow_mut();
        let sender_wallet = contract.wallets.get_mut(&from);

        if let Some(sender) = sender_wallet {
            if sender.balance >= amount {
                sender.balance -= amount;
                let receiver_wallet = contract.wallets.entry(to.clone()).or_insert(Wallet { balance: 0 });
                receiver_wallet.balance += amount;
                return Ok(format!("Transferred {} tokens from {} to {}", amount, from, to));
            } else {
                return Err("Insufficient balance".to_string());
```

```
            }
        } else {
            return Err("Sender wallet does not exist".to_string());
        }
    })
}

// Check the balance of a wallet
#[query]
pub fn balance_of(address: String) -> u64 {
    TOKEN_CONTRACT.with(|contract| {
        let contract = contract.borrow();
        if let Some(wallet) = contract.wallets.get(&address) {
            return wallet.balance;
        }
        0
    })
}

// Unit tests
#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn test_mint() {
        mint("user1".to_string(), 100);
        assert_eq!(balance_of("user1".to_string()), 100);
    }

    #[test]
    fn test_transfer() {
        mint("user1".to_string(), 100);
        mint("user2".to_string(), 50);

        let result = transfer("user1".to_string(), "user2".to_string(), 50);
        assert_eq!(result, Ok("Transferred 50 tokens from user1 to user2".to_string()));
        assert_eq!(balance_of("user1".to_string()), 50);
        assert_eq!(balance_of("user2".to_string()), 100);
    }

    #[test]
    fn test_insufficient_balance() {
        mint("user1".to_string(), 30);
        let result = transfer("user1".to_string(), "user2".to_string(), 50);
        assert_eq!(result, Err("Insufficient balance".to_string()));
    }

    #[test]
    fn test_wallet_does_not_exist() {
        let result = transfer("unknown".to_string(), "user2".to_string(), 50);
        assert_eq!(result, Err("Sender wallet does not exist".to_string()));
```

```
    }
}

// Main canister logic entry points for ICP deployment
#[init]
pub fn init() {
    ic_cdk::println!("Token Wallet initialized");
}
```