Projecte de gestió de préstecs de l'Institut

Taula de continguts

1. Descripció i requisits

En aquest projecte de *blockchain*, l'objectiu principal es gestionar el cicle de vida d'un préstec de material. Per dur a terme aquesta tasca, es crea un `smart contract` que implementa una petita "màquina d'estats".

El préstec del material dependrà de dos rols: l'Administrador (l'institut) i l'Usuari (l'alumne).

L'algorisme es basa en la gestió d'un estat (qui té què, què està disponible, etc.) que només pot ser modificat a través de funcions (*endpoints*) amb unes regles predefinides.

Estructura de dades:

- `PrestecInfo`: estructura de dades que conté la informació essencial d'un préstec actiu: quin tipus d'article s'ha deixat en préstec (`tipus_item`) i quan s'ha de retornar (`data_retorn`)
- `owner`: adreça de l'administrador, que és qui poc executar funcions privilegiades
- `inventari_total` i `inventari_disponible`: permet saber què hi ha disponible per al préstec i què no
- `prestecs`: és el registre dels préstecs actius i aprovats. Associa l'adreça d'un usuari amb la seva informació de préstec. Si un usuari no hi té dades és que no té cap préstec actiu
- `solicituds_pendents`: guarda les peticions que han fet els alumnes però que l'administrador encara no ha aprovat. Associa l'adreça de l'usuari amb el tipus d'article que vol

Lògica de les Funcions (`Endpoints`):

Cada funció representa una acció que es pot realitzar sobre el contracte.

- `init()`
  - Propòsit: inicialitzar el contracte quan es fa el desplegament
  - Objectius:
    - Obtenir l'adreça de la *wallet* que fa la transacció de desplegament per saber qui n'és el propietari (*owner*)
- `afegirTipusItem()` (Només Admin)
  - Propòsit: permetre a l'institut donar d'alta nou material a l'inventari.

    - Objectius:
        - Només pot afegir material l'*owner*
        - Assegurar que hi ha material disponible
- `solicitarPrestec()` (Per a Alumnes)
    - Propòsit: permetre a un alumne iniciar el procés de préstec.
    - Objectius:
        - Obtenir l'adreça de l'alumen que sol·licita el material
        - Comprovar que l'alumne no tingui un préstec actiu (només es permet un préstec per persona)
        - Comprovar que l'alumne no tingui una sol·licitud pendent
        - Comprovar que l'alumne demani un tipus d'article que l'institut realment gestiona
        - Actualitzar l'estat de l'alumne
- `aprovarPrestec()` (Només Admin)
    - Propòsit: oficialitzar un préstec que estava pendent.
    - Objectius:
        - Comprovar que només hi té accés l'*owner*
        - Validar que l'alumne té una sol·licitud pendent
        - Comprovar que hi ha unitats disponibles pel préstec
        - Actualitzar l'estat de l'alumne, esborrar la sol·licitud i restar unitats disponibles, calcular la data de retorn i crear la fitxa del préstec
- `confirmarRetorn()` (Només Admin)
    - Propòsit: tancar el cicle del préstec quan l'article es retorna físicament
    - Objectius:
        - Comprovar que només hi tingui accés l'*owner*
        - Validar que l'alumne tingui un préstec actiu
        - Obtenir la informació del préstec per saber quin tipus d'article és
        - Esborrar el préstec de la llista d'actius (prestecs.clear())
        - Incrementar en 1 el comptador d'unitats disponibles d'aquell article

Funcions de Vista (`#[view]`):

- `consultar_prestec()`:
    - Propòsit: retorna la informació del préstec actiu d'una adreça específica
- `consultar_disponibles()`:
    - Propòsit: retorna la quantitat d'unitats disponibles d'un tipus d'article
- `consultar_solicitud_pendent()`:
    - Propòsit: retorna el tipus d'article que ha sol·licitat un usuari i que està pendent d'aprovació

2. Algorisme

Algorisme de l'smart contract

```
#![no_std]

multiversx_sc::imports!();
multiversx_sc::derive_imports!();

/// Aquest contracte gestiona el préstec de material informàtic per a un institut,
/// separant les responsabilitats entre un rol d'Administrador (`owner`) i els Usuaris
(alumnes).
/// El cicle de vida d'un préstec és: Sol·licitud -> Aprovació -> Retorn Confirmat.

// --- Estructures de Dades ---

/// Conté la informació essencial d'un préstec que ja ha estat aprovat i està actiu.
#[type_abi]
#[derive(TopEncode, TopDecode)]
pub struct PrestecInfo<M: ManagedTypeApi> {
    /// El tipus d'article prestat (p.ex., "portatil", "teclat").
    pub tipus_item: ManagedBuffer<M>,
    /// La data límit de retorn, guardada com a timestamp de bloc (en segons).
    pub data_retorn: u64,
}

/// Defineix la interfície del contracte amb tots els seus endpoints i vistes
públiques.
#[multiversx_sc::contract]
pub trait PrestecsInstitut {
    /// Funció d'inicialització. S'executa només un cop en desplegar el contracte.
    #[init]
    fn init(&self) {
        // Obtenim l'adreça de la wallet que desplega el contracte.
        let caller = self.blockchain().get_caller();
        // La guardem com a propietària (administrador) del contracte.
        // Això garanteix que només qui crea el contracte en té el control inicial.
        self.owner().set(caller);
    }

    // ===================================================================
    // === Endpoints d'Administrador (requereixen ser el 'owner') ===
    // ===================================================================

    /// Endpoint privat per afegir un nou tipus d'article a l'inventari o actualitzar-
ne la quantitat.
    /// Només pot ser cridat per l'administrador (`owner`).
    #[only_owner]
    #[endpoint(afegirTipusItem)]
    fn afegir_tipus_item(&self, tipus_item: ManagedBuffer, quantitat_inicial: usize) {
```

```
        // Validació: La quantitat inicial ha de ser superior a zero per evitar estats
invàlids.
        require!(quantitat_inicial > 0, "La quantitat ha de ser més gran que 0");

        // Donem d'alta el nou tipus d'article, establint el seu estoc total i
disponible.
        self.inventari_total(&tipus_item).set(quantitat_inicial);
        self.inventari_disponible(&tipus_item).set(quantitat_inicial);
    }

    /// Endpoint privat que permet a l'administrador aprovar una sol·licitud de
préstec pendent.
    #[only_owner]
    #[endpoint(aprovarPrestec)]
    fn aprovar_prestec(&self, adreca_usuari: ManagedAddress) {
        // Validació 1: Comprovem que l'usuari realment té una sol·licitud pendent.
        require!(!self.solicituds_pendents(&adreca_usuari).is_empty(), "Aquest usuari
no té cap sol·licitud pendent.");

        let tipus_item = self.solicituds_pendents(&adreca_usuari).get();

        // Validació 2: Comprovem que hi ha unitats disponibles en el moment de
l'aprovació.
        let mut unitats_disponibles = self.inventari_disponible(&tipus_item).get();
        require!(unitats_disponibles > 0, "No queden unitats disponibles d'aquest
article.");

        // --- Accions (Transició d'Estat: de 'Pendent' a 'Actiu') ---

        // 1. Eliminem la sol·licitud de la llista de pendents.
        self.solicituds_pendents(&adreca_usuari).clear();

        // 2. Actualitzem l'inventari restant una unitat.
        unitats_disponibles -= 1;
        self.inventari_disponible(&tipus_item).set(unitats_disponibles);

        // 3. Calculem la data de retorn (p.ex., 15 dies des d'ara).
        let periode_prestec_segons = 15 * 24 * 60 * 60; // 15 dies en segons
        let data_retorn = self.blockchain().get_block_timestamp() +
periode_prestec_segons;

        // 4. Creem la "fitxa" del préstec i la guardem al registre de préstecs
actius.
        let info_prestec = PrestecInfo { tipus_item, data_retorn };
        self.prestecs(&adreca_usuari).set(info_prestec);
    }

    /// Endpoint privat que permet a l'administrador confirmar la recepció d'un
article retornat.
    #[only_owner]
    #[endpoint(confirmarRetorn)]
```

```
    fn confirmar_retorn(&self, adreca_usuari: ManagedAddress) {
        // Validació: Comprovem que l'usuari tenia un préstec actiu per retornar.
        require!(!self.prestecs(&adreca_usuari).is_empty(), "Aquest usuari no té cap
article en préstec per retornar.");

        // Obtenim la informació del préstec per saber quin tipus d'article hem de
reincorporar a l'inventari.
        let info_prestec = self.prestecs(&adreca_usuari).get();
        let tipus_item_retornat = info_prestec.tipus_item;

        // --- Accions (Transició d'Estat: de 'Actiu' a 'Retornat') ---

        // 1. Eliminem el préstec del registre d'actius.
        self.prestecs(&adreca_usuari).clear();

        // 2. Incrementem les unitats disponibles de l'article retornat.
        let mut unitats_disponibles =
self.inventari_disponible(&tipus_item_retornat).get();
        unitats_disponibles += 1;
        self.inventari_disponible(&tipus_item_retornat).set(unitats_disponibles);
    }

    // ====================================================================
    // === Endpoints d'Usuari (Alumnes) ===
    // ====================================================================

    /// Endpoint públic que permet a un usuari (alumne) sol·licitar un article.
    /// La sol·licitud queda en estat pendent fins que l'administrador l'aprovi.
    #[endpoint(solicitarPrestec)]
    fn solicitar_prestec(&self, tipus_item: ManagedBuffer) {
        // Obtenim l'adreça de qui fa la crida.
        let caller = self.blockchain().get_caller();

        // --- Validacions (Regles de Negoci) ---
        // 1. L'usuari no pot demanar un article si ja en té un altre en préstec
actiu.
        require!(self.prestecs(&caller).is_empty(), "Ja tens un article en préstec
actiu.");
        // 2. L'usuari no pot fer una nova sol·licitud si ja en té una de pendent.
        require!(self.solicituds_pendents(&caller).is_empty(), "Ja tens una
sol·licitud pendent.");
        // 3. El tipus d'article sol·licitat ha d'existir a l'inventari.
        require!(!self.inventari_total(&tipus_item).is_empty(), "Aquest tipus
d'article no existeix.");

        // Si totes les validacions són correctes, registrem la sol·licitud.
        self.solicituds_pendents(&caller).set(tipus_item);
    }


    // ====================================================================
```

```
    // === Vistes (Consultes de Només Lectura) ===
    // ================================================================

    /// Retorna la informació del préstec actiu d'una adreça específica.
    #[view(consultarPrestec)]
    fn consultar_prestec(&self, adreca: ManagedAddress) ->
OptionalValue<PrestecInfo<Self::Api>> {
        let prestec = self.prestecs(&adreca);
        if prestec.is_empty() {
            OptionalValue::None // Retorna 'None' si l'usuari no té cap préstec actiu.
        } else {
            OptionalValue::Some(prestec.get()) // Retorna la informació del préstec si
en té.
        }
    }

    /// Retorna la quantitat d'unitats disponibles d'un tipus d'article.
    #[view(consultarDisponibles)]
    fn consultar_disponibles(&self, tipus_item: ManagedBuffer) -> usize {
        self.inventari_disponible(&tipus_item).get()
    }

    /// Retorna el tipus d'article que ha sol·licitat un usuari i que està pendent
d'aprovació.
    #[view(consultarSolicitudPendent)]
    fn consultar_solicitud_pendent(&self, adreca: ManagedAddress) ->
OptionalValue<ManagedBuffer> {
        let solicitud = self.solicituds_pendents(&adreca);
        if solicitud.is_empty() {
            OptionalValue::None
        } else {
            OptionalValue::Some(solicitud.get())
        }
    }


    // --- Storage Mappers ---
    // Aquesta secció defineix les "variables" o "taules" on el contracte
    // emmagatzema permanentment el seu estat a la blockchain.

    /// Guarda l'adreça de l'administrador del contracte.
    #[storage_mapper("owner")]
    fn owner(&self) -> SingleValueMapper<ManagedAddress>;

    /// Mapa que associa un tipus d'item amb la seva quantitat total registrada.
    #[storage_mapper("inventariTotal")]
    fn inventari_total(&self, tipus_item: &ManagedBuffer) -> SingleValueMapper<usize>;

    /// Mapa que associa un tipus d'item amb la quantitat actualment disponible per a
préstec.
    #[storage_mapper("inventariDisponible")]
```

```
    fn inventari_disponible(&self, tipus_item: &ManagedBuffer) ->
SingleValueMapper<usize>;

    /// Mapa que associa l'adreça d'un usuari amb la informació del seu préstec ACTIU.
    #[storage_mapper("prestecs")]
    fn prestecs(&self, adreca_usuari: &ManagedAddress) ->
SingleValueMapper<PrestecInfo<Self::Api>>;

    /// Mapa que associa l'adreça d'un usuari amb la seva sol·licitud de préstec
PENDENT.
    #[storage_mapper("solicitudsPendents")]
    fn solicituds_pendents(&self, adreca_usuari: &ManagedAddress) ->
SingleValueMapper<ManagedBuffer>;
}
```

3. Script Client

Exemple de crida a una funció del contracte:

[source, bash

```
# Exemple per afegir 20 portàtils
mxpy contract call erd1qqqqqqqqqqqqqpgquuqezkwswma3x9tegn548mmxa5zaa3vd6zfsq52rcd \
  --recall-nonce \
  --pem ./wallets/admin.pem \
  --gas-limit 10000000 \
  --proxy https://devnet-gateway.multiversx.com \
  --chain D \
  --function "afegirTipusItem" \
  --arguments str:portatil 20 \
  --send
```

Script de la part client:

```
#!/bin/bash

################################################################################
# CONFIGURACIÓ - MODIFICA AQUESTES VARIABLES
################################################################################

# Adreça del Smart Contract un cop desplegat
CONTRACT_ADDRESS="erd1qqqqqqqqqqqqqpgquuqezkwswma3x9tegn548mmxa5zaa3vd6zfsq52rcd"

# Camí al fitxer PEM de l'administrador (institut)
PEM_ADMIN="./wallet.pem"
# Adreça de l'administrador
ADDRESS_ADMIN=$(mxpy wallet pem-address $PEM_ADMIN)
```

```
# Camí al fitxer PEM d'un alumne d'exemple
PEM_ALUMNE="./wallets/alumne1.pem"
# Adreça de l'alumne
ADDRESS_ALUMNE=$(mxpy wallet pem-address $PEM_ALUMNE)

# Configuració de la xarxa (devnet, testnet o mainnet)
PROXY="https://devnet-gateway.multiversx.com"
CHAIN_ID="D"

# Límits de gas
GAS_LIMIT=10000000

###############################################################################
# FUNCIONS AUXILIARS
###############################################################################

# Funció per executar una transacció (crida a un endpoint)
# Ús: executar_tx <funcio> <pem_file> [arguments...]
executar_tx() {
    local funcio=$1
    local pem_file=$2
    shift 2
    local args=("$@")

    echo "Executant transacció..."
    mxpy --verbose contract call ${CONTRACT_ADDRESS} --recall-nonce --pem=${pem_file}
\
        --gas-limit=${GAS_LIMIT} --proxy=${PROXY} --chain=${CHAIN_ID} \
        --function="${funcio}" --arguments "${args[@]}" --send || echo "ERROR: La
transacció ha fallat."
}

# Funció per executar una consulta (crida a una vista)
# Ús: executar_query <funcio> [arguments...]
executar_query() {
    local funcio=$1
    shift
    local args=("$@")

    echo "Executant consulta..."
    mxpy --verbose contract query ${CONTRACT_ADDRESS} --proxy=${PROXY} \
        --function="${funcio}" --arguments "${args[@]}"
}

###############################################################################
# MENÚ PRINCIPAL
###############################################################################

while true; do
    echo "================================================"
    echo "    SISTEMA DE GESTIÓ DE PRÉSTECS DE L'INSTITUT    "
```

```
    echo "================================================"
    echo
    echo "--- MENÚ ADMINISTRADOR ---"
    echo "  1. Afegir tipus d'item a l'inventari"
    echo "  2. Aprovar un préstec pendent"
    echo "  3. Confirmar el retorn d'un article"
    echo
    echo "--- MENÚ ALUMNE ---"
    echo "  4. Sol·licitar un préstec"
    echo
    echo "--- CONSULTES GENERALS ---"
    echo "  5. Consultar inventari disponible"
    echo "  6. Consultar estat d'un préstec actiu"
    echo "  7. Consultar una sol·licitud pendent"
    echo
    echo "  0. Sortir"
    echo "------------------------------------------------"
    read -p "Tria una opció: " opcio

    case $opcio in
        1)
            read -p "Introdueix el tipus d'item (ex: portatil): " tipus
            read -p "Introdueix la quantitat inicial: " quantitat
            executar_tx "afegirTipusItem" "${PEM_ADMIN}" "str:${tipus}" "${quantitat}"
            ;;
        2)
            read -p "Introdueix l'adreça de l'alumne a aprovar (erd1...): "
adreca_alumne
            executar_tx "aprovarPrestec" "${PEM_ADMIN}" "addr:${adreca_alumne}"
            ;;
        3)
            read -p "Introdueix l'adreça de l'alumne que retorna (erd1...): "
adreca_alumne
            executar_tx "confirmarRetorn" "${PEM_ADMIN}" "addr:${adreca_alumne}"
            ;;
        4)
            read -p "Introdueix el tipus d'item a sol·licitar (ex: portatil): " tipus
            echo "Es farà la sol·licitud com a ALUMNE: ${ADDRESS_ALUMNE}"
            executar_tx "solicitarPrestec" "${PEM_ALUMNE}" "str:${tipus}"
            ;;
        5)
            read -p "De quin tipus d'item vols consultar la disponibilitat? " tipus
            executar_query "consultarDisponibles" "str:${tipus}"
            ;;
        6)
            read -p "De quina adreça vols consultar el préstec actiu? " adreca
            executar_query "consultarPrestec" "addr:${adreca}"
            ;;
        7)
            read -p "De quina adreça vols consultar la sol·licitud pendent? " adreca
            executar_query "consultarSolicitudPendent" "addr:${adreca}"
```

```
            ;;
        0)
            echo "Sortint..."
            break
            ;;
        *)
            echo "Opció invàlida. Torna a intentar-ho."
            ;;
    esac
    echo
    read -p "Prem [Enter] per continuar..."
    clear
done
```