Università di Bologna

Corso di Blockchain and Cryptocurrencies
Dipartimento di Informatica

BlueWoodchuck

Blockchain of custody

Crespan Lorenzo [0001038888]

Cosenza Alessandra [0001052401]

30 marzo 2023

Referente: *lorenzo.crespan@studio.unibo.it*

Indice

1 Introduzione **1**
1.1 Introduzione progetto 1
1.2 Catena di custodia 1
1.3 Obiettivo del progetto 2

2 Tecnologie utilizzate **4**
2.1 Tecnologie utilizzate 4

3 Implementazione **6**
3.1 Architettura e logica del progetto 6
3.2 Smart-Contract 7
3.2.1 Strutture dati Smart-Contract 8
3.2.2 Funzioni Smart-Contract 10
3.3 Applicazione decentralizzata 12

4 Analisi costi **14**
4.1 Gas Fees 14
4.2 Costi specifici 15
4.2.1 Risultati 15
4.3 Considerazioni 17

5 Conclusioni **18**
5.1 Criticità riscontrate 18
5.2 Potenziali soluzioni 19
5.3 Lavori futuri 19

1 Introduzione

1.1 Introduzione progetto

Il seguente documento riporta le informazioni relative allo sviluppo di un sistema per la tracciabilità di artefatti nell'ambiente giuridico (penale e civile) andando a traslare le caratteristiche e funzionalità della "catena di custodia" che ad oggi rappresenta l'unico strumento utilizzabile per tale scopo.

Ogni scelta progettuale ed implementativa sarà illustrata in modo approfondito nelle seguenti sezioni.

1.2 Catena di custodia

La catena di custodia rappresenta uno dei processi più critici per la presentazione di prove all'interno di una corte di giustizia. Lo scopo principale di tale documento/procedura è quello di fornire una garanzia in termini di **integrità e affidabilità delle informazioni raccolte**.

Nello specifico con tale termine ci si riferisce alla documentazione accurata e sistematica relativa al percorso che una prova segue dalla sua raccolta fino alla sua presentazione in corte (Figura 1). In altri termini, si impone che ogni passaggio dal processo di raccolta, al trasporto, all'analisi fino alla conservazione deve essere registrato e tracciato in modo da garantire che la prova sia sempre sotto controllo e non vi sia margine di dubbio per potenziali alterazioni, contaminazioni o manipolazioni di quest'ultimo.

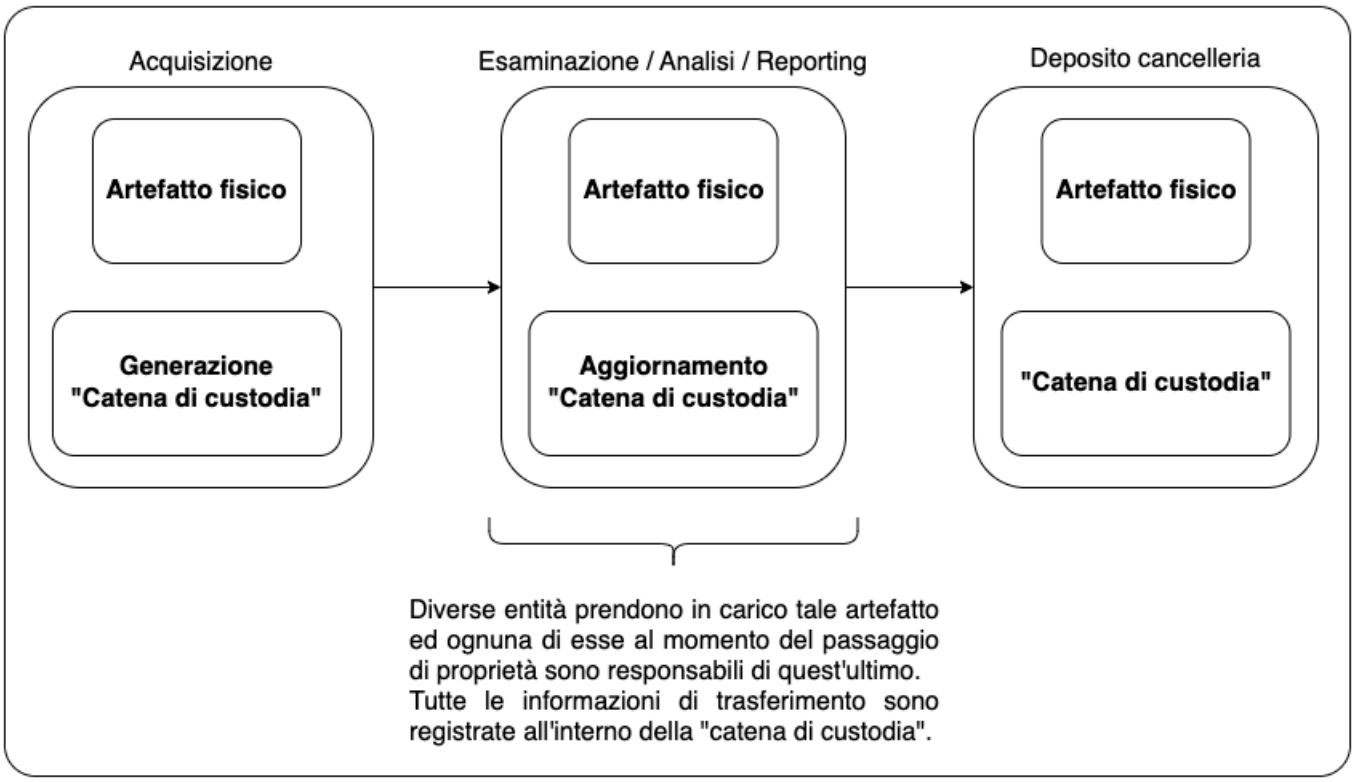


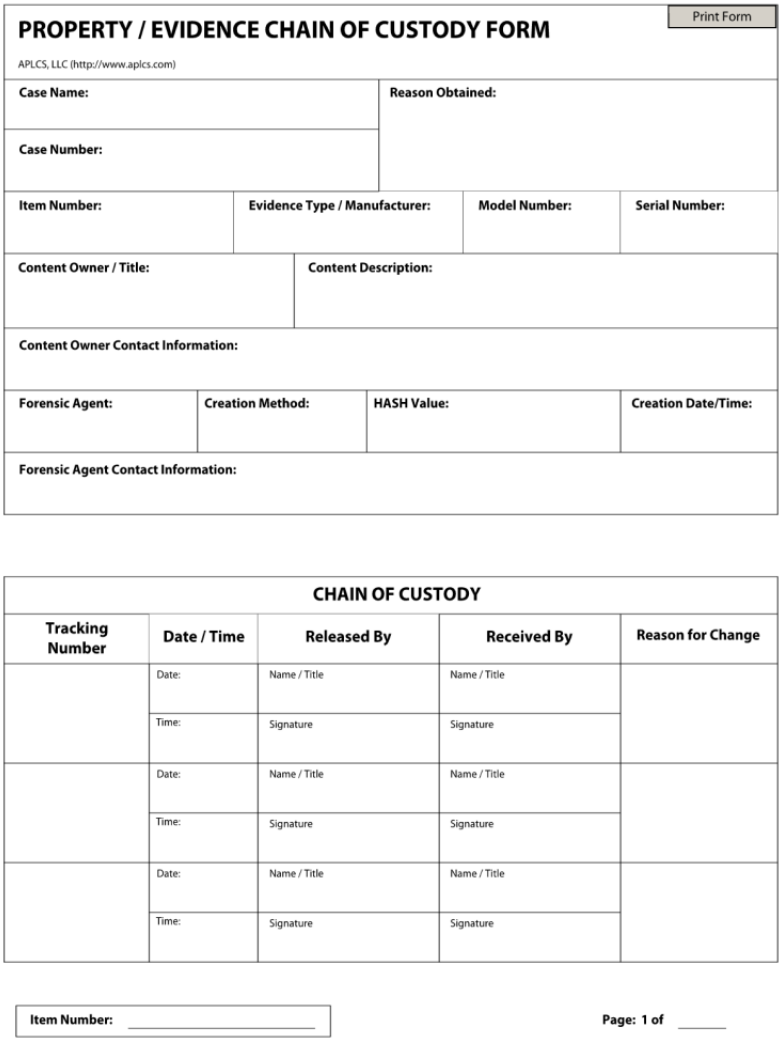
Figura 1: Schema della catena di custodia.

Un processo giuridico per terminare correttamente necessita della collaborazione di diverse figure professionali, tra cui polizia, tecnici forensi ed avvocati, e tale aspetto è particolarmente critico, nonostante la presenza di specifiche norme e linee guida che garantiscono teoricamente la validità delle prove raccolte. In tale ambito, l'utilizzo di un sistema basato su Blockchain permetterebbe di facilitare il lavoro andando, ad esempio, a diminuire gli effetti di errore umano.

1.3 Obiettivo del progetto

L'obiettivo del progetto è stato quello di creare una soluzione basata su Blockchain per garantire l'integrità e la tracciabilità degli artefatti giuridici, in modo tale da ridurre il rischio di alterazione delle prove, migliorare la trasparenza e l'affidabilità del processo di produzione di una prova giuridica.

Per ottenere tale risultato si è deciso di trasporre la struttura della catena di custodia (Figura 2) all'interno di uno Smart-Contract e fornire le relative funzionalità.



PROPERTY / EVIDENCE CHAIN OF CUSTODY FORM — Print Form

APLCS, LLC (http://www.aplcs.com)

Case Name:		**Reason Obtained:**	
Case Number:			
Item Number:	**Evidence Type / Manufacturer:**	**Model Number:**	**Serial Number:**
Content Owner / Title:	**Content Description:**		
Content Owner Contact Information:			
Forensic Agent:	**Creation Method:**	**HASH Value:**	**Creation Date/Time:**
Forensic Agent Contact Information:			

CHAIN OF CUSTODY

Tracking Number	Date / Time	Released By	Received By	Reason for Change
	Date:	Name / Title	Name / Title	
	Time:	Signature	Signature	
	Date:	Name / Title	Name / Title	
	Time:	Signature	Signature	
	Date:	Name / Title	Name / Title	
	Time:	Signature	Signature	

Item Number: _____ **Page: 1 of** _____

Figura 2: Form della catena di custodia.

In relazione alla Blockchain e nello specifico allo Smart-Contract si intende implementare le seguenti funzionalità.

- **Sistema di accesso limitato ai partecipanti di una rete Ethereum**
  Permettere solo ad utenti registrati alla rete di Ethereum di poter accedere e operare con il loro portafoglio di riferimento.
- **Creazione di un form per un artefatto giudiziario**
  Permettere agli utenti di poter creare una prova giuridica andando di fatto a creare il form relativo.
- **Ricerca e visualizzazione di un form relativo ad un artefatto giudiziario**
  Permettere agli utenti di poter ricercare in modo libero uno o più form e di poterne visualizzare le informazioni.
- **Gestione e trasferimento di un form relativo ad un artefatto giudiziario**
  Permettere agli utenti di gestire al meglio il form, tenendo sempre in considerazione il ruolo centrale che quest'ultimo ricopre all'interno dell'ambiente giuridico e delle implicazioni legali attribuite al detentore.
- **Generazione documentazione di un form relativo ad un artefatto giudiziario**
  Permettere, vista la natura legale del progetto, di poter generare dei documenti adeguati all'associazione form/artefatto (generazione di un'etichetta da poter apporre sull'oggetto per ricercare all'interno della rete il form corrispondente) ed inserimento all'interno di report.

Oltre alla creazione e messa in funzione del sistema, il progetto si focalizza anche in una valutazione del prodotto ottenuto con una riflessione delle potenziali criticità ed alternative alle componenti sviluppate.

2 Tecnologie utilizzate

2.1 Tecnologie utilizzate

Di seguito sono riportate le tecnologie utilizzate per lo sviluppo ed implementazione del progetto.

Lista delle tecnologie principali	
Nome	**Versione**
Truffle	5.7.7
Ganache	7.7.5
React.js	18.2.0
Web3.js	1.8.1
Metamask	2.4.1
Solidity	0.8.17

Tabella 1: Elenco delle principali tecnologie e relative versioni utilizzate.

- **Truffle**
  Truffle è un framework open-source per lo sviluppo di applicazioni Blockchain basate su Ethereum. Lo scopo di tale prodotto è quello di offrire agli sviluppatori uno strumento completo che consenta la creazione, lo sviluppo, la simulazione e la distribuzione di applicazioni decentralizzate che sfruttano la Blockchain, Tali applicazioni sono definite nel settore come dApp (Applicazioni decentralizzate).

- **Ganache**
  Ganache è un ambiente di sviluppo Blockchain che consente agli sviluppatori di creare, testare applicazioni decentralizzate su rete Ethereum. Nello specifico, Ganache fornisce una Blockchain Ethereum privata locale, consentendo agli sviluppatori di testare le proprie applicazioni senza dover accedere alla rete principale o di test di Ethereum.

- **React.js**
  React.js è una libreria open-source di JavaScript utilizzata per la creazione di interfacce utente dinamiche per le applicazioni web.

- **Web3.js**
  Web3.js è una libreria open-source di JavaScript utilizzata per interagire con la Blockchain Ethereum. Web3.js fornisce un'interfaccia semplificata per interagire con i nodi della rete Ethereum, inviare transazioni, interagire con gli Smart-Contract e recuperare i dati della Blockchain.

- **Metamask**
  Metamask è una estensione disponibile sui principali browser che consente agli utenti di interagire con applicazioni decentralizzate basate su Ethereum utilizzando il proprio browser web. Metamask funziona come un portafoglio digitale che consente agli utenti di gestire le proprie criptovalute e di interagire con la Blockchain Ethereum.

- **Solidity**
  Solidity è un linguaggio di programmazione ad alto livello utilizzato per la scrittura di contratti intelligenti sulla Blockchain Ethereum. Solidity è stato sviluppato da Ethereum Foundation ed è basato sulla sintassi di C++, Python e JavaScript.

  Solidity consente agli sviluppatori di creare Smart-Contract sulla Blockchain Ethereum, utilizzabili per eseguire transazioni, gestire asset digitali e automatizzare processi finanziari.

3 Implementazione

3.1 Architettura e logica del progetto

Nello sviluppo del progetto si è fatto riferimento al seguente schema “Figura 3” per la comunicazione tra le due componenti principali, l’applicazione decentralizzata e lo Smart-Contract.

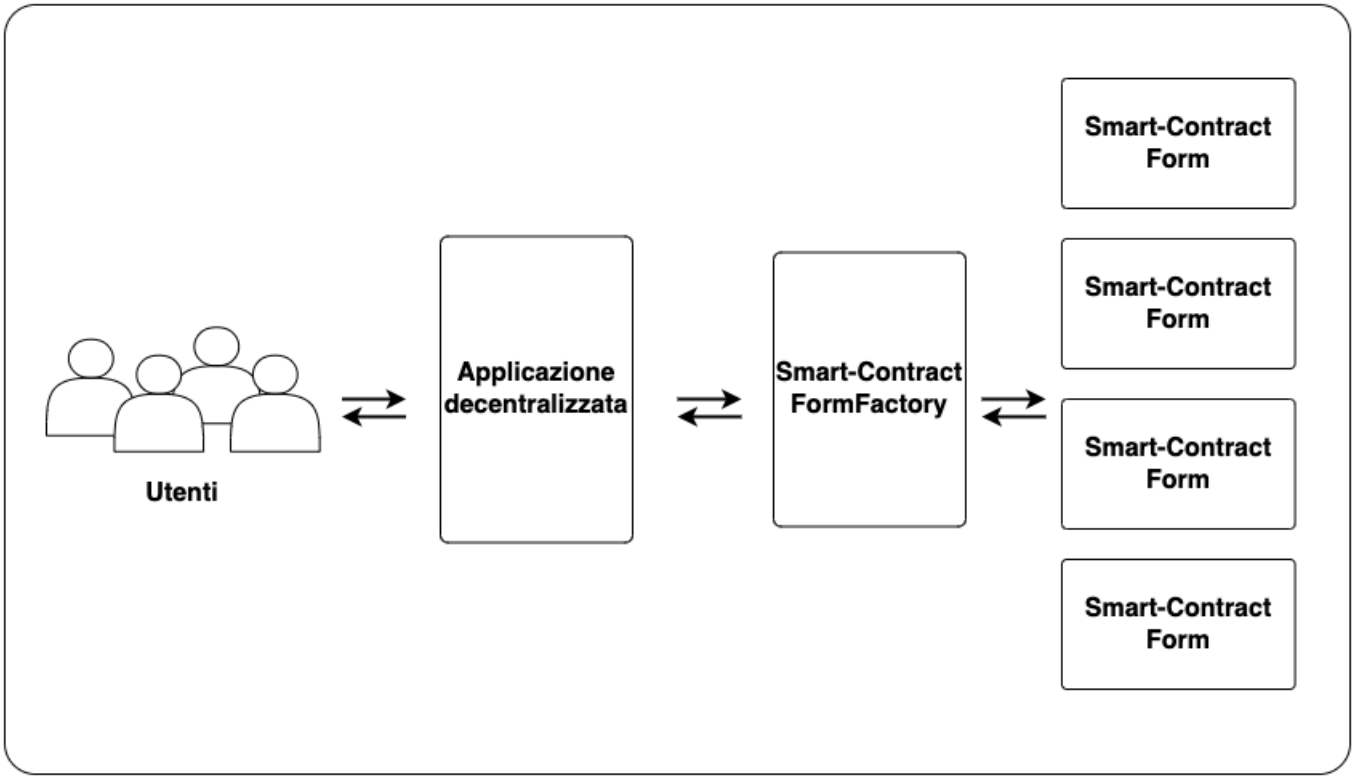


Figura 3: Schema di comunicazione delle componenti principali del progetto.

Nelle seguenti sezione si andranno ad analizzare nello specifico le strutture dati e le funzioni utilizzate per l’applicazione decentralizzata e lo Smart-Contract, ponendo una particolare attenzione alle interazioni fra queste ultime.

3.2 Smart-Contract

Lo Smart-Contract rappresenta nel progetto l'elemento più importante dal momento che permette la generazione e la gestione delle informazioni relativi ad ogni artefatto giuridico.

Dal punto di vista progettuale sono stati creati due contratti: "FormFactory" e "Form" (Figura 4).

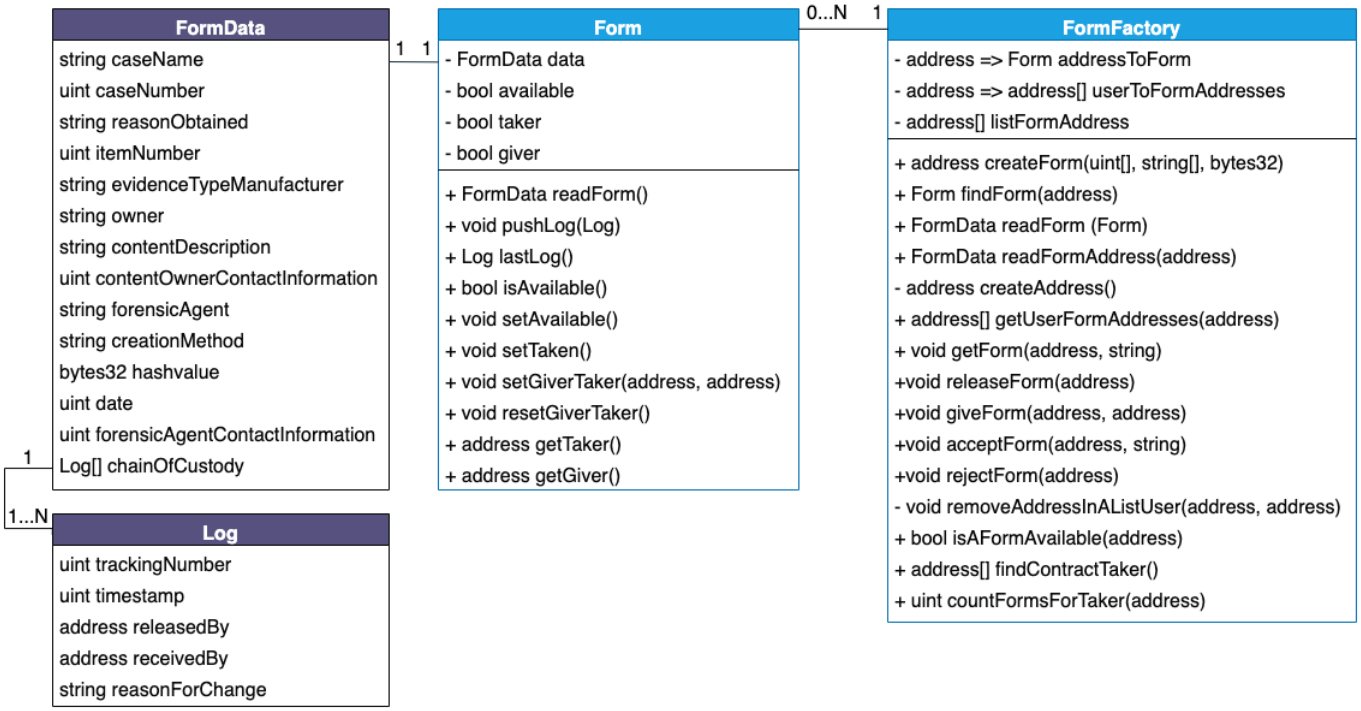


Figura 4: Schema strutture dati e funzioni di degli Smart-Contract.

Lo Smart-Contract "FormFactory", come suggerisce il nome, si occupa di creare e gestire lo stato degli altri contratti "Form". Lo scopo di quest'ultimi, invece, è quello di memorizzare le informazioni e mettere a disposizione del "FormFactory" le funzioni di modifica di alcune variabili utili al trasferimento di proprietà.

3.2.1 Strutture dati Smart-Contract

Per la memorizzazione delle informazioni si è creata una struct all'interno del contratto "Form".

Lista dei parametri dello Smart-Contract "Form"

```
// Struct to store the information of the form.
struct FormData {
    string caseName;                        // Name of the case
    uint caseNumber;                        // Number of the case
    string reasonObtained;                  // Reason of the evidence
    uint itemNumber;                        // Number of the item
    string evidenceTypeManufacturer;        // Type evidence manufacturer
    string owner;                           // Name owner evidence

    string contentDescription;              // Description of the evidence
    uint contentOwnerContactInformation;    // Contact information of owner
    string forensicAgent;                   // Name of agent
    uint forensicAgentContactInformation;   // Contact information of agent

    bytes32 hashValue;                      // Hash value of the evidence
    uint date;                              // Date of the evidence

    Log[] chainOfCustody;                   // Chain of custody
}
```

Tra i campi che compongono la struct "FormData" vi è "chainOfCustody", un array di struct "Log" composto a sua volta dai seguenti campi:

Lista dei parametri dello Smart-Contract "Form"

```
// Struct to store the information of the chain of custody.
struct Log {
    uint trackingNumber;        // Tracking number of the form
    uint timestamp;             // Epoch time of the change
    address releasedBy;         // Address entity released form
    address receivedBy;         // Address entity received form
    string reasonForChange;     // Reason for the change
}
```

Lo scopo di "Form" è quello di gestire le struct presentate. Lo scopo delle due strutture è quello di memorizzare i dati di un artefatto giuridico e di permettere, in caso di necessità, la ricostruzione del passaggio di proprietà fino al suo creatore.

Oltre alla struttura descritta, sono presenti diverse variabili che servono a gestire il passaggio di proprietà:

Lista dei parametri dello Smart-Contract “Form”

```
address private taker;
address private giver;
bool private available;
```

Nello specifico, l’attributo “available” serve ad indicare se il contratto è proprietà di qualcuno o meno, invece “taker” e “giver“ sono utilizzati per gestire l’invio e ricezione di un form tra due utenti. Maggiori dettagli su come quest’ultimi sono utilizzati e come viene aggiornata la catena di custodia sono dati nella sottosottosezione 3.2.2.

La gestione dei “Form” è affidata al contratto “FormFactory” che si occupa di creare, contenere e indicizzare gli oggetti (in altri termini i contratti). Per ottenere il risultato desiderato si è utilizzato un mapping che attraverso una chiave di tipo address permette di recuperare le informazioni form desiderato.

Lista dei parametri dello Smart-Contract “FormFactory”

```
// Mapping to store the forms for the evidence of the cases.
mapping(address => Form) private addressToForm;

// Mapping to store the addresses of forms currently in charge of user.
mapping(address => address[]) private userToFormAddresses;

// Array to store the addresses of forms for the evidence of the cases.
address[] private listFormAddress;
```

Maggiori dettagli su come quest’ultimi sono utilizzati e come avviene la generazione degli indirizzi sono dati nella sottosottosezione 3.2.2.

3.2.2 Funzioni Smart-Contract

Di seguito sono riportate le funzioni del contratto "Form".

- **FormData readForm()**
  Ritorna l'intera struttura dati del singolo form, che in quanto privata non può essere letta altrimenti.
- **void pushLog(log)**
  Inserisce un oggetto log all'interno della chainOfCustody del form. È l'equivalente della firma su una catena di custodia fisica.
- **Log lastLog()**
  Restituisce l'ultimo log della chainOfCustody del form. È l'equivalente di leggere l'ultima riga di una catena di custodia fisica.
- **bool isAvailable(), void setAvailable(), void setTaken()**
  Getter e setter dell'attributo available.
- **void setGiverTaker(address, address), address getTaker(), address getGiver()**
  Getter e setter degli attributi giver e taker.
- **void resetGiverTaker()**
  Ripristina i campi riservati all'invio di un form da un perito a un altro.

Di seguito sono riportate le funzioni del contratto "FormFactory".

- **address createForm(uint[], string[], bytes32)**
  Permette di creare un nuovo contratto di tipo Form a partire dai dati passati come parametri, creando poi anche un indirizzo univoco per indicizzare il contratto all'interno di FormFactory.
- **Form findForm(address), FormData readForm(Form), FormData readFormAddress (Form)**
  Ricerca di form con differenti alternative di tipi di ritorno e di parametri.
- **address createAddress()**
  Generazione di un indirizzo univoco; usata per indicizzare i contratti all'interno di FormFactory.
- **address[] getUserFormAddresses(address)**
  Restituisce l'insieme di indici dei form a carico di un perito a partire dal suo indirizzo univoco.
- **void getForm(address, string)**
  Permette di segnalare l'acquisizione di un form a partire dal suo indirizzo e con una ragione (passata come stringa). Questo, dopo aver controllato che il form sia effettivamente disponibile, include l'aggiornamento dei dati all'interno di FormFactory, l'aggiunta di un Log nella catena di custodia.

- **void releaseForm(address)**
  Permette di rilasciare un form, rendendolo quindi disponibile, se se ne è proprietari. Questo aggiornerà anche i dati all'interno di FormFactory.

- **void giveForm(address, address)**
  Permette di inviare un contratto ad un altro perito. Così facendo, il form ha aggiornati nel proprio contratto i campi dedicati (giver e taker), poi il perito indicato come destinatario potrà accettare o rifiutare il passaggio del form tramite le funzioni dedicate che seguono.

- **void acceptForm(address, string)**
  Permette di accettare un contratto a seguito di una ragione passata come stringa, se il richiedente è segnalato come taker all'interno del form (quindi ha una richiesta a suo carico).

- **void rejectForm(address)**
  Permette di rifiutare l'invio di un contratto, se colui che invoca la funzione è segnalato come taker all'interno del form; questo resetta i campi giver e taker.

- **void removeAddressInAListUser(address, address)**
  Funzione interna per aggiornare I dati all'interno di FormFactory; in particolare, questa permette di segnalare come un form non sia più a carico di un certo perito.

- **bool isAFormAvailable(address)**
  Restituisce un booleano che indica se un form (indicato tramite il suo indirizzo univoco) è disponibile o meno.

- **address[] findContractTaker()**
  Restituisce l'insieme di form in cui il chiamante è indicato come taker nel relativo contratto.

- **uint countFormsForTaker(address)**
  Funzione interna per la definizione dell'array della funzione sopra; restituisce infatti il numero di form in cui il chiamante è definito come taker.

3.3 Applicazione decentralizzata

L'applicazione decentralizzata implementata in React.js si compone di diverse componenti che permettono di interagire con lo Smart-Contract.

Dal punto di vista tecnico è stata implementata la seguente struttura:

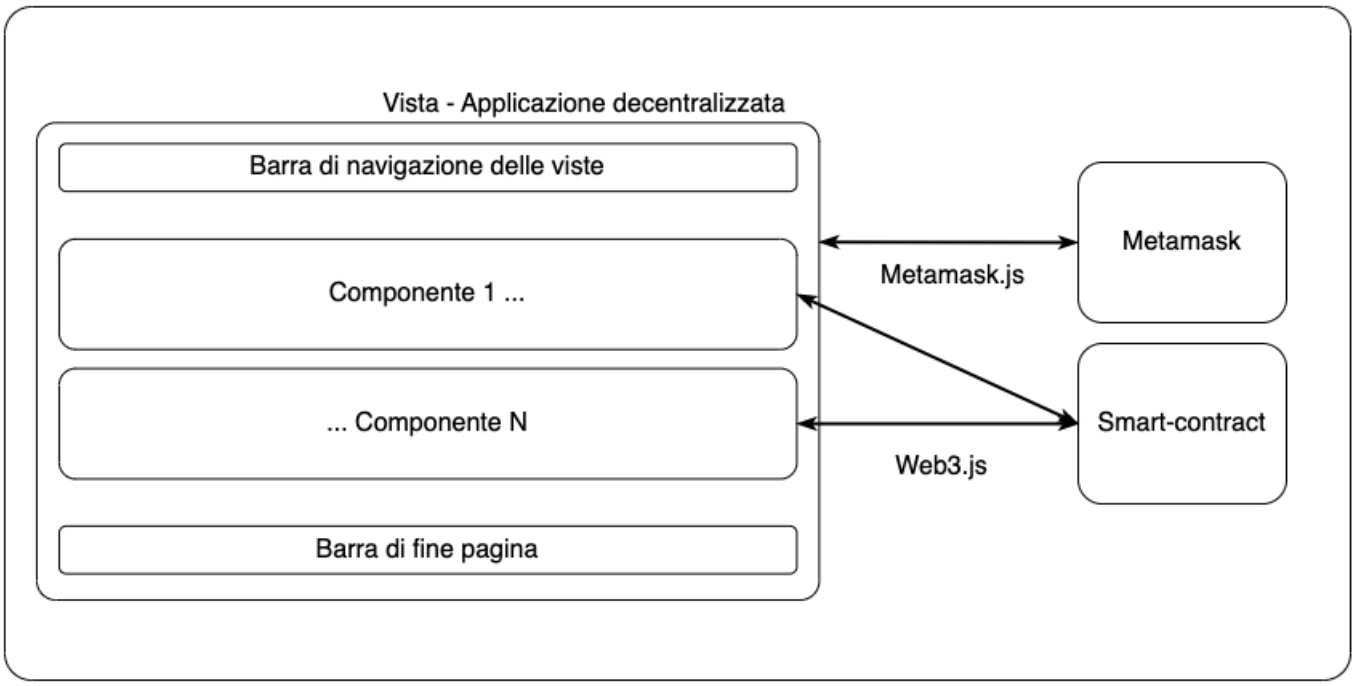


Figura 5: Architettura del progetto.

La schermata di login attraverso le API fornite da Metamask permette di verificare la presenza di quest'ultima ed in caso di riscontro positivo avvia la fase di verifica delle credenziali. A login effettuato, si è reindirizzati nella schermata di homepage.

Tra tutte le seguenti pagine la struttura seguita per la loro creazione è la medesima.

Dal punto di vista implementativo le componenti che maggiormente evidenziano l'interazione tra l'applicazione decentralizzata e lo Smart-Contract sono le seguenti:

- **coreFormGenerate.js**
  Il componente è costituito da una serie di campi che devono essere compilati per permettere il richiamo della funzione che a sua volta avvierà la transazione con lo Smart-Contract.

  Mediante il richiamo della funzione "createForm", a transazione terminata con successo si effettua una ricarica della pagina e dopo la validazione del blocco, contenete la transazione, sarà possibile visionare il contratto appena creato nella sezione di visualizzazione dei form in carico. In caso di errore nella transazione viene notificato il problema di generazione attraverso un popup.

- **coreFormList.js**
  Il componente è costituito da un blocco centrale che permette di visualizzare una serie di se-

zioni contenenti le informazioni dei contratti attualmente il carico (il medesimo componente è utilizzato in altre pagine).

Per la generazione degli elementi che lo compongono viene effettuata una chiamata allo Smart-Contract della funzione “getUserFormAddresses”, la quale permette di recuperare gli indirizzi dei contratti in carico ad un dato utente.

- **coreFormShow.js**
  Il componente che permette la visualizzazione delle informazioni relative ad un dato form. Mediante tale schermata è possibile richiamare le funzioni per operare sul contratto (renderlo disponibile al trasferimento pubblico o con un utente specifico), scaricare una copia in formato PDF dell’intero form, per poterla inserire come documentazione negli atti, oppure ottenere una versione minimale contenente il codice QR e delle informazioni per poterla collocare sull’artefatto giuridico.

- **coreSendRequest.js e coreRequest.js**
  Le componenti permetttono la gestione della transazione senza la necessità di rendere pubblica la possibilità di acquisizione di un form. La prima componente permette di indicare un utente destinatario ed il relativo form da inviare. A transazione avvenuta, il mittente riceve nella relativa sezione la richiesta di acquisizione che se accettata porta all’effettivo trasferimento.

- **coreSearch.js**
  La componente permette di effettuare una ricerca su “FormFactory” sulla base di una data di threshold per poter ottenere una lista di contratti coerenti secondo tale parametro. La ricerca è un esempio di potenziale ricerca implementabile anche con altri parametri caratteristici dei contratti creati.

4 Analisi costi

4.1 Gas Fees

Il concetto di Gas nella rete Ethereum è fondamentale per comprendere come funzionano le transazioni sulla piattaforma. Con tale termine, si fa riferimento all'unità di misura utilizzata per quantificare lo sforzo computazionale necessario per eseguire operazioni specifiche sulla rete di Ethereum.

In altri termini, ogni operazione eseguita sulla rete Ethereum richiede una certa quantità di gas per essere completata. Questo perché le transazioni vengono elaborate dai nodi della rete, i quali richiedono risorse computazionali per eseguire i calcoli necessari (Figura 6).

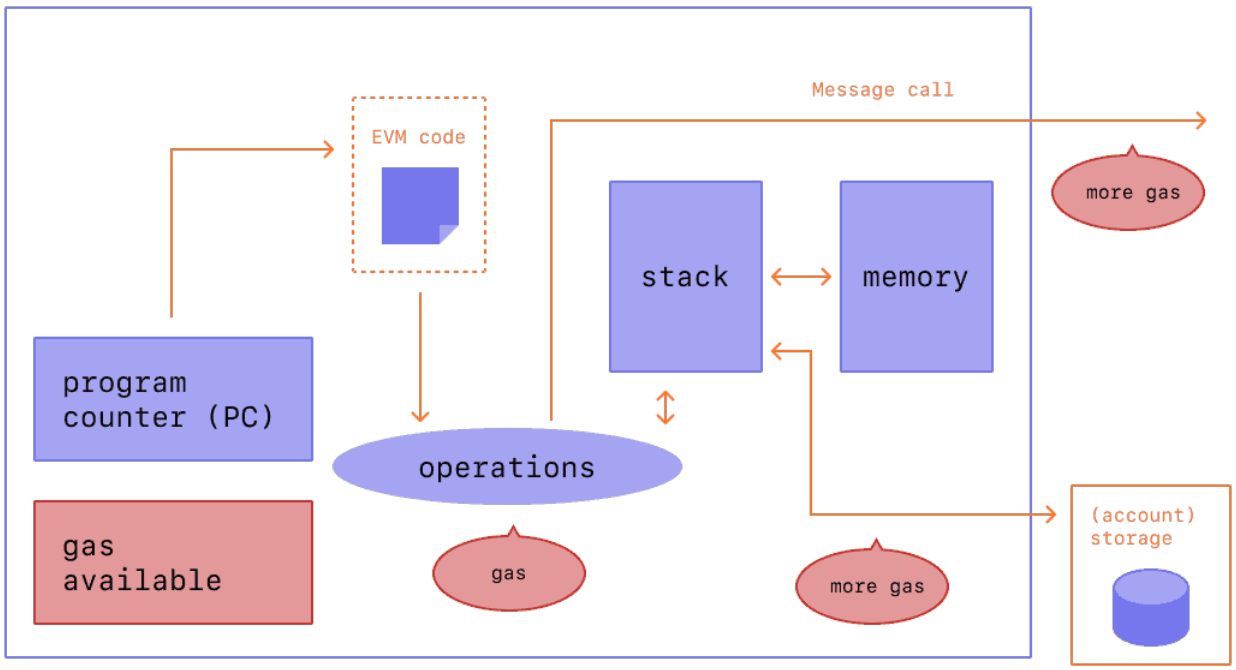


Figura 6: Calcolo delle Gas Fees nella rete Ethereum.

Tale sistema è necessario per disincentivare attività malevole come ad esempio lo spam di operazioni sulla rete, oppure la generazione di cicli infiniti accidentali o meno, oltre ad altri sprechi di calcolo nel codice.

Dal punto di vista implementativo, Solidity introduce numerose keyword per poter minimizzare il costo se vengono rispettati determinati parametri. Prendendo in esempio quanto implementato per lo sviluppo del progetto, la ricerca dei form, in cui un utente è segnalato come taker, richiede due cicli completi di lettura della blockchain, ma dal momento che non portano ad una modifica (ed alla generazione di una transazione) il loro costo effettivo è nullo.

L'utilizzo di funzioni che presentano le keyword "view" o "pure" permettono di implementare funzionalità senza andare ad intaccare il costo.

4.2 Costi specifici

Nel contratto sviluppato, vi è la presenza di diverse funzioni che variano sugli effetti. Alcune funzioni necessitano della generazione di una transazione che deve essere salvata all'interno di un blocco (come ad esempio la generazione o il rilascio/acquisizione di un contratto), mentre altre portano ad operazioni che non pesano sul portafoglio dell'utente (come ad esempio la ricerca).

Per effettuare un'analisi empirica per la valutazione dei seguenti aspetti:

- **Costi inserimento form**
  Questo richiede una particolare cura in quanto la presenza di campi stringa rischia di influire sui costi a prescindere dalla quantità di informazioni nella catena. Perciò è nostra attenzione utilizzare sempre gli stessi (o quasi) campi nella creazione dei form e tenere traccia di eventuali cambiamenti dei costi.

- **Costi transazioni di form**
  Poiché la cessione e l'acquisizione di un form ha un costo relativamente fisso (ricerca di un form nel mapping e modifica di un campo), ci aspettiamo che questo costo resti invariato nel corso del tempo.

- **Tempi di ricerca**
  Questa sezione rientra nel caso in cui i costi siano puramente computazionali, infatti vengono utilizzate soltanto funzioni di tipo ""view".

Per i nostri il test verranno creati 30 form e saranno riportati i dati ottenuti.

4.2.1 Risultati

Sulla base di quanto riportato nella Figura 7 si è osservato quanto segue:

- **Costi inserimento form**
  L'inserimento dei form sembra non variare nel corso del tempo: infatti, le operazioni da svolgere sono le medesime a prescindere dalla mole di dati pre-esistente e il costo è sempre di 0,05770974 ETH. L'unica differenza è nella creazione del primo form di un utente (0,05860974 ETH): infatti, questo comporta la creazione di un nuovo array in userToFormAddresses all'indice di tale user e ha un costo leggermente maggiore (0,0009 ETH di differenza pari a 1,50€ su un totale di 90€ sulla base dell'attuale rapporto di conversione).

- **Costi transazioni di form**
  L'acquisizione di un form ha un costo costante (0,00396852 ETH), in quanto si effettua una push dell'address del form nel proprio indice utente e segnala tale form come acquisito.

  Per quanto riguarda il rilascio di un form, invece, i costi sono direttamente proporzionali alla quantità di form in proprio possesso: infatti, il rilascio di un form comporta l'eliminazione del relativo address all'interno del proprio array e questa operazione prevede di scorrere tutti i propri form alla ricerca di quello da eliminare. I costi riscontrati variano leggermente a seconda della posizione del form nell'array.

- **Tempi di ricerca**

  I tempi di ricerca, come previsto, aumentano direttamente con il numero dei form esistenti.

Numero Form	Costo Previsto	Costo effettivo	Rilascio 1°	Rilascio 2°	Rilascio N - 1	Rilascio N	Acquisizione	Tempo
1	0,05861	0,03907316	0,00116484	Nan	Nan	Nan	0,00426852	156
2	0,05771	0,03847316	0,00154884	0,00150518	Nan	Nan	0,00396852	269
3	0,05771	0,03847316	0,00154884	0,00150518	Nan	0,00154552	0,00396852	431
4	0,05771	0,03847316	0,00154884	0,00150518	0,00162952	0,00158586	0,00396852	474
5	0,05771	0,03847316	0,00154884	0,00150518	0,00166986	0,0016262	0,00396852	658
6	0,05816	0,03847316	0,00154884	0,00150518	0,0017102	0,00166654	0,00396852	904
7	0,05771	0,03847316	0,00154884	0,00150518	0,00175054	0,00170688	0,00396852	901
8	0,05771	0,03847316	0,00154884	0,00150518	0,00179088	0,00174722	0,00396852	1102
9	0,05771	0,03847316	0,00154884	0,00150518	0,00183122	0,00178756	0,00396852	1233
10	0,05771	0,03847316	0,00154884	0,00150518	0,00187156	0,0018279	0,00396852	1391
15	0,05771	0,03847316	0,00154884	0,00150518	0,00207326	0,0020296	0,00396852	2238
20	0,05771	0,03847316	0,00154884	0,00150518	0,00227496	0,0022313	0,00396852	2462
30	0,05771	0,03847316	0,00154884	0,00150518	0,00267836	0,0026347	0,00396852	3404

Figura 7: Tabella risultati.

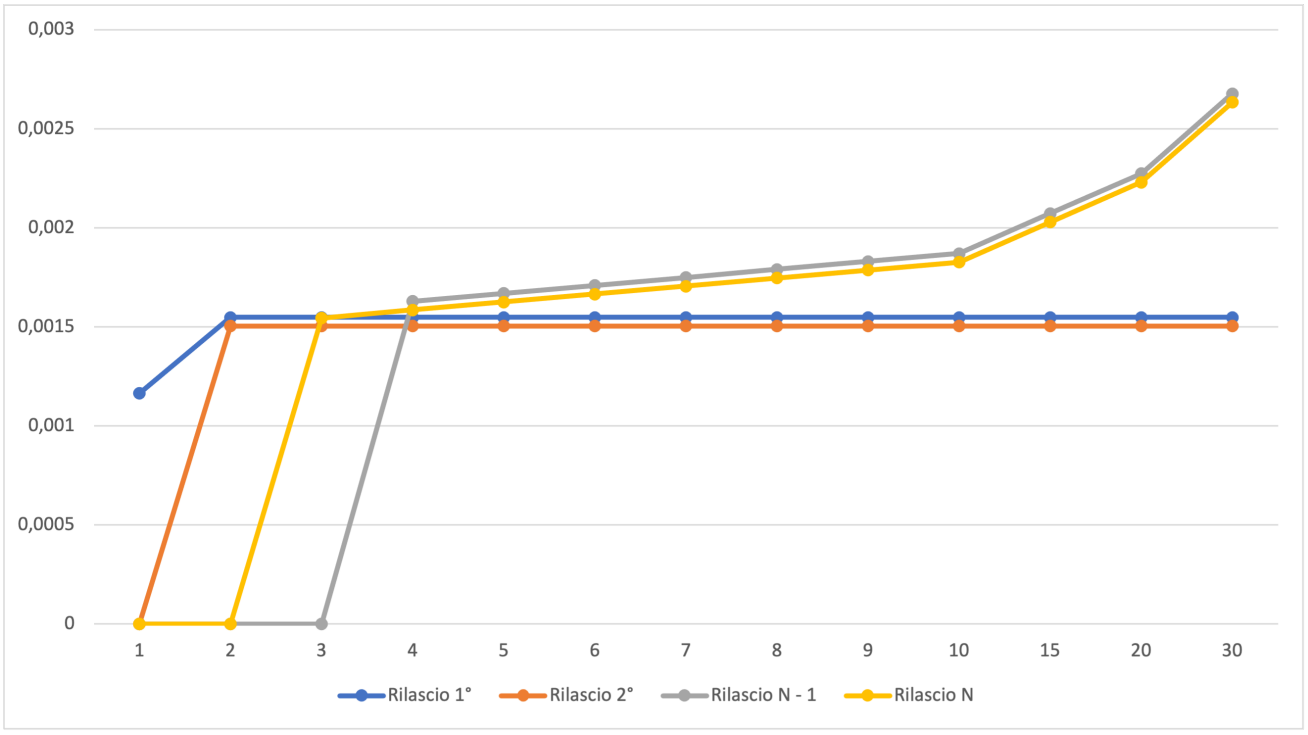


Figura 8: Costi di rilascio di un Contratto.

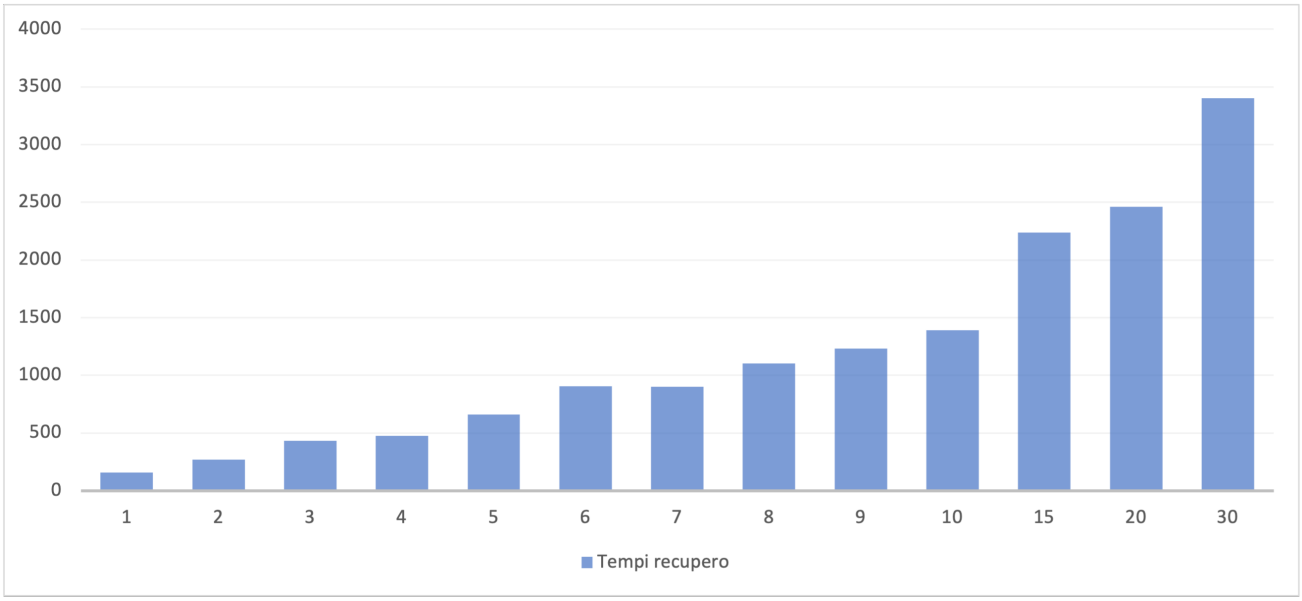


Figura 9: Tempi di recupero dei Contratti.

4.3 Considerazioni

I costi riscontrati sono accettabili: la transazione più alta è quella relativa alla registrazione di un form nell'array, ma dopo di questa tutte le altre operazioni hanno un costo molto più contenuto, che rende il sistema conveniente da un punto di vista economico. Ovviamente il sistema è ottimizzabile, ma le potenziali modifiche applicabili saranno discusse nel capitolo conclusivo.

5 Conclusioni

5.1 Criticità riscontrate

- **Gestione della memoria/complessità computazionale**
  L'utilizzo della memoria e della complessità computazionale per l'aggiornamento e ricerca delle informazioni è uno degli aspetti più critici. Allo stato attuale si è deciso di sfruttare un array per la memorizzazione sulla factory dei contratti generati salvando di fatto l'indirizzo di riferimento. Tale sistema permette di minimizzare la quantità di informazioni salvate dal generatore, ma rende estremamente inefficiente, sopratutto nelle operazioni di ricerca.

- **Scalabilità della Blockchain**
  Il termine "scalabilità" identifica tre principali problemi legati alle prestazioni della Blockchain e sono:

  – **Throughput della Blockchain**
    Il "throughput della Blockchain" indica la capacità di transazioni elaborate dalla rete in un preciso lasso di tempo, calcolato in transazioni al secondo "tps". Prendendo in considerazione il Bitcoin, il suo throughput è allo stato attuale di 7 tps, mentre la rete Ethereum ha un tps che oscilla tra i 10 e i 30.

    Quando le transazioni superano il limite massimo gestibile dalla Blockchain, vengono messe in coda ed elaborate successivamente provocando un congestionamento della rete. Ipotizzando un utilizzo su ampia scala del progetto elaborato questo rappresenta un problema più che plausibile che necessita di una soluzione.

  – **Velocità di consolidamento delle transazioni**
    Le Blockchain per permettere l'inserimento all'interno della catena necessitano di un sistema di consenso come ad esempio il "Proof of Work" o il "Proof of Stake". La velocità di approvazione di un blocco non è costante ed in caso di numerose transazione si può registrare un peggioramento anche significativo delle prestazione causando di fatto un aumento dei tempi.

  – **Commissioni per transazione**
    La gestione dei costi di transazione è un aspetto centrale nel progetto elaborato. La commissione come spiegato nella sezione 4 dipende dalla complessità computazionale delle operazioni eseguite. La presenza di strutture dati e tipi di dato inefficienti, per quanto necessari, causano un aumento consistente della spesa.

    Un aumento eccessivo dei costi renderebbe poco conveniente l'utilizzo dello Smart-Contract, portando la maggior parte degli utenti ad optare anche per sistemi meno sicuri, ma più economici.

5.2 Potenziali soluzioni

Sulla base dei problemi riscontrati una delle principali soluzioni adottate riguardano l'utilizzo della rete IPFS (InterPlanetary File System) per il salvataggio dei dati e l'utilizzo di una serie di strutture e servizi off-chain per l'elaborazione delle operazioni preliminari alle transazioni significative e alle operazioni di ricerca/recupero di informazioni relative ai contratti.

Nello specifico con il termine IPFS si indica un protocollo di archiviazione distribuita e peer-to-peer che consente di archiviare e condividere file su una rete decentralizzata. Attraverso tale meccanismo, si potrebbe ridurre il quantitativo di informazioni salvate nel contratto e potenzialmente ridurre il costo delle operazioni eseguite. A tale sistema deve essere però affiancato un sistema off-chain, ergo una serie di strutture non legate alla Blockchain, ma esterne come ad esempio una base di dati centralizzata per rendere la rete scalabile ed efficiente il prodotto finale.

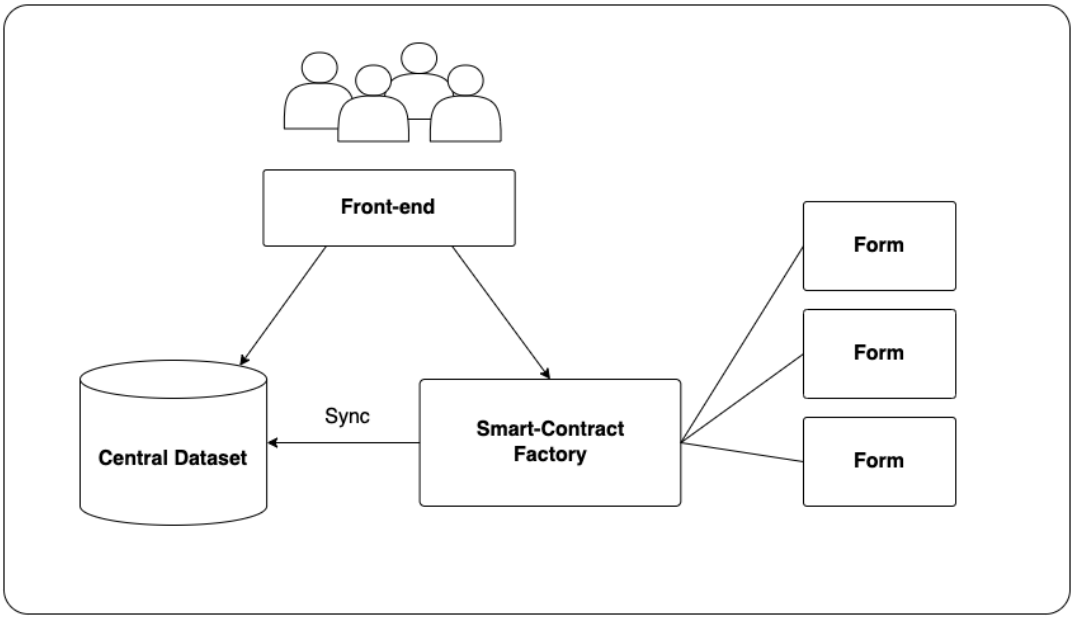


Figura 10: Architettura del risultato finale ipotizzato.

5.3 Lavori futuri

- Implementazione **standard ERC-721**
  Lo standard ERC-721 è un protocollo per la creazione di token non fungibili (NFT) sulla Blockchain di Ethereum. Un token non fungibile è un tipo di token che rappresenta un oggetto unico e irripetibile, come un'opera d'arte, un pezzo di terreno, un'auto d'epoca, ecc.

  Il protocollo ERC-721 definisce un'interfaccia standard per i contratti intelligenti che creano e gestiscono questi token, consentendo a tutti i token ERC-721 di essere compatibili tra loro. Ciò significa che i token NFT creati con questo standard possono essere gestiti e scambiati su qualsiasi piattaforma che supporta il protocollo ERC-721.

Rendendo il contratto in linea con le direttive ERC-721 si renderebbe di fatto i relativi form compatibili ed accessibili ad altri sistemi giuridici che per scelte potrebbero aver deciso di creare applicativi differenti.

- Implementazione **IPFS**
  L'utilizzo del protocollo IPFS (InterPlanetary File System), come illustrato nella sottosezione 5.2, permetterebbe di alleggerire il carico dello Smart-Contract e permetterebbe di avere uno strumento per il salvataggio senza la necessità di un sistema centralizzato.

- Implementazione **off-chain**
  L'implementazione di strutture off-chain sono necessarie per colmare le forti limitazioni relative l'efficienza della Blockchain nella gestione dei dati. Un esempio di potenziale struttura off-chain è rappresentata dall'utilizzo di un dataset esterno per lo svolgimento di operazioni critiche, come la gestione di eventi che non necessitano la registrazione su Blockchain o la ricerca delle informazioni. In entrambi i casi, e nelle idee che si intendono implementare, l'utilizzo di un sistema centralizzato sarebbe limitato ad operazioni poco rilevanti.