Agenzia nazionale per le nuove tecnologie, l'energia e lo sviluppo economico sostenibile

Ministero della Transizione Ecologica

Ricerca di Sistema elettrico

Sviluppo di una piattaforma per comunità energetiche basata su blockchain

Marco Varini, Mario Squillace, Francesco Faenzi

Report RdS/PTR(2021)/038

Sviluppo di una piattaforma per comunità energetiche basata su blockchain

Marco Varini, Mario Squillace, Francesco Faenzi

Giugno 2021

Report Ricerca di Sistema Elettrico

Accordo di Programma Ministero della Transizione Ecologica- ENEA
Piano Triennale di Realizzazione 2019-2021 - II annualità
Obiettivo: Tecnologie
Progetto: Tecnologie per la penetrazione efficiente del vettore elettrico negli usi finali
Work package: Local Energy District
Linea di attività: LA1.47 - Progettazione piattaforma LEC, implementazione infrastruttura e servizi orizzontali
Responsabile del Progetto: Claudia Meloni, ENEA
Responsabile del Work package: Claudia Meloni, ENEA

Il presente documento descrive le attività di ricerca svolte all'interno del Contratto "Sviluppo di una piattaforma per comunità energetiche basata su blockchain"
Responsabile Unico del Procedimento ENEA: Gianluca D'Agosta
Responsabile del Contratto per il Contraente Soft Strategy SpA: Francesco Faenzi, Marco Varini

Indice

Sommario

Il documento descrive le attività progettuali svolte per la realizzazione di una piattaforma per comunità energetiche basata su blockchain.

Uno degli elementi fondamentali per la creazione della comunità è la realizzazione di un sistema che possa garantire la trasparenza e l'affidabilità di tutte le informazioni che circolano nell'ambito della comunità stessa, soprattutto di quelle che generano valore sia economico che sociale. Tale sistema è basato sulla tecnologia blockchain che, attraverso il meccanismo degli smart contract, consente di implementare in maniera sicura e trasparente le logiche e le politiche di premialità e penalità nell'ambito dei consumi energetici, ma anche supportare ed essere il meccanismo per la gestione, vendita e acquisto dei servizi e dei beni sociali a disposizione della comunità. La blockchain è quindi il componente della piattaforma su cui si agganciano i dati di consumo energetici con un meccanismo per una economia locale basato su token.

La soluzione realizzata permette agli utenti delle comunità energetiche di gestire i token presenti nel proprio "wallet" in modo semplice, intuitivo e trasparente, permettendone quindi lo scambio all'interno della comunità per usufruire di determinati servizi messi a disposizione.

1 Introduzione

L'obiettivo di questo progetto è quello di realizzare un sistema e relativi sotto-moduli che:

- permetta la gestione di uno o più token di Comunità (separando le differenti Comunità che afferiranno alla medesima piattaforma), seguendo delle specifiche politiche di comunità che sono state definite.
- permetta l'implementazione trasparente e sicura dei modelli e delle logiche di guadagno già definiti, e perdita e scambio di token fra gli utenti e la piattaforma, legate sia ai servizi elettrici (consumo, produzione ma anche messa a disposizione di flessibilità energetica) che sociali (fruizione o fornitura dei servizi);
- permetta di controllare e di modificare, attraverso dei parametri, le stesse logiche all'interno dei singoli smart contract da parte di un operatore (non per forza esperto programmatore), in base alle necessità di modifica del modello della comunità;
- sia connessa ad una o più Dapp per la gestione/visualizzazione delle transazioni e mostri le informazioni relative attraverso delle interfacce grafiche e dashboard;
- sia connessa a sorgenti dati interne, come i database relazionali ma anche eventualmente di altro genere, che agiscano come applicazioni "oracoli" ad uso della blockchain stessa e degli smart contract in essa sviluppati.

Gli smart contract devono quindi andare a implementare le logiche e i modelli di scambio dei token, consentendo, al gestore stesso della comunità, di modificare alcuni parametri interni agli smart contract, in modo da modificare il risultato della transazione stessa in funzione delle proprie necessità.
Tutti gli utenti che si registrano alla comunità devono poter disporre di un portafogli ("wallet") che mantenga il bilancio dei token in possesso, e deve essere prevista la creazione di una Dapp che permetta agli utenti di interagire con la piattaforma, avendo accesso al proprio saldo in token, permettendo il "pagamento" o la "riscossione" di token. Rispetto alle applicazioni già sviluppate, l'obiettivo è quello di rendere "semplice" l'utilizzo dei token, ad esempio attraverso la lettura di un QR-Code installato presso un esercizio commerciale che partecipi all'iniziativa e che, in cambio di token, dia accesso ad uno sconto o a dei servizi differenti.)

2 Descrizione delle attività svolte e risultati

2.1 Soluzione proposta

La soluzione proposta, rappresentata dal blueprint di Figura 1, indirizza gli obiettivi progettuali di permettere ai diversi attori di poter usufruire della soluzione.

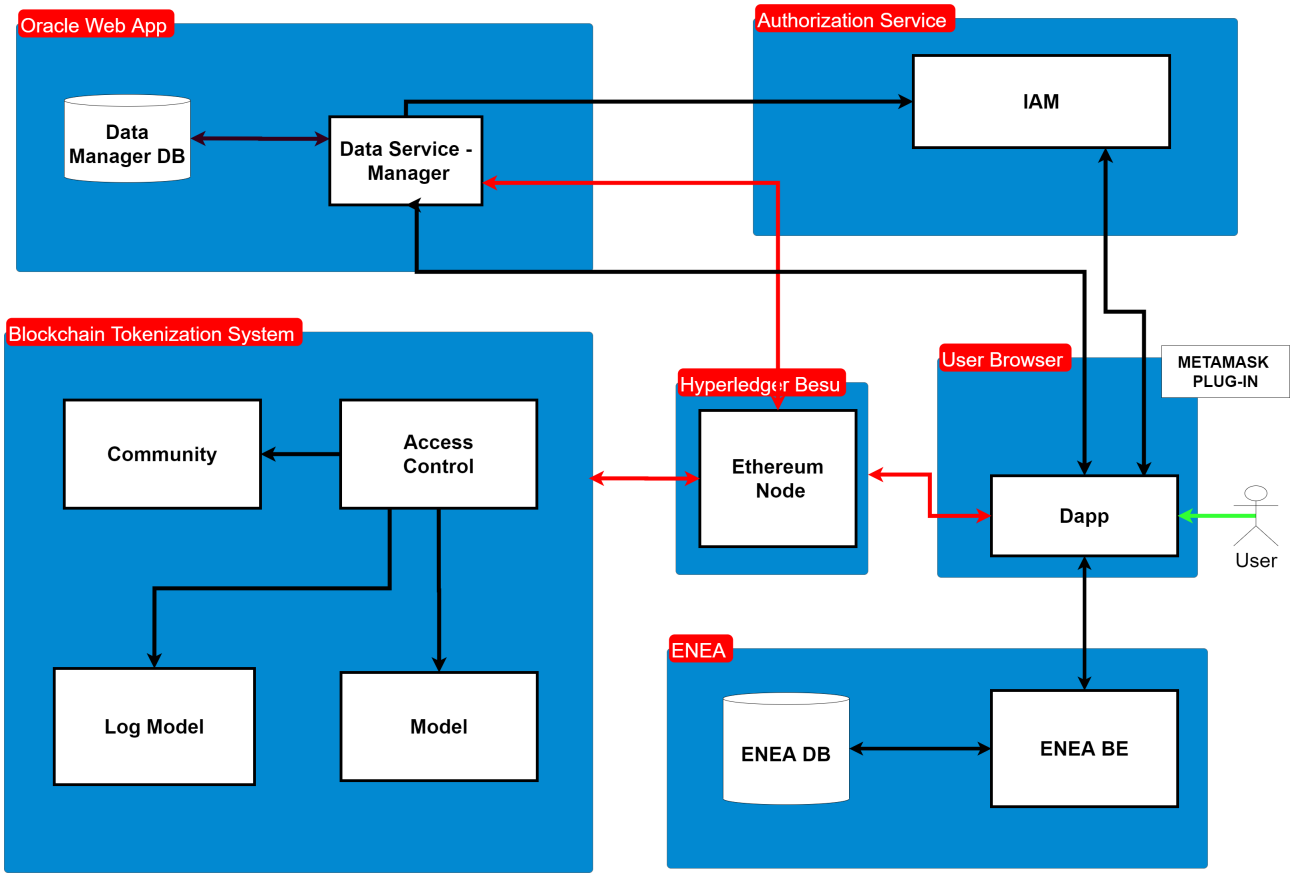


Figura 1 Blueprint architetturale della soluzione

Di seguito si presentano gli attori coinvolti e le operazioni a loro riferite.

Platform Admin:

- Crea Comunità
- Crea utenti amministratori
- Associa i ruoli di amministratore su blockchain
- Associa il ruolo token-holder agli utenti su Identity Provider

Oracle Admin:

- Configura i modelli su Data manager DB
- Configura i parametri dei modelli su Blockchain

Community Admin:

- Assegna ruolo token-holder su Blockchain
- Associa utenti alla comunità
- Gestisce fee in fase di assegnazione token
- Gestisce incentivi in fase di scambio token
- Blocca le operazioni degli utenti sui token

- Gestisce token bonus di benvenuto

Token Holder:
- Scambia token
- Sign-up
- Richiede takeout

Scheduler Admin:
- Assegna token agli utenti

2.2 Smart Contract

Gli Smart Contract sono rappresentati in Figura 1 all'interno del blocco Blockchain Tokeniztion System che rappresenta lo strato della Business Logic della gestione dei token e dei relativi processi della piattaforma.

2.2.1 Community Smart Contract

Il Community Smart Contract si occupa di:
- Gestire i token ERC-20 di comunità
- Gestire le comunità
- Gestire gli account ethereum degli utenti per ogni comunità
- Gestire le fee in fase di assegnazione token
- Gestire gli incentivi in fase di trasferimento token
- Assegnare i token di bonus benvenuto quando un utente viene assegnato ad una comunità
- Può pagare una quota di servizi agli utenti di comunità

Nel Community Smart Contract, in fase di start-up, viene indicato un wallet di comunità. Questo, potrà essere accreditato o addebitato dai token di comunità. Se, ad esempio, una assegnazione token è soggetta ad una fee (vedere capitolo 2.2.1.3), tale fee viene accreditata al community wallet, viceversa, se la spesa per un servizio prevede un incentivo, tale incentivo (vedere capitolo 2.2.1.4) può essere pagato dal community wallet.

Tali funzionalità vengono abilitate implementando l'interfaccia IERC-20, IERC-1203, IFee, IIncentive, IUserCommunity, ICommunity e ITokenBonus.

2.2.1.1 IERC-20 Smart Contract

L'IERC-20 è l'interfaccia standard ERC-20 per interagire con i diversi Community Token.
Ogni utente appartenente ad una comunità potrà scambiare token solo con gli utenti appartenenti alla stessa comunità.

Il token indicato in precedenza (WP230-005-v1-Modello per calcolo virtuosismo energetico e token) come Token Regulations è stato eliminato in quanto non garantisce gli opportuni meccanismi per forzare l'applicazione di fee alle transazioni, questo perché il precedente Community Token esponeva le proprie funzionalità all'interno del network e, quindi, permetteva lo scambio dei token bypassando il Token Regulations. Per tale motivo le funzionalità del Token Regulation sono state inserite nel Community Token.
In Figura 2 sono rappresentate le funzionalità dello Smart Contract implementate per lo sviluppo del token.

```
contract ERC20 {
    function totalSupply() public view returns (uint256);
    function balanceOf(address _owner) public view returns (uint256);
    function transfer(address _to, uint256 _value) public returns (bool);
    function approve(address _spender, uint256 _value) public returns (bool);
    function allowance(address _owner, address _spender) public view returns (uint256);
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool);

    event Transfer(address indexed _from, address indexed _to, uint256 _value);
    event Approval(address indexed _owner, address indexed _spender, uint256 _value);
}
```

Figura 2 Token ERC20 – descrizione dello Smart Contract

2.2.1.2 IERC-1203

L'IERC-1203 permette di gestire all'interno di un singolo token un numero arbitrario di token ERC-20 ognuno identificato dalla propria classe. Nel caso specifico di progetto, la classe è l'identificativo di comunità. L'utilizzo di questo standard permette la piena gestione dei token di comunità risolvendo la complessità di dover deployare su Blockchain uno Smart Contract dedicato per ogni singolo token di comunità. In Figura 3 sono rappresentate le funzionalità dello Smart Contract implementate per lo sviluppo dei token.

```
contract ERC1203 is ERC20 {
    function totalSupply(uint256 _class) public view returns (uint256);
    function balanceOf(address _owner, uint256 _class) public view returns (uint256);
    function transfer(address _to, uint256 _class, uint256 _value) public returns (bool);
    function approve(address _spender, uint256 _class, uint256 _value) public returns (bool);
    function allowance(address _owner, address _spender, uint256 _class) public view returns (uint256);
    function transferFrom(address _from, address _to, uint256 _class, uint256 _value) public returns (bool);

    function fullyDilutedTotalSupply() public view returns (uint256);
    function fullyDilutedBalanceOf(address _owner) public view returns (uint256);
    function fullyDilutedAllowance(address _owner, address _spender) public view returns (uint256);
    function convert(uint256 _fromClass, uint256 _toClass, uint256 _value) public returns (bool);

    event Transfer(address indexed _from, address indexed _to, uint256 _class, uint256 _value);
    event Approval(address indexed _owner, address indexed _spender, uint256 _class, uint256 _value);
    event Convert(uint256 indexed _fromClass, uint256 indexed _toClass, uint256 _value);
}
```

Figura 3 Token ERC1203 - descrizione delle Smart Contract

2.2.1.3 IFee

L'IFee è l'interfaccia che permette di impostare, per ogni token di comunità, una fee in fase di assegnazione token.

La fee può essere a valore fisso o a percentuale:

- Valore fisso: lo smart contract sottrarrà il valore di fee al totale dell'assegnazione.
  Ad esempio: per l'assegnazione pari ad 1 token per la comunità 001 la fee è pari a 0,1 per la tipologia "FIXED", il totale token assegnato al Token Holder sarà pari a 0,9.
- Percentuale: lo smart contract sottrarrà la percentuale indicata nel valore della fee.
  Ad esempio: per l'assegnazione pari ad 1 token per la comunità 001 la fee è pari al 20% per la tipologia "PERCENTAGE", Il totale token assegnato al Token Holder sarà pari a 0.8.

In Figura 4 sono rappresentate le interfacce implementate per lo sviluppo delle fee.

```
interface IFee {
    /*
     * @dev set fee used before token mint
     * @param {_communityID}
     * @param {_feeId} fee identifier
     * @param {_fee} fee amount
     * @param {_type} fee's type.
     */
    function setFee(uint256 _communityID, uint8 _feeId, uint256 _fee, uint256 _type) external;
}
```

Figura 4 IFee – descrizione delle interfacce

2.2.1.4 IIncentive

Gli incentivi vengono gestiti dall'interfaccia IIncentive.
Ogni comunità avrà degli specifici incentivi per una serie di servizi.

Quando un utente decide di scambiare i propri token per un determinato servizio, lo Smart Contract, prima di trasferire i token, leggerà il valore di incentivo e sottrarrà tale valore dai token da spendere.
L'incentivo può essere a valore fisso o a percentuale.

- Valore fisso: lo smart contract sottrarrà il valore di incentivo al totale.
  Ad esempio: per il servizio con id 001 per la comunità 001 l'incentivo è di 0,1 token di tipologia "FIXED".
  Nel momento in cui l'utente scambierà 1 token per il servizio 001, la sua spesa totale sarà di 0,9 token (i token mancanti verranno emessi dalla comunità stessa per coprire la spesa totale del servizio).
- Percentuale: lo smart contract sottrarrà la percentuale indicata nel valore di incentivo.
  Ad esempio: per il servizio con id 002 per la comunità 001 l'incentivo è di 10 di tipologia "PERCENTAGE".

Nel momento in cui l'utente scambierà 1 token per il servizio 002, la sua spesa totale sarà di 0,9 token (i token mancanti verranno emessi dalla comunità stessa per coprire la spesa totale del servizio).
In Figura 5 sono rappresentate le interfacce implementate per lo sviluppo degli incentivi.

```
interface IIncentive {

    function setIncentive(uint8 communityId, uint8 serviceId, uint8 incentive, unit8 type) external returns (bool);
}
```

Figura 5 IIncentive - descrizione delle interfacce

2.2.1.5 IUserCommunity

L' IUserCommunity è l'interfaccia (rappresentata in Figura 6) che permette di aggiungere o rimuovere gli utenti da una comunità.

```
interface IUserCommunity {

    function addAccountToCommunity(uint8 communityId, aaddress account) external returns (bool);

    function removeAccountToCommunity(uint8 communityId, address account) external returns (bool);
}
```

Figura 6 IUserCommunity - descrizione delle interfacce

2.2.1.6 ICommunity

L'ICommunity è l'interfaccia (rappresentata in Figura 7) che permette di aggiungere o rimuovere comunità al sistema. Sarà possibile eseguire operazioni sui token di comunità solo dopo che questa verrà inserita nel sistema.

```
interface ICommunity {

    public addCommunity(uint8 communityId) external returns (bool);

    public removeCommunity(uint8 communityId) external returns (bool);
}
```

Figura 7 ICommunity - descrizione delle interfacce

2.2.1.7 ITokenBonus

L'ITokenBonus è l'interfaccia (rappresentata in Figura 8) che permette di gestire l'ammontare di token di benvenuto da assegnare ad un Token Holder quando viene aggiunto ad una comunità.

```
interface ITokenBonus{

    function setBonus(uint8 communityId, uint8 bonus) returns (bool);
}
```

Figura 8 ITokenBonus - descrizione delle interfacce

2.2.2 Access Control Smart Contract IAccessControl

L'interfaccia IaccessControl (rappresentata in Figura 9) permette di assegnare e revocare i ruoli agli utenti della piattaforma.
Ogni funzionalità all'interno della soluzione, prima di essere eseguita, deve essere autorizzata tramite chiamata allo Smart Contract che ne implementa l'interfaccia.
L'Access Control è scalabile rispetto ai ruoli, questi, non sono indicati nella sua implementazione.
I ruoli vengono definiti all'esterno della blockchain e tramite Access Manager viene assegnato un ruolo ad ogni utente.
Nella soluzione proposta i ruoli vengono mappati 1:1 tra IAM e Access Manager. Prendiamo ad esempio l'utente X con ruolo TOKEN_HOLDER.
Come visto nel paragrafo 2.3.1.1 il Community Admin tramite Interfaccia dello IAM assegna il ruolo TOKEN_HOLDER all'utente. In questo modo l'utente sarà abilitato a chiamare le API esposte dallo IAM e dal Data Service Manager autorizzate per il ruolo TOKEN_HOLDER.
L'utente, a questo punto, non è ancora abilitato ad operare come TOKEN_HOLDER verso le funzionalità offerta dalla blockchain per il suo ruolo.
Il Community Admin, collegandosi alla DAPP, selezionerà l'utente e da una combo box selezionarà il ruolo TOKEN_ADMIN. La selezione da una combo box, forza una mappatura 1:1 tra i ruoli dello IAM e la DAPP.
Sulla Dapp, i ruoli vengono rappresentati come una mappa chiave valore, dove la chiave è il ruolo e il valore la rappresentazione byte32 del ruolo:

Es. "TOKEN_HOLDER" => sha3("TOKEN_HOLDER")

Cliccando su assegna ruolo, viene invocata la funzionalità dell'Access Control grantRole che assegna all'account ethereum dell'utente X il ruolo sha3("TOKEN_HOLDER") vedere paragrafo 2.3.1.1 per il flusso di dettaglio.

```
interface IAccessControl {

    function grantRole(bytes32 role, address account) public external returns (bool);

    function revokeRole(bytes32 role, address account) public external return (bool);
}
```

Figura 9 IAccess Control - descrizione delle interfacce

2.2.3 LogModel Smart Contract

ILogModel

L'interfaccia ILogModel (rappresentata in Figura 10) permette di loggare all'interno della blockchain un'assegnazione di token per un utente. Si è scelto di utilizzare un sistema di logging per i seguenti motivi:

- Loggare un'operazione è molto meno dispendioso in termini di consumo di gas rispetto al salvataggio dei singoli valori nello State della blockchain
  Il salvataggio dei valori nello state comporterebbe almeno 4 operazioni SSTORE da 20000gas ognuna, per un totale di 80000gas.
  Il Log della transazione comporta il consumo di 375gas di log + 750gas per i due topic indicizzati + 8gas x 128 (totale dei bytes loggati) per un totale di 2149gas.
- I parametri loggati possono essere letti sotto forma di pseudo query attraverso una *call* al nodo.

```
interface ILogModel
{
    event TokenAssigned (
        address indexed user,
        uint8 indexed year,
        uint8 indexed month,
        bytes32 BcParams,
        bytes32 hash
    );
}
```

Figura 10 ILogModel - descrizione delle interfacce

L'implementazione dell'interfaccia ILogModel esporrà una API che permetterà di loggare le informazioni utili al fine di risalire a tutte le assegnazioni di token da parte del Data Service Manager (vedere paragrafo 2.3.5).

I dati loggati sono:

- User: account ethereum del Token Holder
- Year: anno di assegnazione dei token
- Month: mese di assegnazione dei token
- BcParams: parametri notarizzati su Blockchain (vedere paragrafo 2.2.4) del modello utilizzati in fase di calcolo dei token
- Hash: digest crittografico (vedere paragrafo 2.3.2.1)

Attraverso i log, qualsiasi applicativo collegato alla blockchain (nella piattaforma RDS la Dapp) è in grado di conoscere tutte le assegnazioni token verso i Token Holder e avere le informazioni utili per richiedere il takeout (vedere paragrafo 2.3.4.3).

La lettura dei log è un'operazione che non ha impatti sulle performance della blockchain, queste non sono *transaction* ma delle *call.* Le call non vengono propagate a tutti i nodi ma, sono delle chiamate al nodo al quale si è collegati. Pur non essendo delle chiamate onerose, è difficile prevedere i tempi di lettura per una notevole quantità di log (es tutti i log di un Token Holder degli ultimi 5 anni).

Per tale motivo, alcuni dei capi sono stati indicizzati in modo da permettere un filtro iniziale nella lettura dei log.
I campi indicizzati sono: user, year e month e, grazie a questi, è possibile filtrare i log per utente, per anno e per mese.

Un uso pratico di tali filtri è ben comprensibile presentando la schermata di assegnazione dei log proposta dalla Dapp ai Token Holder.
Ciascun Token Holder, nella pagina di riepilogo delle assegnazioni, vedrà inizialmente le assegnazioni filtrate per:

- Account: proprio account
- Year: anno corrente
- Month: mese corrente.

Il Token Holder non potra modificare il campo di filtro account, ma potrà modificare i campi year e month, in questo modo, potrà avere una visione precisa e puntuale di tutte le assegnazioni riferite ad uno specifico mese e ad uno specifico anno.

I limiti attuali di Ethereum non permettono di eseguire filtri più precisi in merito alla data.

Un possibile workaround potrebbe prevedere l'indicazione da parte dell'utente di una data a ritroso nel tempo, convertire la data in timestamp e analizzare il timestamp di ciascun blocco a ritroso nel tempo fino a trovare il blocco più vicino alla data selezionata e, quindi, filtrare i log da un blocco iniziale al blocco corrente. L'implementazione comporterebbe comunque un ciclo con un numero non ben definito di blocchi e, pertanto, assicurare una stabilità in termini di performance non è possibile.

La soluzione attuale prevederà, da parte dell'utente la sola possibilità di filtrare per anno e per mese, in modo da visualizzare le assegnazioni mensili di token.

2.2.4 Models Smart Contract

Al fine di garantire:

- privacy sui dati di consumo delle utenze
- trasparenza e gestione dei parametri configurabili da parte dell'Oracle Admin

si è deciso di:

- salvare i parametri su Blockchain in chiaro
- salvare su blockchain delle informazioni che attestano i consumi associati ai token assegnati senza rivelarne il contenuto
- lasciare che il Data Service Manager legga tali parametri – in fase di calcolo – dalla blockchain.

Lo Smart Contract che offre le funzionalità di gestione dei parametri è il Models Smart Contract.

A titolo esemplificativo consideriamo:

- una comunità Energetica con id: **001**

- un modello di calcolo con id: **002**
- dei parametri di calcolo **C = 100** ed **E = 50**

Lo Smart Contract, in una opportuna struttura dati, notarizzerà che per la comunità **001** e per il modello **002**, i parametri da utilizzare in fase di calcolo dei token da assegnare sono C = **100** e E = **50**.

Una delle richieste progettuali è la possibilità, da parte di un Oracle Admin di modificare a piacimento i parametri del modello. La modifica non riguarda unicamente i valori dei parametri, nel nostro caso **100** e **50** ma, anche il numero dei parametri del modello.
Se, in una diversa fase di vita della soluzione, per il modello **002** si dovesse ravvisare l'esigenza di un terzo parametro **D,** una gestione dei parametri come singole allocazioni di memoria nello stato dello Smart Contract (es. variabile C = 100) e la natura immutabile degli Smart Contract, richiederebbe di deployare su Blockchain un nuovo Smart Contract.

Appare evidente che tale soluzione non offre gli adeguati livelli di scalabilità.

Al fine di garantire un'adeguata scalabilità dei parametri di modello si è deciso di rappresentarli come array di byte.
Lo stato su Smart Contract non è più composto da n allocazioni di memoria (una per ogni parametro), ma da una sola allocazione che contiene l'array di byte.
I parametri vengono visualizzati e gestiti, lato Amministratore, come un array di valori, nel nostro caso rappresentato a titolo d'esempio in Figura 11:

```
const params = [
  {
    name: 'C',
    value: 100
  },
  {
    name: 'E',
    value: 50
  }
]
```

Figura 11 Esempio array di parametri di configurazione del modello

Quest'array viene:
- convertito in una stringa
- convertito in un HEX (stringa esadecimale)
- convertito in un array di byte

L'array di byte viene salvato su blockchain in modo che: per l'id di comunità **001**, per il modello **002** i parametri da utilizzare siano quelli rappresentati dall'array di byte**.**

In questo modo è stata garantita:
- trasparenza sui parametri: l'array può essere letto e riconvertito da chiunque (oltre al Data Service Manager) fino ad arrivare all'array iniziale;
- scalabilità: all'array iniziale possono essere aggiunti altri parametri e questo nuovo array può essere sostituito al precedente, senza effettuare il deployment di un nuovo smart contract.

La scelta di utilizzare un array di byte risulta più efficiente e meno oneroso verso la EVM rispetto all'utilizzo di stringhe.

In Figura 12 sono rappresentate le interfacce implementate per lo sviluppo dei modelli lato Blockchain.

```
interface IModels {

    function setModel(uint8 communityId, uint8 modelId, bytes32 params) external returns (bool);

}
```

Figura 12 IModels - descrizione delle interfacce

2.3 Flussi

Di seguito vengono presentati i flussi di dettaglio delle funzionalità suddivise tra i diversi Admin e gli utenti finali.

2.3.1 Platform Admin

Il Platform Admin può:

- Crea comunità
- Crea utenti amministratori
- Associa ruoli amministratore su blockchain
- Abilitare gli utenti sull'Identity Provider
- Associare gli account utente alle comunità
- Assegnare ruoli agli utenti

2.3.1.1 Creazione Comunità

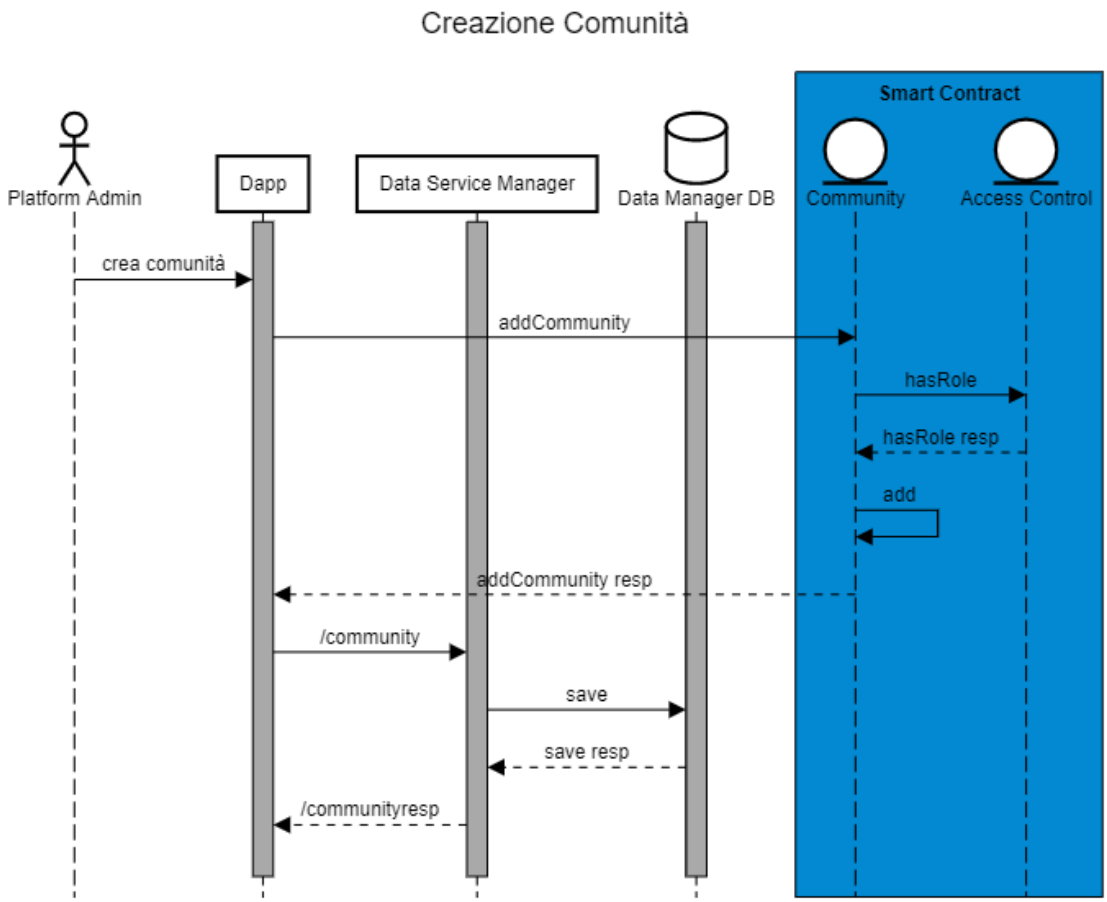


Figura 13 Creazione Comunità - diagramma di flusso

Nel diagramma che rappresenta i flussi operativi (Figura 13) sono descritte le seguenti interazioni:

1) Il Platform Admin tramite apposita form della Dapp inserisce:
    a. Id di comunità
    b. Nome di comunità
    c. Simbolo del Token
    d. Chiave di comunità
    e. Wallet di comunità
2) La Dapp invoca il Community Smart Contract e crea la comunità.
3) Il Community Smart Contract verifica se l'utente ha i giusti permessi
4) Il Community Smart Contract aggiunge la Comunità
5) La Dapp chiama il Data Service manager e salva la comunità
6) Il Data Service Manager salva la comunità

2.3.1.2 CREAZIONE UTENTI AMMINISTRATORI

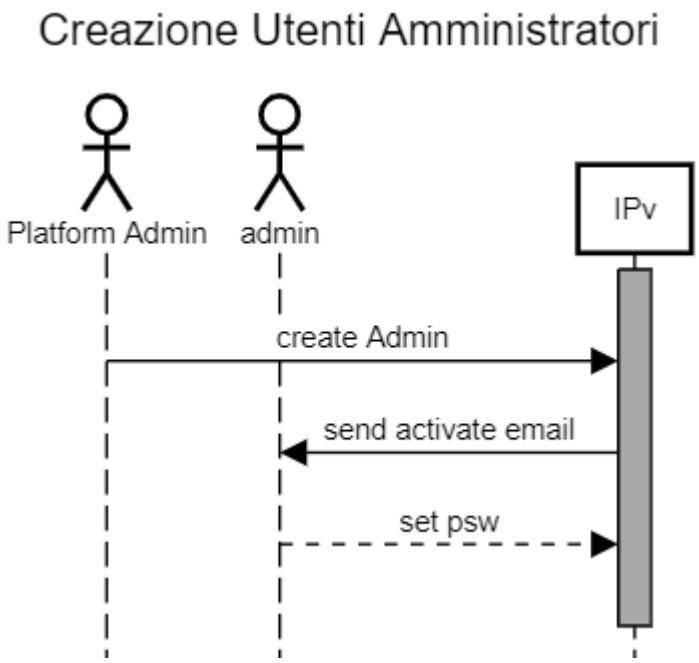


Figura 14 Creazione Utenti Amministratori - descrizione di flusso

Nel diagramma che rappresenta i flussi operativi (Figura 14) sono descritte le seguenti interazioni:

1) Il Platform Admin si collega all'interfaccia dell'Identity Provider e crea un nuovo utente amministratore assegnandogli il ruolo corrispondente
2) L'Identity Provider invia una mail di attivazione all'amministratore
3) L'amministratore tramite link contenuto nell'email setta la propria password di accesso alla DAPP

2.3.1.3 ASSOCIAZIONE RUOLO AMMINISTRATORE SU BLOCKCHAIN

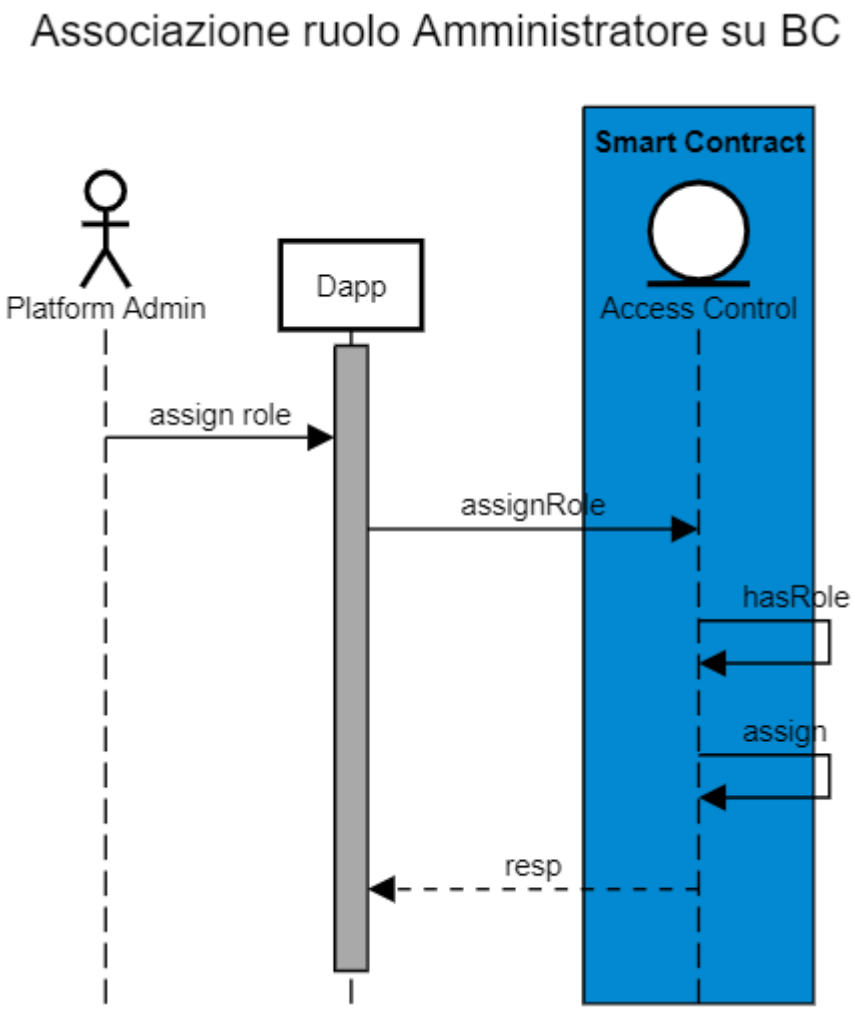


Figura 15 Associazione ruolo Amministratore BC - descrizione di flusso

Nel diagramma che rappresenta i flussi operativi (Figura 15) sono descritte le seguenti interazioni:

1) Il Platform Admin tramite interfaccia della Dapp seleziona un amministratore e sceglie il ruolo da assegnargli.
2) La Dapp invoca l'Access Control e verifica i permessi del chiamante
3) L'Access Control assegna il ruolo all'amministratore

2.3.1.4 ASSOCIAZIONE RUOLO TOKEN-HOLDER SU IPv

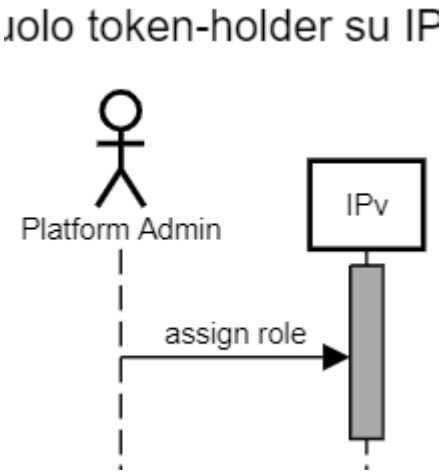


Figura 16 Associazione ruolo Token-Holder su Identity Provider – descrizione di flusso

Nel diagramma che rappresenta i flussi operativi (Figura 16) sono descritte le seguenti interazioni:

1) Il Platform Admin collegandosi all'interfaccia dell'Identity Provider assegna il ruolo token-holder all'utente

2.3.1.5 Revoca operatività utente

Qualora il sistema dovesse rilevare un'anomalia nell'assegnazione dei token a livello di singolo utente o, per qualsiasi altra ragione, il Platform Admin ha la possibilità di sospendere l'operatività dell'Utente sul sistema. Tale sospensione, può essere eseguita su due livelli:

i. Revoca accesso alla piattaforma
ii. Revoca accesso alle funzionalità su blockchain

2.3.1.6 Revoca accesso alla piattaforma

Il Platform Admin, accedendo all'interfaccia di Key Cloak, può revocare il ruolo ad un utente inibendogli l'accesso alla piattaforma. Nello specifico, la rimozione del ruolo non consentirà:

- Lato IAM di richiamare le API esposte, nello specifico, l'utente non potrà eseguire la login e quindi ricevere il token JWT di autorizzazione verso le API esposte dello IAM.
- Lato Data Service Manager l'utente non avrà più nessun livello autorizzativo per poter chiamare la totalità delle API esposte.

La rimozione del ruolo su Key Cloak non comporta la rimozione automatica del ruolo su Blockchain, pertanto, l'utente avrà ancora le autorizzazioni per operare su blockchain. La rimozione del ruolo su blockchain è affrontata nel paragrafo 2.3.1.4

2.3.1.7 Revoca accesso alle funzionalità su blockchain

Revocare un ruolo su Blockchain implica l'impossibilità – per uno specifico utente – di operare sugli Smart Contract della piattaforma. La revoca dei ruoli può essere eseguita dal Platform Admin o dal Community Admin in base al ruolo da eliminare.
Il Platform Admin si occupa della rimozione dei ruoli agli Amministratori mentre, il Community Admin, del ruolo ai Token Holder.

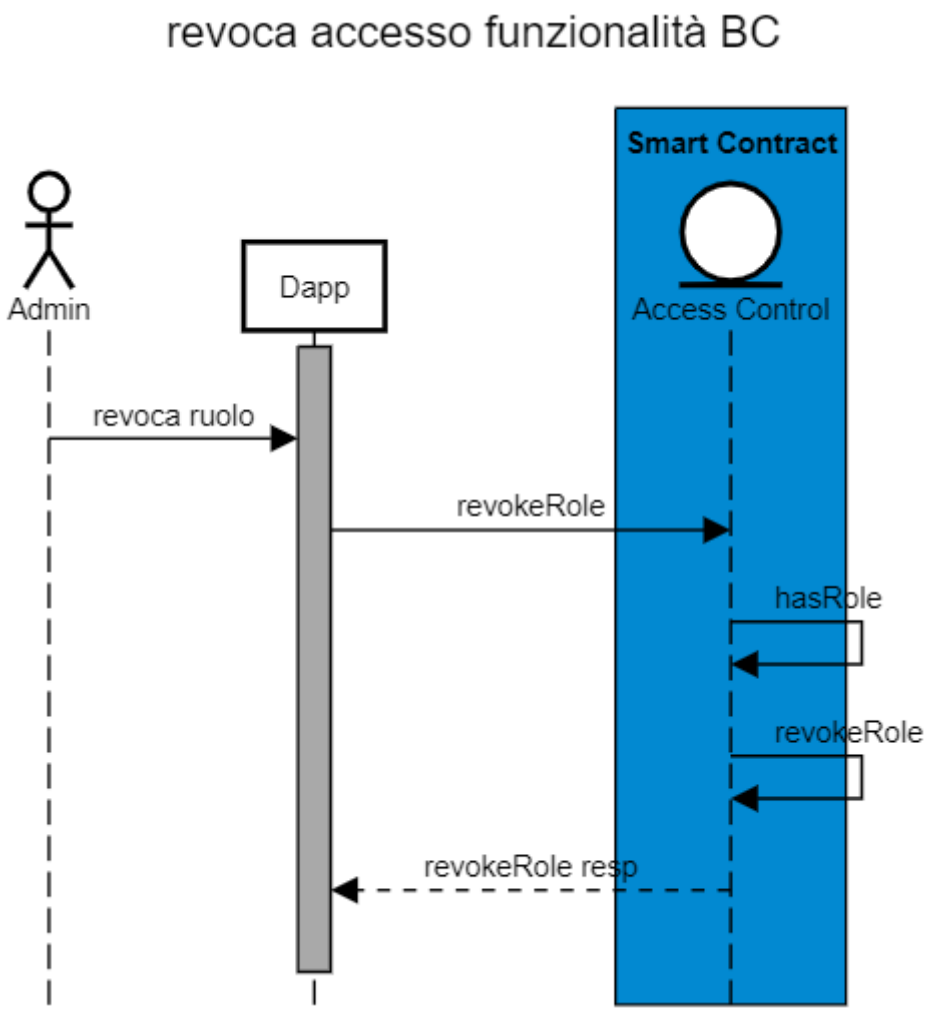


Figura 17 Revoca accesso funzionalità BC - descrizione di flusso

Nel diagramma che rappresenta i flussi operativi (Figura 16) sono descritte le seguenti interazioni:

1) Il Platform Admin o il Community Admin tramite form della Dapp, invia una richiesta all'Access Control Smart Contract per revocare il ruolo ad uno specifico utente
2) L'Access Control verifica il ruolo del chiamante
3) L'Access Control revoca il ruolo all'utente

2.3.2 Oracle Admin

L'Oracle Admin può:

- Aggiornare i modelli sul Data Service Manager
- Aggiornare i parametri dei modelli su Blockchain.

2.3.2.1 Aggiornamento modello

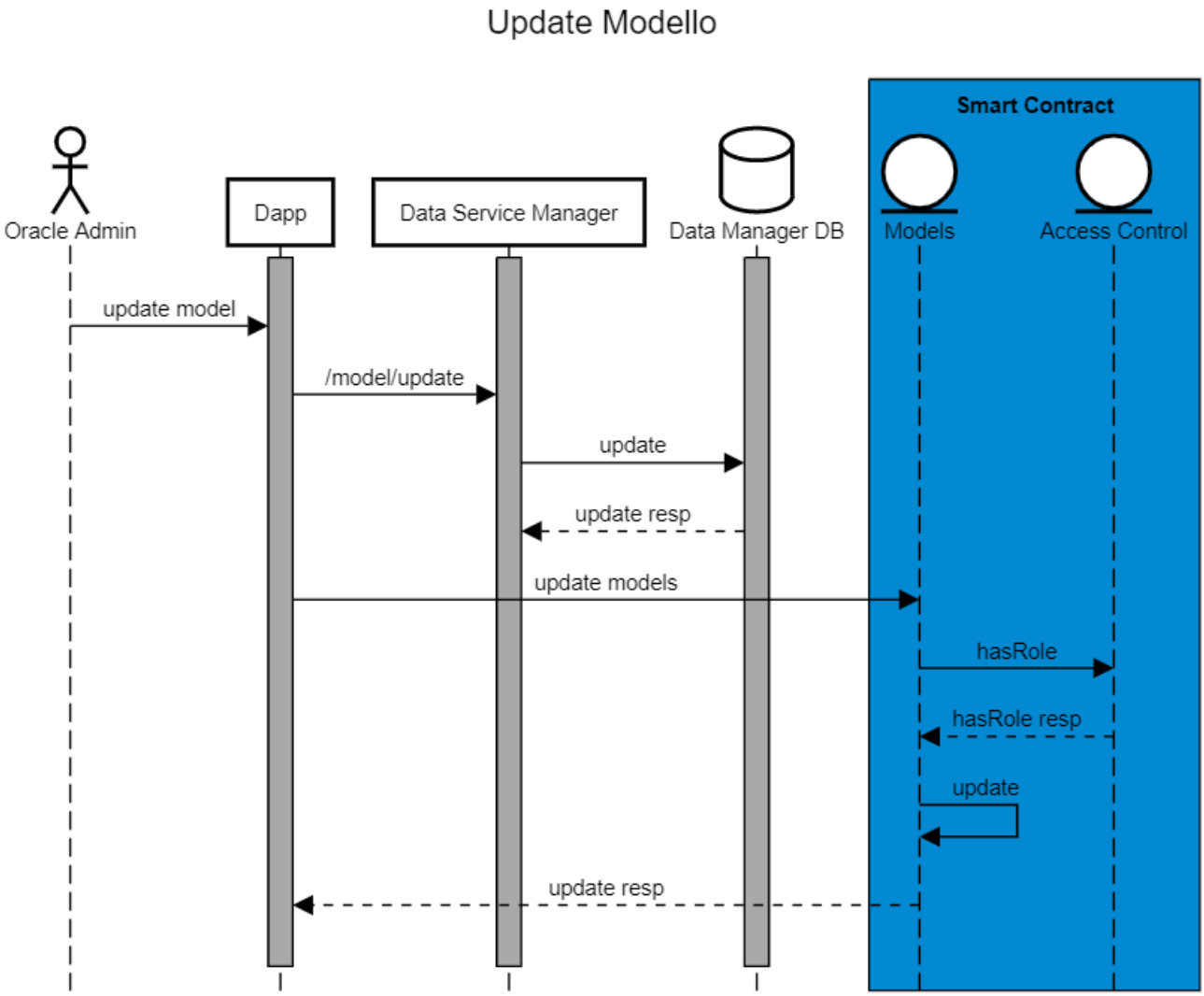


Figura 18 Aggiornamento del Modello - descrizione di flusso

Il flusso (Figura 18) rappresenta la configurazione di un modello sul Data Service Manager da parte dell'Oracle Admin.

1) L'Oracle Admin tramite interfaccia della Dapp può modificare i parametri del modello salvati su Data Manager DB o notarizzati su blockchain.
2) Se i parametri su Data manager DB vengono modificati la Dapp invoca il Data Service Manager
3) Il Data Service Manager esegue un update dei parametri
4) Se i parametri notarizzati su Blockchain vengono modificati la Dapp chiama il Model Smart Contract per aggiornare i parametri
5) Il Model Smart Contract verifica tramite Access Control se il chiamante ha i permessi adeguati
6) Il model Smart Contract aggiorna i parametri.

Di seguito il modello di configurazione salvato su Data Manager DB:

```
const DM_DB_params = {
  id: 1,
  community: ["A", "B"],
  frequency: 60
};
```

Figura 19 Modello di configurazione utilizzato dal Data Manager DB

Dove:

- Id: identificativo del modello
- Community: indica le comunità alle quali viene applicato il modello
- Frequency: frequenza di esecuzione dello scheduler

I parametri community e frequency possono essere modificati in qualsiasi momento

Di seguito il modello di configurazione salvato su Model Smart Contract:

```
const BC_params = {
  id: 1,
  params: [{name: "A", value: 100}, {name: "B", value: 200}]
};
```

Figura 20 Modello di configurazione utilizzato dallo Smart Contract

Dove:

- Id: Identificativo del modello
- Params: parametri da notarizzare su blockchain. Viene rappresentato come un array di oggetti, ogno oggetto rappresenta un parametro. Un parametro è identificato da due chiavi.
  - name: nome del parametro
  - value: valore del parametro

  Sia name che value possono essere modificati in qualsiasi momento.

2.3.3 Community Admin

Il Community Admin può:

- Assegnare ruolo token-holder su BC
- Associare utenti alla comunità
- Gestire le fee in fase di assegnazione token
- Gestire gli incentivi in fase di scambio token
- Bloccare lo scambio dei token in caso di rilevata anomalia
- Gestire i token di benvenuto.

2.3.3.1 Assegnazione Ruolo Token-Holder Su Blockchain

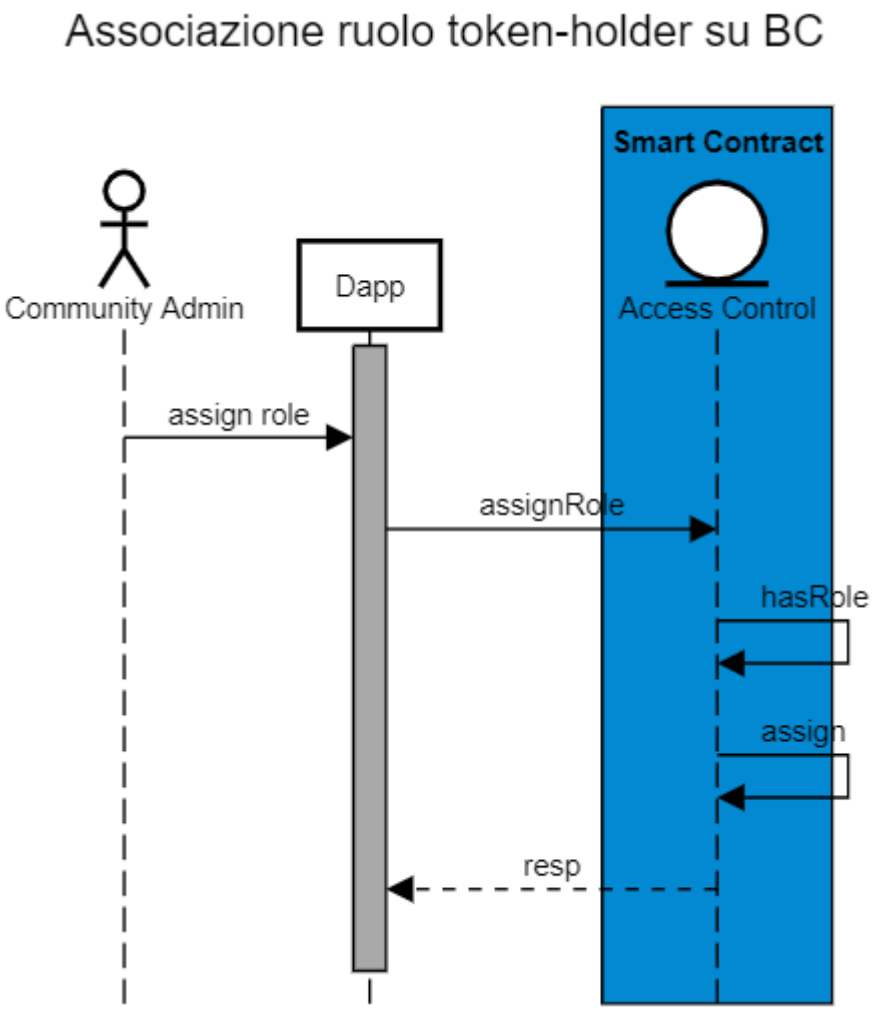


Figura 21 Associazione ruolo Token-Holder su Blockchain - descrizione di flusso

Nel diagramma che rappresenta i flussi operativi (Figura 21) sono descritte le seguenti interazioni:

1) Il Community Admin tramite interfaccia della Dapp seleziona un utente di comunità.
2) La Dapp invoca l'Access Control e verifica i permessi del chiamante
3) L'Access Control assegna il ruolo all'utente

2.3.3.2 Associazione Utente A Comunità

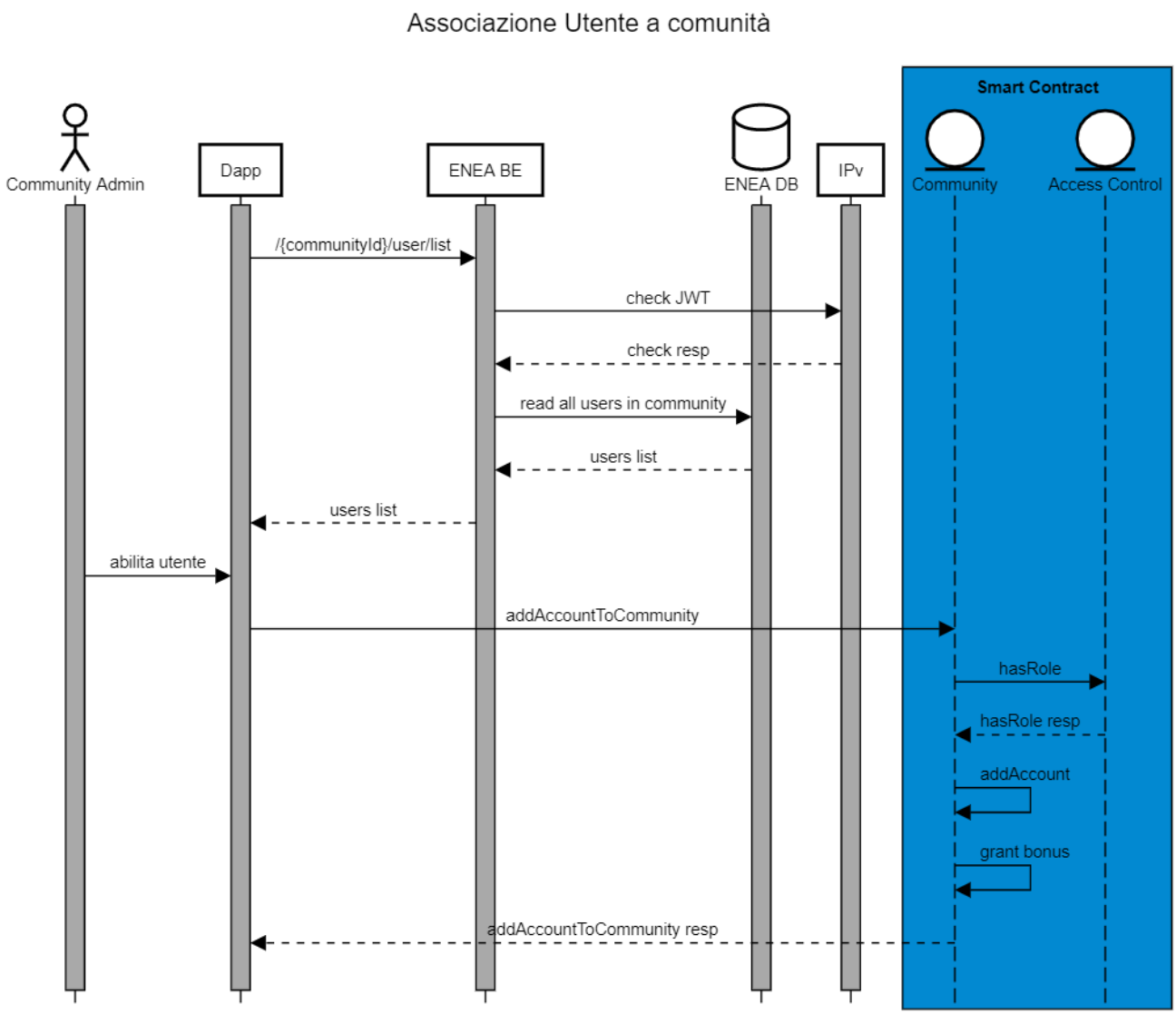


Figura 22 Associazione utente a comunità - descrizione di flusso

Il flusso (Figura 22) rappresenta il processo di associazione di un utente ad una comunità da parte del Community Admin. Si ricorda che:

- un utente si registra in via autonoma (vedere paragrafo 2.3.4.2)
- gli viene associato il ruolo di token-admin dal Platform Admin su Identity Provider (vedere paragrafo 2.3.1.4)
- gli viene associato il ruolo di token-admin dal Community Admin su Blockchain (vedere paragrafo 2.3.3.1)

Di seguito il dettaglio del flusso rappresentato in Figura 22:

1) La Dapp effettua una chiamata all'ENEA DE per la lista di tutti gli utenti della comunità
2) ENEA BE verifica tramite IPv la validità del token JWT nell'header della chiamata
3) ENEA BE legge tutti gli utenti della comunità da ENEA DB
4) ENEA BE ritorna la lista di tutti gli utenti alla DAPP
5) Il Community Admin tramite interfaccia della Dapp seleziona un utente e invia una richiesta al Community Smart Contract

6) Il Community Smart Contract tramite Access Control Smart Contract verifica che il chiamante abbia i permessi adeguati e che l'utente abbia il ruolo di token-holder
7) Il Community Smart Contract assegna l'eventuale bonus di benvenuto
8) Il Community Smart Contract aggiunge l'utente alla comunità

2.3.3.3 Gestione Fee

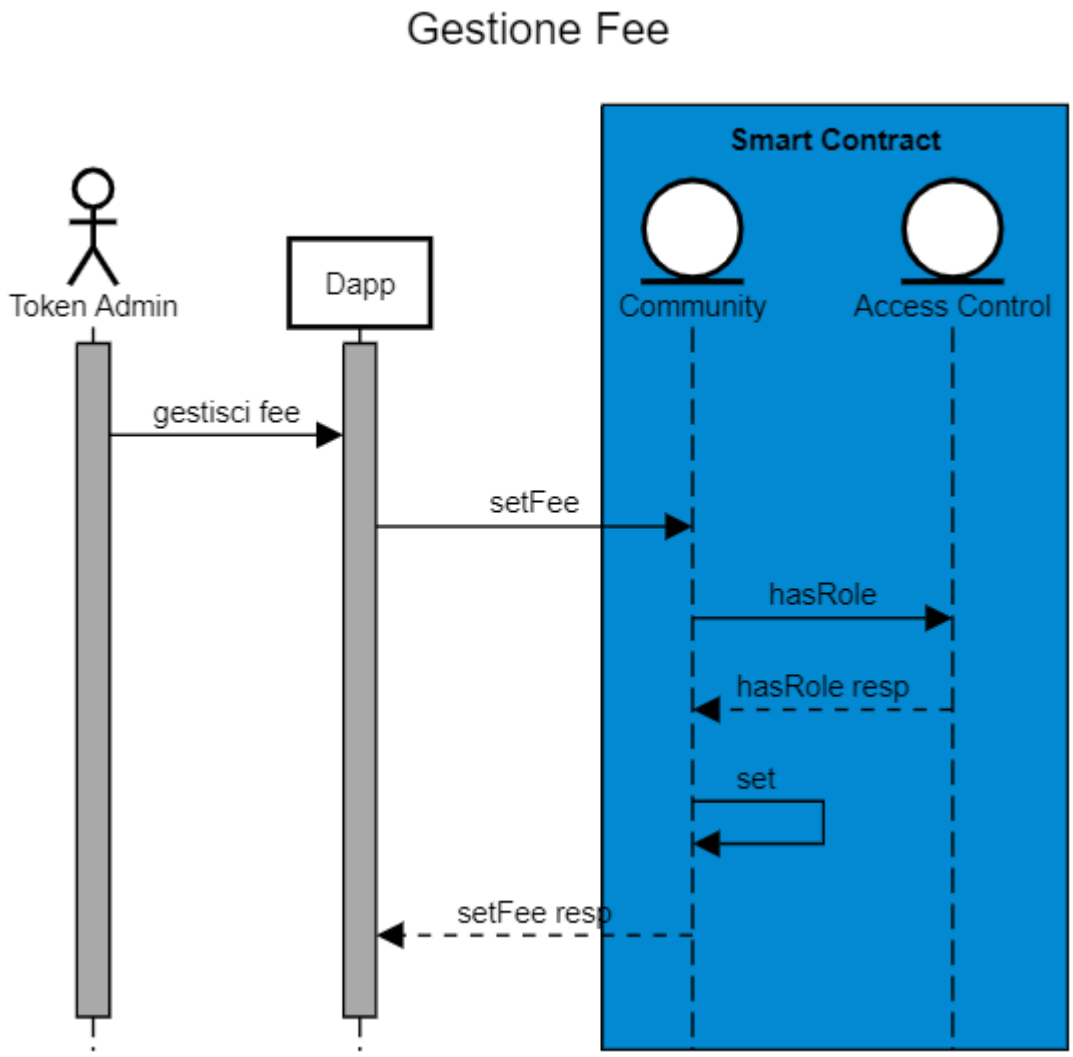


Figura 23 Gestione Fee - descrizione di flusso

Nel diagramma che rappresenta i flussi operativi (Figura 23) sono descritte le seguenti interazioni:

1) Il Token Admin tramite apposita form della Dapp inserisce l'id di comunità e la fee
2) La Dapp invoca il Community Smart Contract
3) Il Community Smart Contract verifica se l'utente ha i giusti permessi
4) Il Community Smart Contract assegnala fee alla Comunità

2.3.3.4 Gestione Incentivi

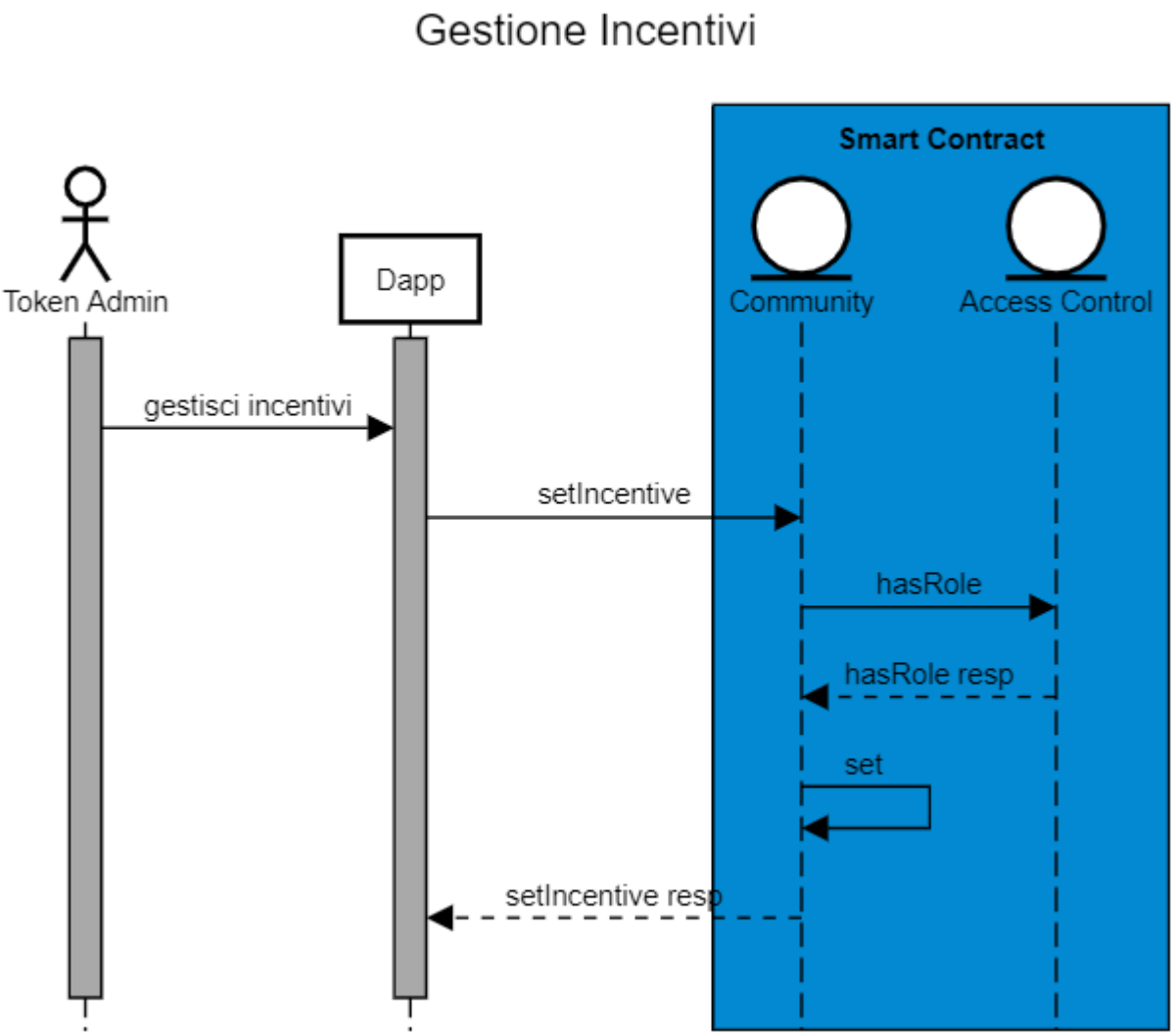


Figura 24 Gestione Incentivi - descrizione di flusso

Nel diagramma che rappresenta i flussi operativi (Figura 21) sono descritte le seguenti interazioni:

1) Il Token Admin tramite apposita form della Dapp inserisce l'id di servizio e l'incentivo
2) La Dapp invoca il Community Smart Contract
3) Il Community Smart Contract verifica se l'utente ha i giusti permessi
4) Il Community Smart Contract assegnala l'incentivo al servizio

2.3.3.5 Blocco Operazioni sui token

Nel caso in cui si riscontrassero anomalie in fase di assegnazione token o in fase di scambio token, il Token Admin può sospendere l'operatività del Community Smart Contract. La sospensione preclude l'esecuzione di tutte le funzionalità eseguibili da parte dei Token Holder.

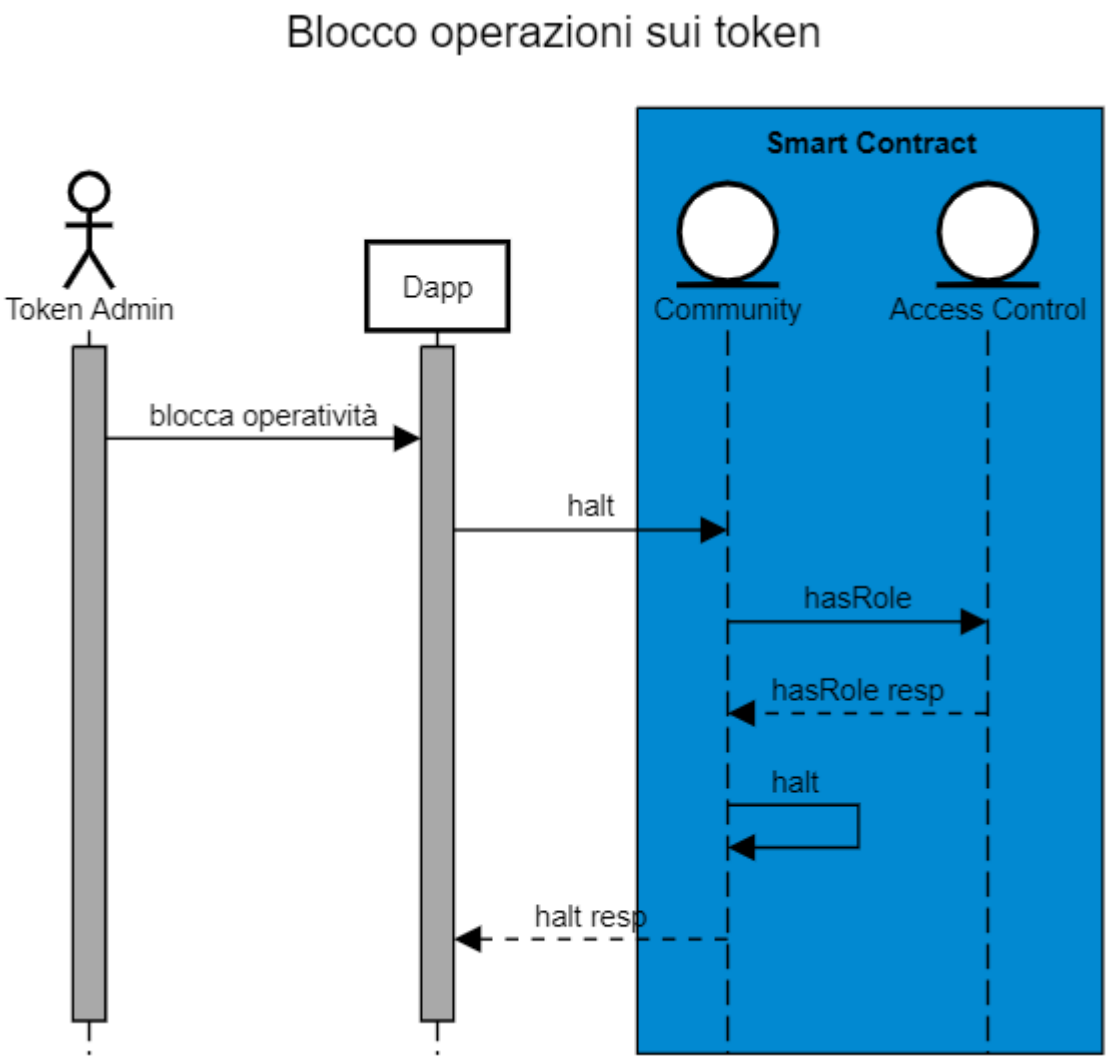


Figura 25 Blocco operazioni sui token - descrizione di flusso

Nel diagramma che rappresenta i flussi operativi (Figura 25) sono descritte le seguenti interazioni:

1) Il Token Admin tramite apposita funzionalità della Dapp può decidere di bloccare l'operatività sul Community Smart Contract
2) La Dapp richiama la funzionalità di sospensione del Community Token
3) Il Community Token verifica se l'utente ha i permessi adeguati per richiamare la funzionalità
4) Il Community Smart Contract blocca l'operatività.

2.3.3.6 Gestione Bonus

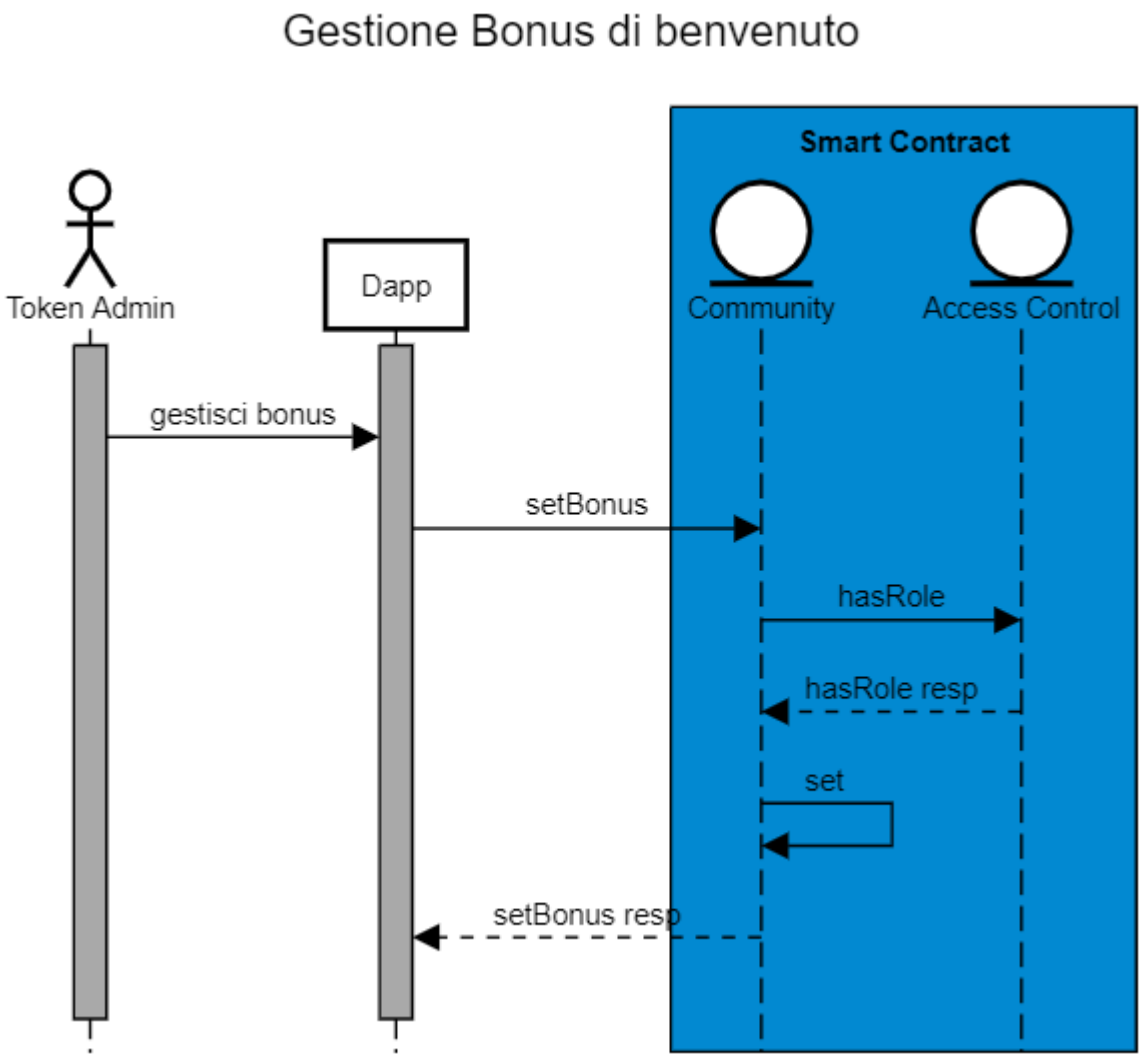


Figura 26 Gestione bonus di benvenuto - descrizione di flusso

Nel diagramma che rappresenta i flussi operativi (Figura 26) sono descritte le seguenti interazioni:

1) Il Token Admin tramite apposita form della Dapp inserisce l'id di comunità e il bonus
2) La Dapp invoca il Community Smart Contract
3) Il Community Smart Contract verifica se l'utente ha i giusti permessi
4) Il Community Smart Contract assegna il bonus alla comunità

2.3.4 Token holder

Il Token Holder è l'utenza che viene premiato, sotto forma di token, per i virtuosismi in termini di consumo o produzione energetica. I Token accumulati possono essere scambiati con altri utenti della rete o, presso specifici service provider.

Il Token Holder, oltre alla pagina di profilo in comune con gli altri utenti che, presenterà i dati anagrafici memorizzati sullo IAM*: nome, cognome, account ethereum, codice fiscale*, avrà a disposizione una propria pagina riepilogativa che riporterà il saldo totale dei token a sua disposizione e gli consentirà dispenderli per i servizi associati alla comunità.

2.3.4.1 Spesa Token Per Servizi

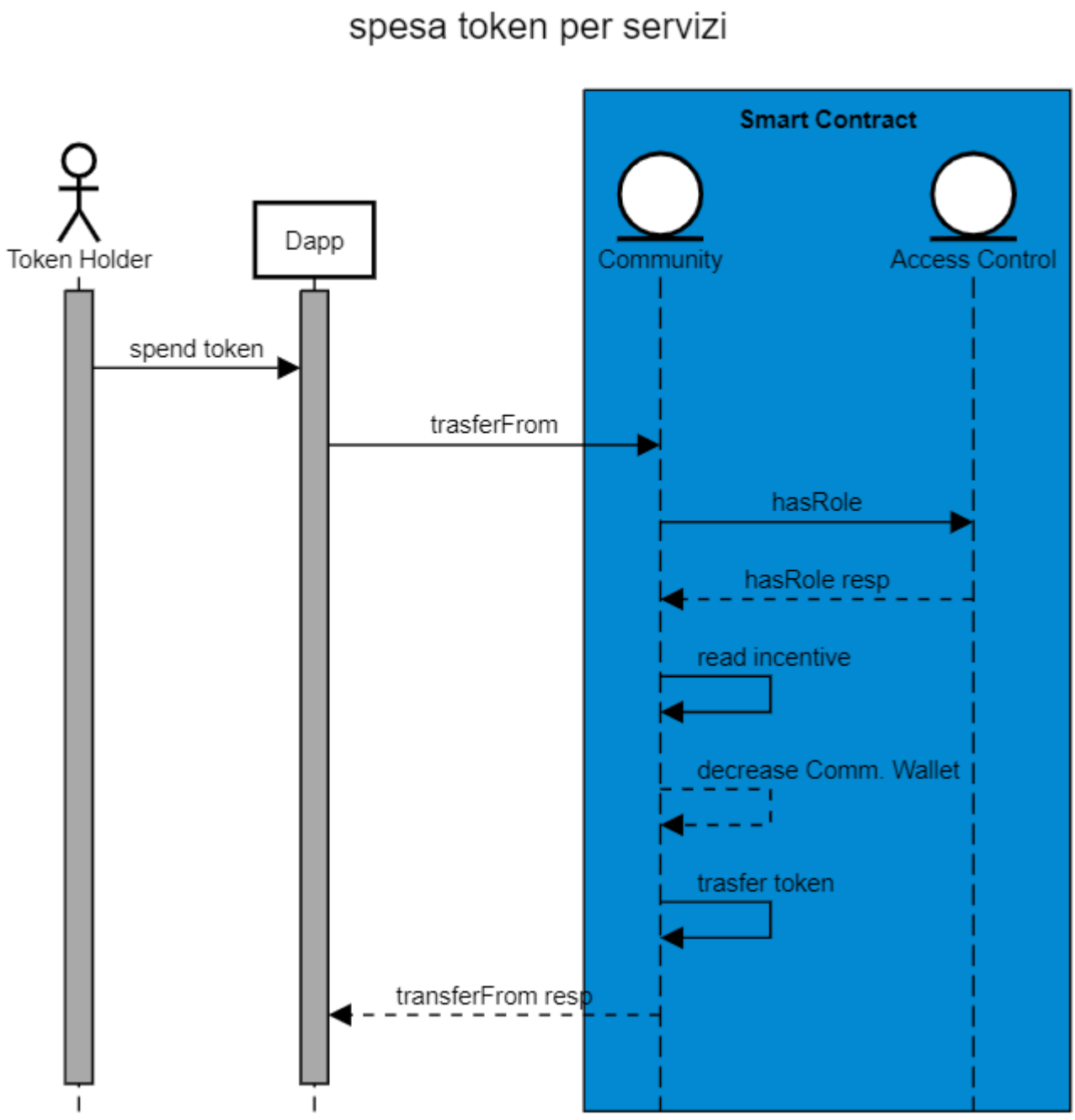


Figura 27 Spesa token per servizi - descrizione di flusso

Un utente può spendere i token accumulati tramite produzione energetica per una serie di servizi di comunità. La spesa può essere soggetta ad incentivi (vedere paragrafo 2.2.1.4) questo vuol dire che parte del dovuto, viene pagato dai token in possesso dal community wallet (vedere capitolo 2.2.1). Di seguito il processo rappresentato in Figura 27 nel dettaglio:

1) Il Token Holder può spendere i propri token collegandosi alla Dapp e indicando:
    a. numero di token da scambiare
    b. identificativo del servizio
2) La Dapp invoca la funzione transferFrom per il trasferimento dei token

3) Il Community Smart Contract verifica che l'utente abbia il ruolo di Token Holder
4) Il Community Smart Contract verifica eventuali incentivi in fase di spesa per il servizio
5) Se l'incentivo è presente il Community Smart Contract sottrae l'importo dell'incentivo dal Community Wallet
6) Il Community Smart Contract sottrae i restanti token da pagare dal wallet dell'utente.
7) Il Community Smart Contract accredita i token al beneficiario

2.3.4.2 Signup

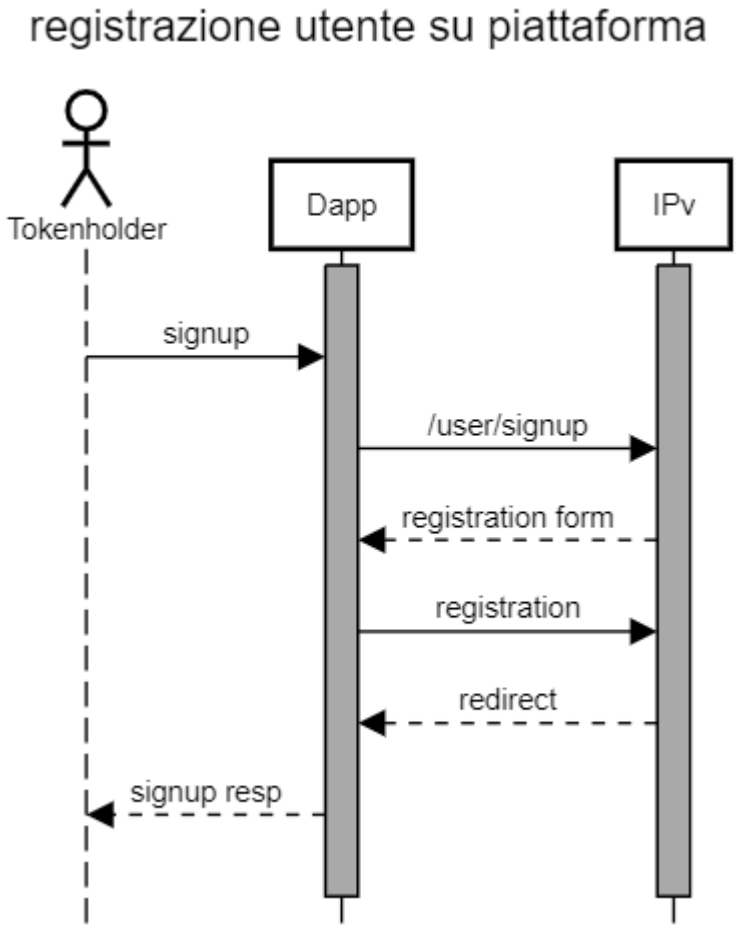


Figura 28 Registrazione utente su piattaforma – descrizione di flusso

Nel diagramma che rappresenta i flussi operativi (Figura 28) sono descritte le seguenti interazioni:

1) La Dapp invia una richiesta di registrazione all'Identity Provider
2) L'Identity Provider ritorna la pagina di registrazione per l'utente
3) L'utente compila la form di registrazione e inserisce: username, nome, cognome, email, walletId e comunità
4) L'Identity Provider salva l'utente e fa redirect verso la Dapp

A questo punto l'utente non è ancora abilitato alle operazioni, serve che l'admin assegni un ruolo per poter essere operativo.

2.3.4.3 Takeout

Uno degli obiettivi cardine della soluzione è di offrire il più alto grado di trasparenza verso gli utenti finali in merito alla correttezza dei token a loro assegnati.

Nel paragrafo 2.3.5 viene descritto il flusso di assegnazione token. Tale assegnazione è il risultato di un modello di calcolo che prende in input i dati di consumo o produzione di un utente e i parametri di modello sono notarizzati su blockchain come descritto nel paragrafo 2.3.2.1.

Le attuali limitazioni delle blockchain, in termini di privacy dei dati, non permettono di eseguire i modelli on-chain tramite opportuni Smart Contract, questi, vengono eseguiti off-chain dal Data Service Manager.

Il flusso appena descritto non offre un livello adeguato di trasparenza. Se, un utente finale, in una fase successiva all'assegnazione, volesse conoscere i dati utilizzati dal modello per calcolare i token a lui assegnati, gli unici dati notarizzati su blockchain sarebbero i parametri del modello e non avrebbe le garanzie adeguate in merito ai valori di consumo o produzione utilizzati in fase di calcolo.

Per ovviare a questa problematica, come descritto nel flusso del paragrafo 2.3.5 dai dati viene calcolato un digest crittografico che agisce come commitment dei parametri. Il digest viene notarizzato nello Smart Contract LogModel visto nel paragrafo 2.2.3. In questo modo l'utente in qualsiasi momento può richiedere un takeout per una specifica assegnazione di token.

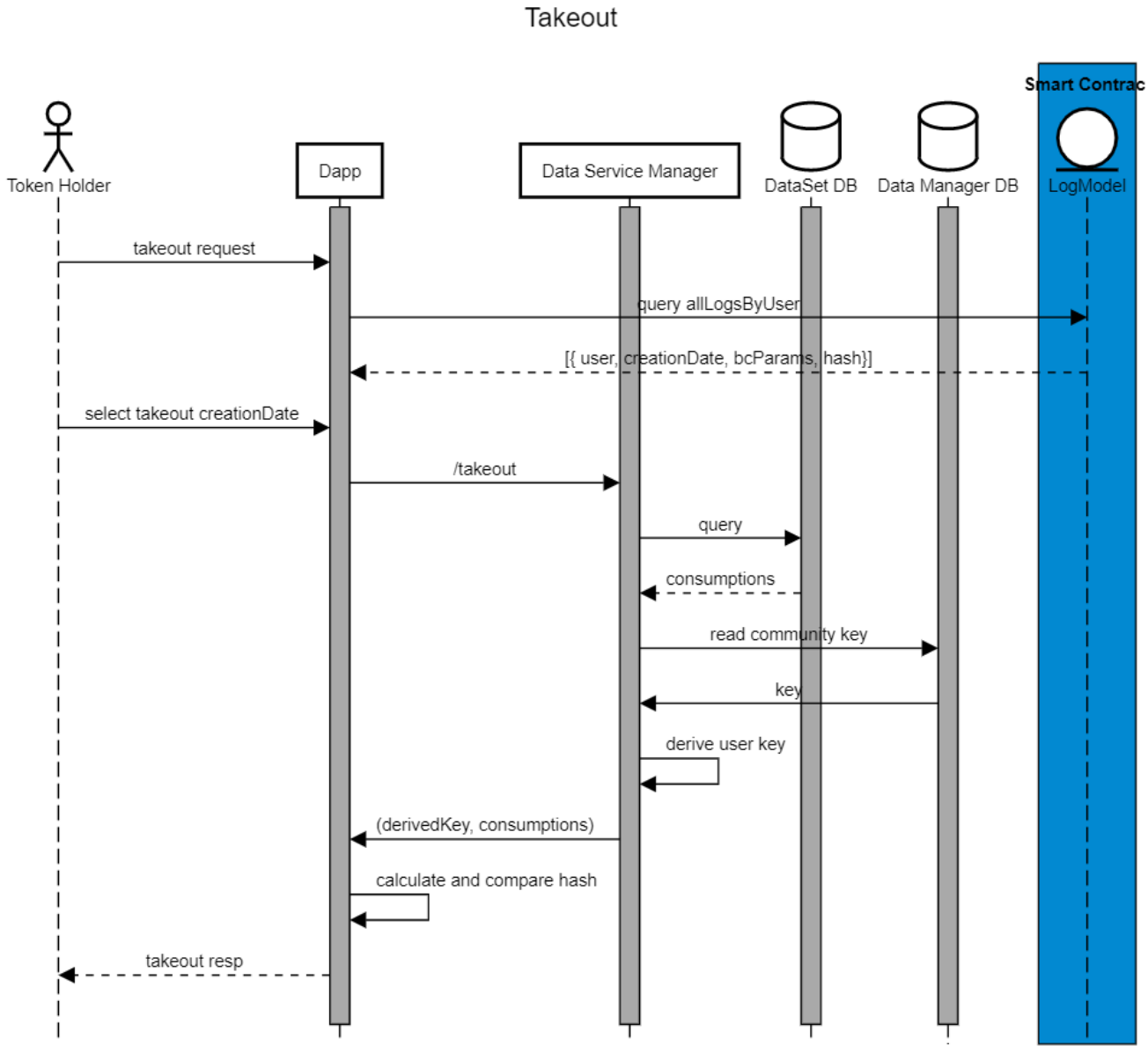


Figura 29 Takeout - descrizione di flusso

Nel diagramma che rappresenta i flussi operativi (Figura 29) sono descritte le seguenti interazioni:

1) Il Token Holder collegandosi alla pagina di riepilogo di tutte le assegnazioni a lui riferite (filtrate per anno e mese vedere paragrafo 2.2.3) di token può richiedere il takeout per una specifica assegnazione.
2) Il token holder richiede il takeout tramite la Dapp
3) La Dapp invoca l'API di takeout del Data Service Manager
4) Il Data Service Manager legge dal DataSet i dati di consumo dell'utente.
5) Il Data Service Manager legge dal Data Manager DB la chiave di comunità.
6) Il Data Service Manager deriva la chiave opportuna rispetto ai dati richiesti dall'utente.
7) Il Data Service Manager restituisce la chiave derivata e i dati di consumo.

La Dapp utilizza i dati di consumo, i parametri del modello e la chiave derivata per verificare il digest di commitment notarizzato su blockchain.

NOTA: Nel paragrafo 2.6 gestione della chiave di comunità con limiti della soluzione adottata

2.3.5 Login

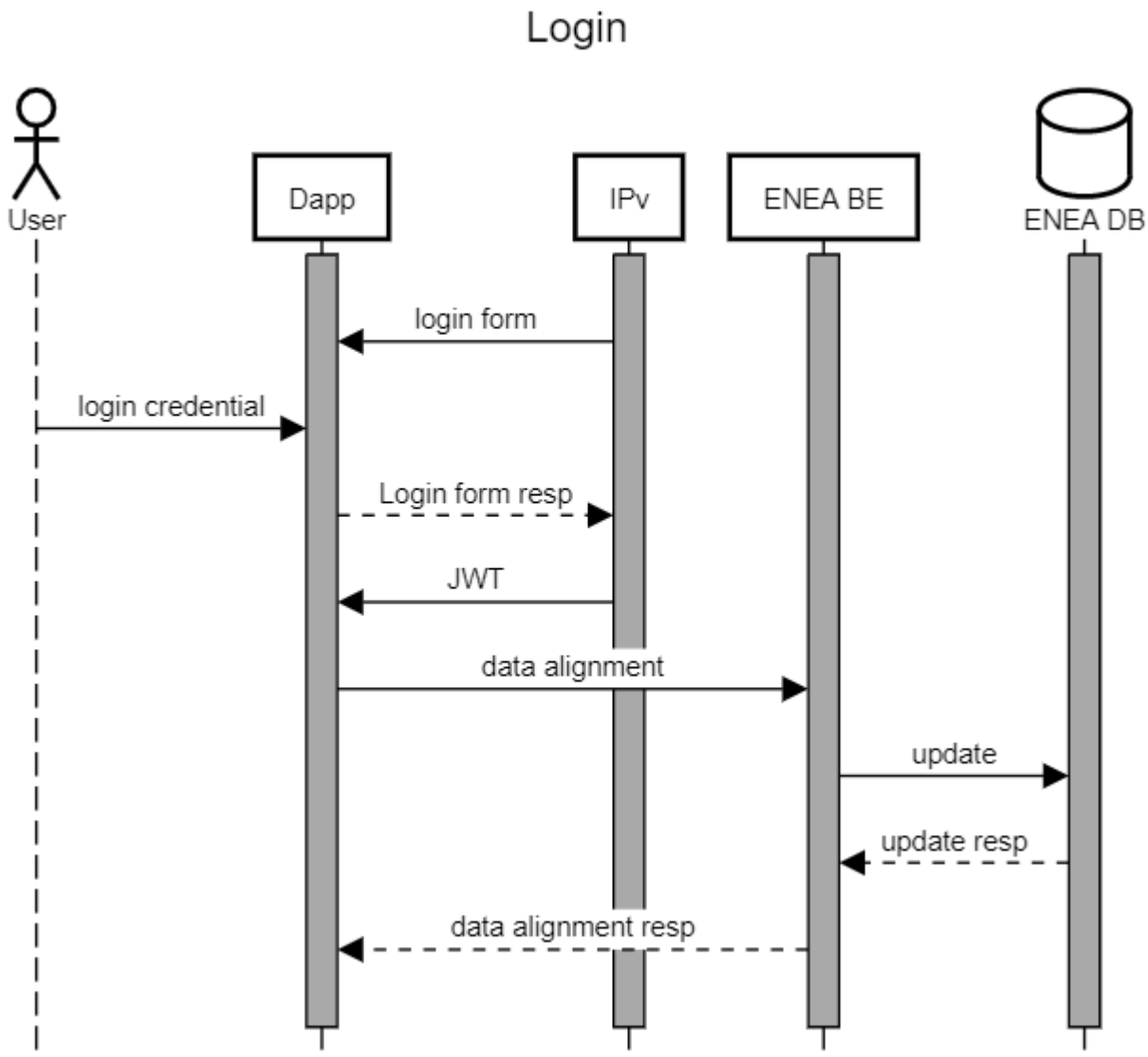


Figura 30 Login - descrizione di flusso

Il flusso di Figura 30 presenta il processo di login per ogni utente della piattaforma. Ad ogni login verrà chiamato un endpoint dell'ENEA BE con il JWT ritornato dall'IPv e contenente tutte le informazioni di un utente registrato in modo da poter allineare le informazioni memorizzate su IPv con il DB ENEA.
Di seguito il flusso rappresentato in Figura 30 nel dettaglio:

1) L'utente accedendo alla Dapp ha la possibilità di loggarsi tramite interfaccia fornita dall'IPv.
2) L'utente inserisce la propria username e password e fa richiesta di login all'IPv
3) In caso di inserimento di credenziali corrette l'IPv restituisce il JWT con le informazioni dell'utente e lo reindirizza all'Home Page della Dapp.
4) La Dapp, invia il JWT all'ENEA BE
5) L'ENEA BE, se l'utente non è presente lo salva o, in alternativa, se i dati utente sono stati modificati, aggiorna il record utente

2.3.6 Scheduler

Lo scheduler è un servizio "batch" del Data Service Manager incaricato di eseguire tutti i modelli salvati nel Data Manager DB, assegnare i token agli utenti e notarizza l'assegnazione per un eventuale takeout da parte del token holder (vedere paragrafo 2.3.4.3).

Lo scheduler è l'unico ruolo autorizzato all'interno della piattaforma ad emettere token verso gli utenti. Per poter eseguire le funzionalità offerte dagli Smart Contract, allo scheduler deve essere associato un wallet in fase di start-up del progetto.

Commitment dei token assegnati a valle dell'assegnazione dei token da parte dello scheduler, viene calcolato e notarizzato un digest crittografico di commitment calcolato sulla base di tre valori:

- userData: Consumi/produzione energetica utente
- params: parametri del modello su blockchain
- key: chiave del tokenholder derivate dalla chiave di comunità

Dal punto di vista tecnico il digest crittografico verrà calcolato tramite una funziona hash standard (SHA3).

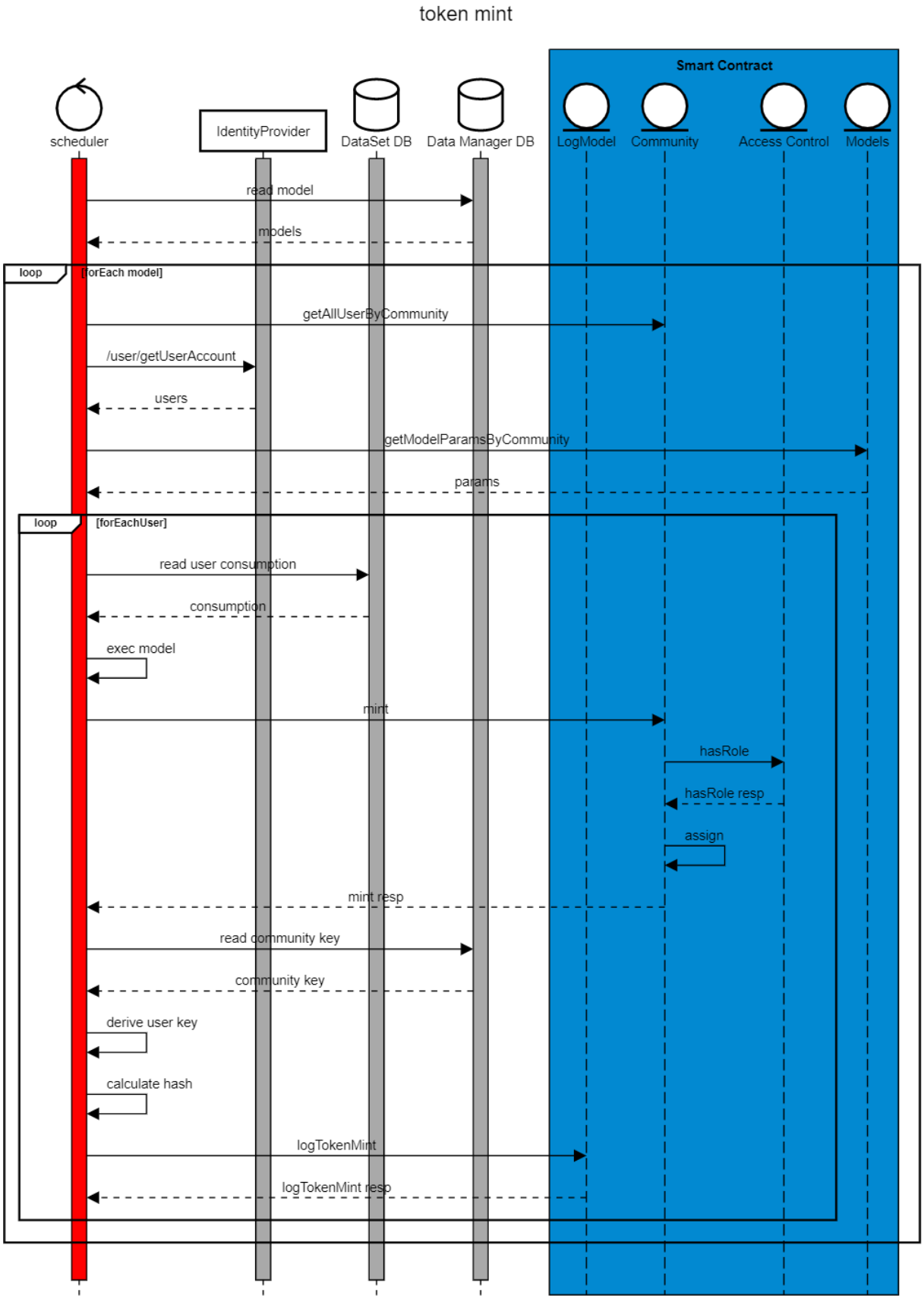


Figura 31 Token mint - descrizione di flusso

Nel diagramma che rappresenta i flussi operativi (Figura 31) sono descritte le seguenti interazioni:

1) Lo scheduler legge tutti i modelli dal Data Manager DB
2) Per ogni modello e per ogni comunità nel modello:
    a. Recupera tutti gli indirizzi ethereum degli utenti di comunità dal Community Smart Contract
    b. Recupera l'utenza per ognuno gli utenti dall'Identity Provider
    c. Recupera i parametri di modello per la comunità in esecuzione
    d. Per ogni utente:
        i. Legge i parametri di consumo dal DataSet DB
        ii. Calcola i token da assegnare
        iii. Assegna i token tramite Community Smart Contract
        iv. Legge la chiave di comunità dal Data Manager DB
        v. Deriva la chiave per i dati da inserire a partire dalla chiave di comunità e dei parametri da inserire
        vi. Calcola il digest per il commitment dei parametr**i.**
        vii. Logga su LogModel Smart Contract:
            1. Account ethereum dell'utente
            2. Data corrente
            3. Digest di commitment
            4. Parametri letti in precedenza su Model Smart Contract

2.4 Authorization Service

Il modulo di Authorization Service comprende lo IAM

Lo IAM è l'entità del sistema che crea, conserva e gestisce le informazioni sull'identità degli utenti e fornisce i servizi di autenticazione.
Nello specifico Lo IAM fornisce le funzionalità di: login, sign-up, gestione utente e autenticazione e autorizzazione del Data Manager Service attraverso le interfacce o le API native di KEYCLOAK.

Lo IAM per ogni utente memorizzerà:

- Nome
- Cognome
- Email
- Codice fiscale
- Account Ethereum
- Community Id

NOTA: Lo IAM utilizzato è stato implementato esclusivamente per rendere funzionante la piattaforma. In questa fase quindi non è implementata la sincronizzazione dei ruoli applicativi e di quelli blockchain.

2.5 Gestione dei ruoli

Come anticipato nel paragrafo 2.1, gli attori coinvolti nella soluzione vengono classificati in base al loro ruolo, i ruoli identificati sono:

1. Community Admin
2. Oracle Admin
3. Token Admin
4. Share Holder

In base al ruolo di appartenenza, gli utenti sono autorizzati ad eseguire determinate operazioni della soluzione.
I ruoli utente vengono memorizzati e gestiti sullo IAM e sullo Smart Contract Access Control. La necessità di avere una doppia gestione dei ruoli è dovuta alle diverse finalità che questi hanno nel sistema.

I ruoli gestiti dallo IAM controllano le autorizzazioni sulle singole API esposte dal Data Service Manager.
I ruoli vengono assegnati manualmente tramite interfaccia dello IAM in fase di abilitazione utente e vengono modificati o rimossi con le medesime modalità.
I ruoli gestiti dall'Access Control controllano le autorizzazioni sulle singole funzionalità offerte da tutti gli Smart Contract del sistema. L'aggiunta, la modifica e la rimozione dei ruoli per i soci, viene eseguita tramite interfaccia offerta dalla Dapp, direttamente l'Access Control Smart Contract.
Una gestione disgiunta dei ruoli sullo IAM e su Blockchain non è sicuramente ottimale in termini di consistenza dei ruoli ma, è obbligata dalle soluzioni di progetto attuali.
Una gestione consistente dovrebbe prevedere la sincronizzazione dei ruoli tra lo IAM e l'Access Control in modo da poter gestire entrambi i livelli autorizzativi e, questo, permetterebbe di avere lato UI una dashboard di assegnazione ruolo unificata tra i due sistemi.
La rimozione del modulo di Community Gateway non ha avuto impatti sul processo iniziale. Ricordiamo che il processo prevedeva la gestione del ruolo sullo IAM e, successiva, gestione del ruolo via Community Gateway verso la Blockchain. La gestione via Community Gateway era abilitata attraverso API RestFull che a loro volta invocavano le funzionalità offerte dall'Access Control. Si potrebbe dire che le API fungevano solo da wrapper per le funzionalità dell'Access Control.
La rimozione del Community Gateway ha demandato le chiamate alle funzionalità offerte dall' Access Control alla Dapp decentralizzando l'interazione tra la Dapp e la Blockchain e rispettando in modo più fedele i principi e le logiche Web 3.0
In una fase successiva di progetto sarà assolutamente necessario mettere in comunicazione lo IAM con la Blockchain e nello specifico con l'Access Control. Questo permetterà di sincronizzare, nel momento in cui un ruolo viene modificato sullo IAM, il corrispettivo ruolo su Blockchain e avere consistenza tra i due.

2.6 Gestione Chiavi di Comunità

Come previsto dai requisiti di progetto, è necessario garantire la trasparenza del sistema preservando al tempo stesso le garanzie di confidenzialità dei dati di consumo. A questo scopo, come descritto in altre parti del documento (si veda la sezione 2.3.2.2 che descrive il processo di takeout e la sezione 2.3.5.1 che descrive l'inserimento dei dati notarizzati nella blockchain) si scelto di impiegare uno schema di commitment con garanzie di confidenzialità dei dati. Tramite questo schema, il Data Service Manager che inserisce un digest crittografico attesta i dati impiegati per calcolare i token, e permette ad utenti esterni di verificare successivamente la correttezza dei dati impiegati. Si ricorda che il digest è calcolato sulla base dei dati del modello utilizzato, e sui dati di consumo e di produzione energetica, e per questo permette agli utenti di verificare la correttezza di tutte queste informazioni.

Dal punto di vista tecnico, si osserva che per implementare questa strategia si impiegherà uno schema tipico basato sull'impiego integrato di funzioni hash e di chiavi crittografiche segrete.
In fase di creazione della comunità il Community Admin, dovrà inserire anche una chiave specifica per la comunità, questa chiave verrà memorizzata in una specifica tabella del Data Manager DB e verrà letta in fase di assegnazione dei token per creare il digest di commitment. La sicurezza di questo schema richiede l'impiego di una chiave differente per ogni digest generato, e l'utilizzo della stessa chiave per molteplici inserimenti rischia di rivelare informazioni riguardanti l'impiego dei consumi degli utenti. A questo scopo, si integra lo schema di commitment con uno schema di derivazione di chiavi che consente di calcolare a tempo di esecuzione chiavi univoche sulla base degli stessi dati inseriti. Per questo motivo, prima del calcolo del digest il Data Service Manager derivata una chiave unica sulla base dei parametri stessi. Si osserva inoltre che l'approccio di derivazione permette alla piattaforma di effettuare verifiche in modo

selettivo solo a specifici utenti o per specifici dati. In questa fase si propone un livello di profondità di due livelli: il primo livello deriva delle chiavi uniche per ogni utente a partire dalla chiave di comunità; il secondo livello deriva la chiave unica per il dato inserito a partire dalla chiave di un utente. Per ulteriori dettagli riguardo i criteri di progettazione e ulteriori informazioni sui criteri di applicazione dello schema è dettagliata nel documento "RDS - Strategia KDF v0.2" scritto da UNIMORE.

Come discusso durante i meeting di analisi dei requisiti, la gestione delle chiavi demandata al Community Admin rappresenta un compromesso per ottenere una soluzione prototipale sperimentale ma non è la soluzione più sicura e consigliata per ottenere i migliori livelli di sicurezza, in particolare perché le chiavi generate devono garantire livelli di sicurezza impiegabili in schemi crittografici (sequenza di bit random o pseudo-random di lunghezza dipendente dal livello di sicurezza richiesto dallo specifico schema). L'utilizzo di password o informazioni più deboli implica problemi di sicurezza nel progetto. Si considera che in questa fase di progetto "pilota" questa funzione serve a dimostrare le funzionalità della proposta, e in future versioni "in produzione" potrà essere sostituita tramite l'impiego di strategie standard più adatte. Tale scelta, permetterà di integrare in modo relativamente semplice un KMS di gestione chiavi nelle fasi future di progetto. Utilizzando un KMS, il Data Service Manager non dovrà più scrivere e leggere le chiavi dal Data Manager DB ma direttamente dal KMS e similmente, il Community Admin non dovrà più inserire manualmente le chiavi in fase di creazione di una comunità, queste, verranno cerate direttamente dal KMS.

2.7 Soluzioni tecnologiche e modalità di sviluppo

In questo paragrafo sono descritti i framework, le librerie e gli ambienti utilizzati per lo sviluppo delle singole componenti della piattaforma. Per ogni componente viene anche indicata la struttura del codice con la descrizione dei principali package e delle risorse statiche.
Tutto il codice consegnato come elaborato del progetto è corredato di documentazione dettagliata come da standard attuali.

2.7.1 DSM – Data Service Manager

2.7.1.1 Ambiente, Framework e librerie

Ambiente di sviluppo

Ambiente	Java
Compatibilità Java	Java 11
Dependency Management	Maven

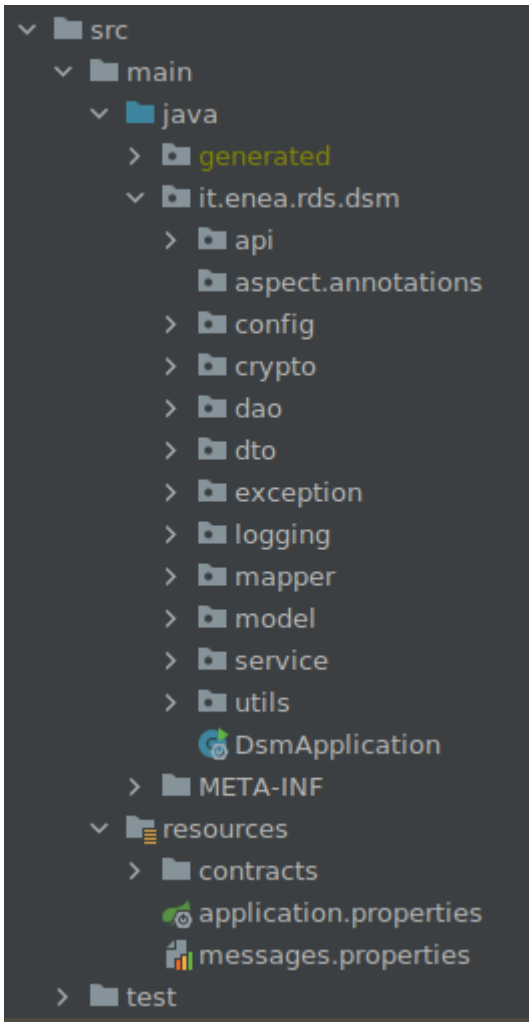
Framework e librerie usati per lo sviluppo

Framework	Spring
Boilerplate	Spring Boot 2.4.4
Layer autorizzativo	Spring Security 5.4.5
Connettore Blockchain	Web3j 5.0.0
Driver Database	org.postgresql 42.2.19
Gestione Token su standard JWT	com.auth0.java-jwt 3.15.0
Gestione testi/Stringhe	org.apache.commons 1.9
Libreria Crittografica	org.bouncycastle 1.68
Unit Test	Junit
Standard crittografici	Sha3, HMAC
Algoritmo di derivazione delle chiavi crittografiche	Custom

2.7.1.2 Struttura del codice

Di seguito è riportata la struttura del packaging di progetto, una descrizione dei principali package, delle risorse statiche, delle api e la rappresentazione del Data Model

2.7.1.3 Principali package



Package api: Contiene le classi (Controller) che espongono le API per la gestione del Data Service Manager.
Package dao: contiene le classi che implementano il modello ORM (Object-relational mapping) e relativi Repository (classi necessarie per l'esecuzione di query sul DB)
Package crypto: contiene le classi che implementano o supportano la gestione delle chiavi criptografiche
Package service: contiene le classi che implementano la business logic delle API
Package model: Contiene le classi che implementano gli algoritmi dei modelli e gli scheduler che ne consentono l'esecuzione

2.7.1.4 API

Community API

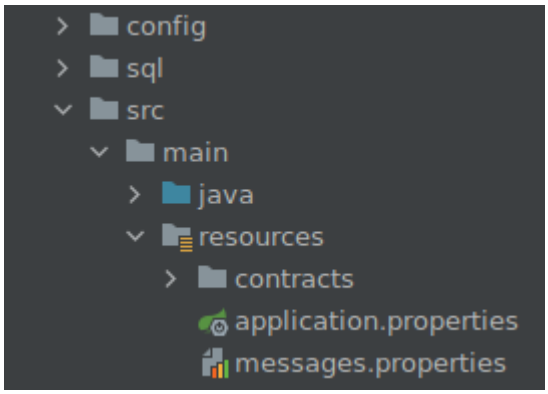
Create	Creazione di una comunità
Update	Aggiornmento dei dati di una comunità
Delete	Eliminazione di una comunità

retrieveAll	Lista delle comunità
retrieve	Dettagli di una singola comunità
assignModelsToCommunity	Assegna un modello a una comunità
disAssignModelsToCommunity	Cancella l'assegnazione di un modello a una comunità
activateModel	Attivazione di un modello su una comunità
deactivateModel	Disattivazione di un modello su una comunità
updateModelParameters	Aggiornamento dei parametri di un modello per una comunità

Model API

retrieveAllModels	Lista di tutti i modelli
retrieveModelParameters	Dettagli di un singolo modello
updateModelParameters	Aggiornamento dei parametri di scheduling di un modello
start	Attivazione globale di un modello
stop	Disattivazione globale di un modello

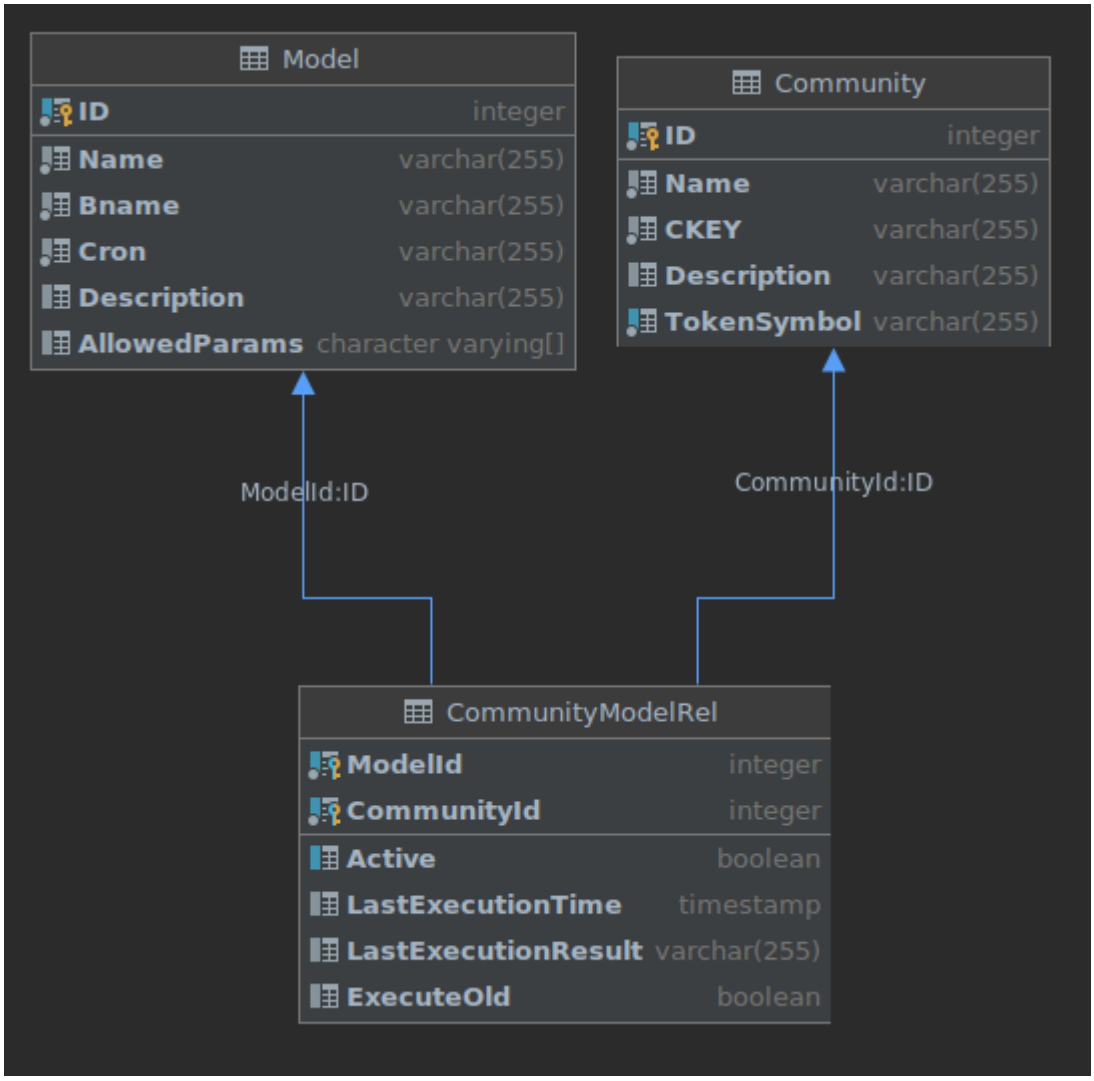
2.7.1.5 Descrizione delle risorse statiche



Sql: script per la creazione del database DMDB contenente dettagli configurativi dei modelli
Contracts: contiene gli smart contract utilizzati

2.7.1.6 DMDB Data Model

Di seguito si riportaa la struttura del DMDB, database utilizzato per la gestione dei modelli e la loro relazione con le comunità



2.7.2 Blockchain BESU

Con il termine Hyperledger Besu si indicano tutte le componenti software che compongono il network blockchain della soluzione.

2.7.2.1 Componenti:

- **Besu**: è un client Open Source scritto in Java che offre le funzionalità di interfacciamento verso la blockchain e, di conseguenza, di analisi e verifica sulla correttezza delle stesse. Besu implementa diversi meccanismi di validazione delle transazioni, tra i quali anche quello utilizzato dalla soluzione: IBFT 2.0.
- **IBFT 2.0**: è il meccanismo di validazione utilizzato dalla soluzione attraverso il quale i validatori raggiungono la super maggioranza nel definire quali transazioni sono legittime per essere aggiunte in un blocco e quindi, nella catena dei blocchi della blockchain.
- **Validator**: la soluzione utilizza un totale di quattro validatori scelti randomicamente ad ogni nuova proposta di blocco da aggiungere alla catena dei blocchi. Nello specifico, il validatore selezionato, impacchetta le nuove transazioni in un blocco e propone il blocco agli altri validatori. Quando il numero minimo dei validatori (nella soluzione 3) valida il blocco proposto, il blocco viene aggiungo.
- **EthSigner**: Ogni transazione, prima di poter essere inviata al client Besu, va firmata ed inviata ad un altro componente della blockchain l'EthSigner. L'EthSigner è un servizio che fa da proxy firmando e inoltrando le transazioni verso il client. Le chiavi utilizzate per firmale le transazioni possono essere salvate in svariati modi, la soluzione proposta prevede l'utilizzo di Metamask.

- **Metamask**: è un plug-in per browser, utilizzato dalla Dapp con il compito di memorizzare gli account degli utenti (chiave pubblica e chiave privata) e offrire un interfacciamento via RPC verso l'EthSigner.

2.7.2.2 Struttura del Codice

- **Config**: folder contenente i file di configurazione per le diverse componenti software
- **Docker-compose.yml**: file yml di sturt up di tutte le componenti in ambiente dockerizzato

Nel folder sono inoltre presenti cinque script utili ad interagire con l'applicazione

- **List.sh**: script di visualizzazione di tutti gli entpoint esposti dai diversi componenti
- **Remove.sh**: script di rimozione di tutti i container docker e la catena dei blocchi
- **Resume.sh**: script di riavvio dei contenitori docker
- **Run.sh**: script di avvio i conteiner docker
- **Stop.sh**: script per stoppare i container docker

2.7.3 Smart Contract Tokenization Stystem

Lo smart Contract Tokenization System è l'insieme di tutti gli Smart Contract dell'applicazione e delle loro interfacce.
Il linguaggio utilizzato per tutti gli smart contract è Solidity compilato con la versione 0.8.2 di solcjs e compatibili con versioni di solidity comprese tra 0.7 e 0.9.

2.7.3.1 Interfacce:

- **IAccessControl:** Definisce le interfacce:
  - creazione ruolo
  - assegnazione ruolo ad utente
  - revoca ruolo ad utente
  - se un utente ha uno specifico ruolo
  - implementa gli eventi di:
    - Creazione ruolo
    - Assegnazione Ruolo
    - Revoca Ruolo
- **ICommunity:** Definisce le interfacce:
  - Creazione comunità
  - Eliminazione comunità
  - implementa gli eventi di:
    - Comunità Creata
    - Comunità Eliminata
- **ICommunityUser:** Definisce le interfacce:
  - Assegnazione utente a comunità
  - Eliminazione utente a comunità
  - se un utente fa parte di una comunità
  - implementa gli eventi di:
    - Utente Aggiunto
    - Utente Rimosso.
- **IERC1203:** Definisce le interfacce:
  - Creazione token
  - Trasferisci token

    - Mint token
    - Token totali creati
    - Token per specifico utente
    - Implementa gli eventi di:
        - Creazione token
        - Trasferimento Token
- **IFee:** Definisce le interfacce:
    - Aggiornamento fee
    - Implementa gli eventi di “Fee aggiornata”
- **ILogModel:** Definisce le interfacce:
    - Logga modello
    - Implementa gli eventi di “Token Mint”
- **IModel:** Definisce le interfacce:
    - Definisci modello
    - Parametri di un modello
    - Implementa gli eventi di “nuovo modello”
- **IServiceIncentives:** Definisce le interfacce:
    - Aggiorna incentivo di servizio
    - Implementa gli eventi di “Incentivo di servizio aggiornato”
- **IWelcomeBonus:** Definisce le interfacce:
    - Aggiorna bonus benvenuto
    - Implementa gli eventi di “Bonus Benvenuto aggiornato”

2.7.3.2 Smart Contract:

- **Access Control:** Implementa l’interfaccia IAccessControl
- **ERC1203:** Implementa l’interfaccia IERC1203
- **Fee:** Implementa l’interfaccia IFee
- **LogModel:** Implementa l’interfaccia ILogModel
- **Model**: Implementa l’interfaccia IModel
- **ServiceIncentive:** Implementa l’interfaccia IServiceIncentive
- **WelcomeBonus:** Implementa l’interfaccia IWelcomeBonus
- **Community:** Implementa l’interfaccia ICommunity, ICommunityUser e espone le interfacce pubbliche per interagire con gli Smart Contract: Fee, ERC1203, WelcomeBonus e ServiceIncentive

2.7.3.3 Truffle

Il framework di sviluppo utilizza è stato Truffle, il quale offre un’ambiente automatizzato per:

- compilazione codice sorgente
- deploy degli smart contract su blockchain
- test degli smart contract
- trascodifica del bytecode degli smart contract in javascript

2.7.3.4 Struttura del Codice

- contract: folder contenente tutti gli smart contract e le interfacce
- docgen: folder contenente la documentazione generata automaticamente per le singole funzionalità degli smart contract
- migration: Folder contenente i file utili per poter deployare gli smart contract su blockchain
- test: file contenente i file di test degli smart contract

- docify.js: script di creazione della documentazione contenuta nel folder docgen
- package.json: file contenente la descrizione, le dipendenze del progetto e gli script di utility
- truffle-config.js: file di configurazione di truffle.

Nel package.json sono definiti 4 script utili ad interagire con l'applicazione:

- ganache: lancia una blockchain di test non persistente utile ad eseguire i test
- test: esegue gli script di test
- deploy: esegue il deploy degli smart contract sulla blockchain
- docify: esegue lo script docify.js per la generazione della documentazione degli smart contract.

3 Conclusioni

La piattaforma per comunità energetiche basata su blockchain è stata realizzata implementando tutte le specifiche tecniche di progetto dettagliate nel documento "Allegato tecnico per RDS – LA 1.47".
Rispetto ai requisiti progettuali non è stato però possibile integrare la piattaforma con l'Identity Gateway centralizzato per la gestione delle utenze in quanto non ancora disponibile. Per risolvere questa problematica e rendere il sistema utilizzabile al pieno delle sue funzionalità è stato implementato un Identity Provider dedicato per la gestione degli utenti e dei relativi ruoli.
In futuro sarà quindi necessario adeguare le componenti DAPP e Blockchain Tokenization System per interagire con Identity Gateway, permettendo così di centralizzare la gestione delle utenze.

La piattaforma permette agli utenti delle comunità energetiche di gestire i token presenti nel proprio "wallet" in modo semplice, intuitivo e trasparente, permettendone quindi lo scambio all'interno della comunità per usufruire di determinati servizi messi a disposizione. Per permettere agli amministratori di comunità di gestire la remunerazione dei token sono stati implementati dei modelli energetici configurabili tramite specifici parametri in modo indipendente per le singole comunità. I modelli gestiti dalla piattaforma sono:

- Modello autoconsumo
- Modello assenza di picchi
- Modello autoconsumo – assenza di picchi
- Modello remunerativo Billing Schema 2

Tali modelli sono descritti nel documento "Annex1 – Dynamic Business Model and Tools".

La piattaforma è stata progettata e realizzata con caratteristiche modulari con l'obiettivo di facilitare sviluppi futuri e l'integrazione con altri sistemi del progetto RDS.
Durante lo svolgimento delle attività progettuali sono emersi alcuni scenari che potrebbero essere oggetto di sviluppi futuri:

- Realizzazione di un connettore blockchain per estrapolare dati ed inserirli in un database per gestire logiche di elaborazione dati e condivisione con altre piattaforme
- Aggiunta indicatori monitoraggio della token economy della comunità
- Funzionalità aggiuntive amministratore token-economy, inclusa visualizzazione indicatori del punto sopra
- Funzionalità aggiuntive token-holder su token-economy, inclusa visualizzazione indicatori
- Integrazione con piattaforma market place
- Integrazione con cruscotto di comunità