Confronta il testo
Trova la differenza tra due file di testo

Untitled diff

Eliminare Salvare Condividere

Real-time diff
Unified diff
Collapse lines

HIGHLIGHT CHANGE
Parola Lettera

SYNTAX HIGHLIGHTING
Choose syntax

STRUMENTI
Convertire in minuscolo
Ordinare le righe
Sostituire le interruzioni di riga con spazi
Tagliare gli spazi bianchi

Confrontare e unire
Esportare su formato PDF

Editore

21 removals — 123 lines — Copia tutti

```
use borsh::{BorshDeserialize, BorshSerialize};
use solana_program::{
    account_info::{next_account_info, AccountInfo},
    entrypoint::ProgramResult,
    program::{invoke, invoke_signed},
    program_error::ProgramError,
    pubkey::Pubkey,
    rent::Rent,
    system_instruction,
    sysvar::Sysvar,
};

use crate::{Wallet, WalletInstruction, WALLET_LEN};

pub fn process_instruction(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    mut instruction_data: &[u8],
) -> ProgramResult {
    match WalletInstruction::deserialize(&mut instruction_data)? {
        WalletInstruction::Initialize => initialize(program_id, accounts),
        WalletInstruction::Deposit { amount } => deposit(program_id, accounts, amount),
        WalletInstruction::Withdraw { amount } => withdraw(program_id, accounts, amount),
    }
}

fn initialize(program_id: &Pubkey, accounts: &[AccountInfo]) -> ProgramResult {
    let account_info_iter = &mut accounts.iter();
    let wallet_info = next_account_info(account_info_iter)?;
    let vault_info = next_account_info(account_info_iter)?;
    let authority_info = next_account_info(account_info_iter)?;
    let rent_info = next_account_info(account_info_iter)?;

    let (wallet_address, wallet_seed) =
        Pubkey::find_program_address(&[&authority_info.key.to_bytes()], program_id);

    Text moved to lines 59-63
    let (vault_address, vault_seed) = Pubkey::find_program_address(
        &[&authority_info.key.to_bytes(), &"VAULT".as_bytes()],
        program_id,
    );

    let rent = Rent::from_account_info(rent_info)?;

    assert_eq!(*wallet_info.key, wallet_address);
    assert!(wallet_info.data_is_empty());

    invoke_signed(
        &system_instruction::create_account(
            &authority_info.key,
            &wallet_address,
            rent.minimum_balance(WALLET_LEN as usize),
            WALLET_LEN,
            &program_id,
        ),
        &[authority_info.clone(), wallet_info.clone()],
        &[&[&authority_info.key.to_bytes(), &[wallet_seed]]],
    )?;

    invoke_signed(
        &system_instruction::create_account(
            &authority_info.key,
            &vault_address,
            rent.minimum_balance(0),
            0,
            &program_id,
        ),
        &[authority_info.clone(), vault_info.clone()],
        &[&[
            &authority_info.key.to_bytes(),
            &"VAULT".as_bytes(),
            &[vault_seed],
        ]],
    )?;

    let wallet = Wallet {
```

43 additions — 143 lines — Copia tutti

```
use borsh::{BorshDeserialize, BorshSerialize};
use solana_program::{
    account_info::{next_account_info, AccountInfo},
    entrypoint::ProgramResult,
    program::{invoke, invoke_signed},
    program_error::ProgramError,
    pubkey::Pubkey,
    rent::Rent,
    system_instruction,
    sysvar::Sysvar,
};

use crate::{Wallet, WalletInstruction, WALLET_LEN};

pub fn process_instruction(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    mut instruction_data: &[u8],
) -> ProgramResult {
    match WalletInstruction::deserialize(&mut instruction_data)? {
        WalletInstruction::Initialize => initialize(program_id, accounts),
        WalletInstruction::Deposit { amount } => deposit(program_id, accounts, amount),
        WalletInstruction::Withdraw { amount } => withdraw(program_id, accounts, amount),
    }
}

fn initialize(program_id: &Pubkey, accounts: &[AccountInfo]) -> ProgramResult {
    let account_info_iter = &mut accounts.iter();
    let wallet_info = next_account_info(account_info_iter)?;
    let vault_info = next_account_info(account_info_iter)?;
    let authority_info = next_account_info(account_info_iter)?;
    let rent_info = next_account_info(account_info_iter)?;

    // Utilizziamo lo stesso account_info per wallet e vault
    let wallet = wallet_info.clone();
    let vault = vault_info.clone();

    // Creazione dell'account wallet
    let (wallet_address, wallet_seed) =
        Pubkey::find_program_address(&[&authority_info.key.to_bytes()], program_id);

    let rent = Rent::from_account_info(rent_info)?;

    assert_eq!(*wallet.key, wallet_address);
    assert!(wallet.data_is_empty());

    invoke_signed(
        &system_instruction::create_account(
            &authority_info.key,
            &wallet_address,
            rent.minimum_balance(WALLET_LEN as usize),
            WALLET_LEN,
            &program_id,
        ),
        &[authority_info.clone(), wallet.clone()],
        &[&[&authority_info.key.to_bytes(), &[wallet_seed]]],
    )?;

    // Creazione dell'account vault
    Text moved from lines 35-39
    let (vault_address, vault_seed) = Pubkey::find_program_address(
        &[&authority_info.key.to_bytes(), &"VAULT".as_bytes()],
        program_id,
    );

    assert_eq!(*vault.key, vault_address);
    assert!(vault.data_is_empty());

    invoke_signed(
        &system_instruction::create_account(
            &authority_info.key,
            &vault_address,
            rent.minimum_balance(0),
            0,
            &program_id,
        ),
        &[authority_info.clone(), vault.clone()],
        &[&[
            &authority_info.key.to_bytes(),
            &"VAULT".as_bytes(),
            &[vault_seed],
        ]],
    )?;

    let wallet_data = Wallet {
```

```
        authority: *authority_info.key,
        vault: vault_address,
    };
        &[&[&authority_info.key.to_bytes(), &[wallet_seed]]],
    )?;


    invoke_signed(
        &system_instruction::create_account(
            &authority_info.key,
            &vault_address,
            rent.minimum_balance(0),
            0,
            &program_id,
        ),
        &[authority_info.clone(), vault_info.clone()],
        &[&[
            &authority_info.key.to_bytes(),
            &"VAULT".as_bytes(),
            &[vault_seed],
        ]],
    )?;

    let wallet = Wallet {
        authority: *authority_info.key,
        vault: vault_address,
    };
fn withdraw(_program_id: &Pubkey, accounts: &[AccountInfo], amount: u64) -> Progra
mResult {
    let account_info_iter = &mut accounts.iter();
    let wallet_info = next_account_info(account_info_iter)?;
    let vault_info = next_account_info(account_info_iter)?;
    let authority_info = next_account_info(account_info_iter)?;
    let destination_info = next_account_info(account_info_iter)?;
    let wallet = Wallet::deserialize(&mut &(*wallet_info.data).borrow_mut()[..])?;


    assert!(authority_info.is_signer);
    assert_eq!(wallet.authority, *authority_info.key);
    assert_eq!(wallet.vault, *vault_info.key);

    if amount > **vault_info.lamports.borrow_mut() {
        return Err(ProgramError::InsufficientFunds);
    }

    **vault_info.lamports.borrow_mut() -= amount;
    **destination_info.lamports.borrow_mut() += amount;

    Ok(())
}
```

```
        authority: *authority_info.key,
        vault: vault_address,
    };
        &[&[&authority_info.key.to_bytes(), &[wallet_seed]]],
    )?;

    // Creazione dell'account vault
Text moved from lines 35-39
    let (vault_address, vault_seed) = Pubkey::find_program_address(
        &[&authority_info.key.to_bytes(), &"VAULT".as_bytes()],
        program_id,
    );

    assert_eq!(*vault.key, vault_address);
    assert!(vault.data_is_empty());

    invoke_signed(
        &system_instruction::create_account(
            &authority_info.key,
            &vault_address,
            rent.minimum_balance(0),
            0,
            &program_id,
        ),
        &[authority_info.clone(), vault.clone()],
        &[&[
            &authority_info.key.to_bytes(),
            &"VAULT".as_bytes(),
            &[vault_seed],
        ]],
    )?;

    let wallet_data = Wallet {
        authority: *authority_info.key,
        vault: vault_address,
    };
fn withdraw(_program_id: &Pubkey, accounts: &[AccountInfo], amount: u64) -> Progra
mResult {
    let account_info_iter = &mut accounts.iter();
    let wallet_info = next_account_info(account_info_iter)?;
    let vault_info = next_account_info(account_info_iter)?;
    let authority_info = next_account_info(account_info_iter)?;
    let destination_info = next_account_info(account_info_iter)?;

    // Utilizziamo lo stesso account_info per wallet e vault
    let wallet = wallet_info.clone();
    let vault = vault_info.clone();

    let wallet_data = Wallet::deserialize(&mut &(*wallet.data).borrow_mut()[..])?;

    assert!(authority_info.is_signer);
    assert_eq!(wallet_data.authority, *authority_info.key);
    assert_eq!(wallet_data.vault, *vault.key);

    if amount > **vault.lamports.borrow_mut() {
        return Err(ProgramError::InsufficientFunds);
    }

    **vault.lamports.borrow_mut() -= amount;
    **destination_info.lamports.borrow_mut() += amount;

    Ok(())
}
```

Diff salvati

Your saved diffs will appear here.

Testo originale Open file

```
    assert_eq!(wallet.authority, *authority_info.key);
    assert_eq!(wallet.vault, *vault_info.key);

    if amount > **vault_info.lamports.borrow_mut() {
        return Err(ProgramError::InsufficientFunds);
    }

    **vault_info.lamports.borrow_mut() -= amount;
    **destination_info.lamports.borrow_mut() += amount;

    Ok(())
}
```

Testo modificato Open file

```
    assert_eq!(wallet_data.authority, *authority_info.key);
    assert_eq!(wallet_data.vault, *vault.key);

    if amount > **vault.lamports.borrow_mut() {
        return Err(ProgramError::InsufficientFunds);
    }

    **vault.lamports.borrow_mut() -= amount;
    **destination_info.lamports.borrow_mut() += amount;

    Ok(())
}
```


Trovare la differenza