



UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

Corso di Laurea magistrale **in Ingegneria Informatica e dell'Automazione**

Realizzazione di una applicazione decentralizzata basata su blockchain per la gestione della cessione dei crediti di imposta

Development of a decentralized application for credit assignment based on blockchain technology

Relatore: Chiar.mo/a

Prof. Luca Spalazzi

Tesi di Laurea di:

Luzi Mattia

A.A. 2020 / 2021

Indice

| | |
|---|-----------|
| Indice | 1 |
| Introduzione | 3 |
| Superbonus 110% e Cessione del Credito | 4 |
| Superbonus 110% | 4 |
| Super Ecobonus e Super Sismabonus | 4 |
| Tipologie di immobili | 4 |
| Tipologie di interventi | 5 |
| Cessione del credito | 5 |
| Blockchain | 7 |
| Definizione | 7 |
| Caratteristiche | 7 |
| Classificazione | 9 |
| Algoritmi di consenso | 10 |
| Applicazioni e casi d'uso | 11 |
| Problematiche | 12 |
| Smart Contract | 13 |
| Perchè la Blockchain? | 14 |
| Hyperledger Sawtooth | 17 |
| Introduzione | 17 |
| Caratteristiche distintive | 17 |
| Architettura della rete | 19 |
| Transaction Family e Transaction processor | 20 |
| Global State | 21 |
| Journal e blockchain | 23 |
| Transaction Scheduling | 26 |
| Scheduling Sequenziale vs Parallelo | 26 |
| Scheduling all'interno del validatore | 27 |
| Transactions and Batches | 28 |
| Transaction | 28 |
| Batch | 29 |
| L'applicazione | 31 |
| Raccolta dei requisiti | 31 |
| Definizione delle fonti | 31 |
| Classificazione dei requisiti | 31 |
| Analisi dei requisiti | 34 |
| Story Card (Storie Utente) | 34 |
| Diagramma di Gantt delle storie utente | 44 |
| Task Card | 44 |
| Mockup (Modelli di Navigazione) | 45 |
| Diagramma di Stato | 50 |
| Progettazione | 53 |

| | |
|---|-----------|
| Diagramma delle classi | 53 |
| Progettazione del Database | 53 |
| Realizzazione | 56 |
| Applicazione Client e Database | 56 |
| Blockchain | 58 |
| Transaction Families (Smart Contract) | 60 |
| proposta_cessione | 61 |
| Indirizzi | 61 |
| Strutture dati | 61 |
| richiesta_accreditamento | 64 |
| Indirizzi | 64 |
| Strutture dati | 64 |
| utente | 65 |
| Indirizzi | 65 |
| Strutture dati | 65 |
| Sicurezza | 67 |
| Sicurezza della rete Blockchain | 67 |
| Sicurezza della firma digitale | 69 |
| Sicurezza delle chiavi | 70 |
| Sicurezza dell'identità su Blockchain | 74 |
| Sviluppi Futuri e Conclusioni | 76 |
| Sviluppi Futuri | 76 |
| Rilascio in produzione della rete blockchain | 76 |
| Miglioramento sicurezza recupero chiave | 77 |
| Maggiore automatizzazione dei controlli | 77 |
| Gestione degli altri servizi attraverso la blockchain | 77 |
| Gestione di più tipologie di bonus | 77 |
| Supporto alla "ricessione" del credito da parte dell'acquirente | 78 |
| Conclusioni | 78 |
| Bibliografia - Sitografia | 79 |

Introduzione

Questa tesi si propone di descrivere la progettazione e realizzazione di una applicazione per la gestione della cessione dei crediti di imposta, decentralizzata e basata su tecnologia blockchain. L'idea di questo progetto nasce dalla collaborazione con l'azienda MC Energy per lo sviluppo di un nuovo servizio per la gestione del credito d'imposta da aggiungere a quelli forniti dalla piattaforma Superbonus Manager. Lo sviluppo dell'applicazione ha interessato due tematiche fondamentali: la **cessione del credito**, in particolar modo riferito al Superbonus 110%, e la **tecnologia blockchain**. Negli ultimi anni lo stato italiano ha iniziato a concedere incentivi nella forma di bonus per la ristrutturazione edilizia, ammodernamento, efficientamento energetico, ecc. che danno diritto a delle detrazioni fiscali a chi ne beneficia. Queste detrazioni fiscali possono essere cedute come credito d'imposta a terzi (in genere grandi contribuenti che possono massimizzare l'utilità delle detrazioni stesse) in cambio di un rimborso di tutte o di una parte delle spese sostenute dal beneficiario. In questo contesto l'obiettivo principale dell'applicazione è quello di mettere in contatto il cedente e l'acquirente del credito, permettendo loro di gestire il processo di cessione in sicurezza e trasparenza, garantendo inoltre il soddisfacimento di tutti i requisiti previsti per il bonus. Uno dei mezzi per conseguire questo obiettivo è appunto l'utilizzo della blockchain. Negli ultimi tempi questa tecnologia è spesso al centro dell'attenzione sia in campo informatico che finanziario. Tutti infatti hanno sentito almeno una volta parlare di valute digitali o criptovalute, mining e Bitcoin; tuttavia le criptovalute sono solo una delle possibili applicazioni della tecnologia blockchain, benché la più conosciuta. Per questo motivo, l'altro importante obiettivo di questo progetto è dimostrare come, grazie alle sue proprietà di sicurezza, robustezza, decentralizzazione, ridondanza, questa tecnologia possa essere sfruttata in altri ambiti che non siano quello meramente finanziario delle criptovalute.

Oltre alla descrizione del processo di sviluppo questa tesi si propone di approfondire alcuni aspetti relativi all'applicazione che riguardano in particolar modo la tecnologia e il particolare framework blockchain utilizzato, la sicurezza, il dominio applicativo e i possibili sviluppi futuri. Per questo motivo è strutturata in sei capitoli che si concentrano su questi argomenti. Il primo capitolo contiene una breve descrizione del Superbonus 110% e del meccanismo di cessione del credito d'imposta per dare un'idea al lettore del problema che l'applicazione intende risolvere ma senza entrare nei dettagli tecnici e burocratici. Il secondo capitolo invece si apre con una panoramica sulle caratteristiche della tecnologia della blockchain e dei suoi possibili utilizzi, in modo da dare abbastanza informazioni al lettore per capire le scelte fatte in merito al suo utilizzo e che vengono esposte al termine del capitolo. Il terzo capitolo contiene una descrizione dettagliata del framework blockchain scelto per lo sviluppo dell'applicazione: Hyperledger Sawtooth. In questo capitolo vengono introdotte le caratteristiche del framework in merito alla struttura della rete, dei nodi, dei componenti dei singoli nodi e degli smart contract; evidenziando in particolar modo le peculiarità che lo distinguono dagli altri framework. Il quarto capitolo si concentra sulla descrizione del processo di sviluppo dell'applicazione vero e proprio, a partire dalla raccolta dei requisiti fino ad arrivare alla realizzazione, passando per le fasi di analisi dei requisiti e progettazione. Il quinto capitolo focalizza la sua attenzione sugli aspetti di sicurezza dell'applicazione. Vengono descritte le decisioni di sicurezza prese e le varie alternative considerate in merito a chiavi private, nodi della rete e firma digitale. Il sesto e ultimo capitolo contiene alcune idee in merito a possibili sviluppi futuri dell'applicazione e delle considerazioni finali a conclusione della tesi.

Superbonus 110% e Cessione del Credito

Questo capitolo descrive in breve il Superbonus 110% e il meccanismo di cessione del credito che costituiscono il dominio applicativo di quanto sviluppato.

Superbonus 110%

Il **Superbonus 110%** è una misura di incentivazione introdotta dal D.L. "Rilancio" 19 maggio 2020, n. 34¹, che ha l'obiettivo di incentivare opere di ristrutturazione delle abitazioni per renderle più sicure e più efficienti dal punto di vista energetico. L'incentivo consiste in una **detrazione fiscale del 110%** che si applica sulle spese sostenute dal 1° luglio 2020 al 30 giugno 2022 da ripartire tra gli aventi diritto in cinque quote annuali e, per la parte di spesa sostenuta nell'anno 2022, in quattro quote annuali di pari importo. La detrazione fiscale va intesa come possesso di "credito d'imposta" nei confronti dello stato, questo significa che i beneficiari del bonus potranno utilizzarlo per compensare, o se possibile estinguere, eventuali debiti dovuti allo Stato, come quelli relativi ad imposte o contributi.

Super Ecobonus e Super Sismabonus

Il Superbonus 110% si suddivide in due sottotipologie:

- **Super Ecobonus:** Bonus che riguarda i lavori di efficientamento energetico. Per quanto riguarda il Super Ecobonus, le persone fisiche possono svolgere i lavori su un massimo di due unità abitative, salvo gli interventi sulle parti comuni che sono sempre agevolabili, a prescindere dal numero di unità possedute.
- **Super Sismabonus:** è invece pensato per interventi di adeguamento antisismico. Sul Super Sismabonus, invece non ci sono limiti sul numero delle abitazioni ristrutturabili. Per poter godere della detrazione al 110% l'edificio deve essere situato nelle zone sismiche 1,2,3.

Nell'applicazione sviluppata questa distinzione non è rilevante in quanto già gestita dal servizio di gestione delle pratiche del SuperBonus 110 preesistente nella piattaforma a cui l'applicazione è stata integrata.

Tipologie di immobili

Il Superbonus garantisce una detrazione fiscale sulle spese sostenute per interventi effettuati da condomini, persone fisiche al di fuori dell'attività di impresa, Istituti autonomi case popolari (IACP), cooperative di abitazione a proprietà indivisa, organizzazioni non lucrative di utilità sociale, organizzazioni di volontariato, associazioni di promozione sociali, associazioni e società sportive dilettantistiche, persone fisiche che risiedono in edifici composti da due a quattro unità immobiliari distintamente accatastate possedute da un unico proprietario o in comproprietà da più persone fisiche. Possono quindi beneficiare del Superbonus le persone fisiche che vivono in condomini, in edifici composti da due a quattro unità immobiliari distintamente accatastate possedute da un unico proprietario o in comproprietà, in edifici unifamiliari o in unità immobiliari site all'interno di

¹ <https://www.gazzettaufficiale.it/eli/id/2020/07/18/20A03914/sg>

edifici plurifamiliari che siano funzionalmente indipendenti² e dispongano di uno o più accessi autonomi dall'esterno³.

Tipologie di interventi

Il Superbonus 110% distingue due tipologie di interventi:

- **Trainanti:** Gli interventi trainanti sono sostanzialmente due e consistono: nell'*isolamento termico dell'involucro dell'edificio* (che deve essere plurifamiliare o unifamiliare) e *nella sostituzione degli impianti termici con impianti centralizzati*, nella (su edifici unifamiliari o sulle unità immobiliari site all'interno di edifici plurifamiliari che siano funzionalmente indipendenti e dispongano di uno o più accessi autonomi dall'esterno). Per poter godere del Super Ecobonus è necessario effettuare almeno un intervento trainante.
- **Trainati:** Gli interventi trainati consistono nella sostituzione degli infissi, le schermature solari, l'installazione di impianti fotovoltaici, dei sistemi di accumulo, delle colonnine per la ricarica dei veicoli elettrici, degli impianti di domotica, l'eliminazione delle barriere architettoniche per le persone portatrici di handicap in situazione di gravità e per le persone con età superiore ai 65 anni, e altri. Una volta eseguito almeno uno degli interventi trainanti, il beneficiario può decidere di effettuare anche gli interventi trainati.

L'insieme di questi interventi (trainanti e trainati) deve comportare un miglioramento minimo di almeno due classi energetiche dell'edificio o dell'unità immobiliare sita all'interno di edifici plurifamiliari che sia funzionalmente indipendente e disponga di uno o più accessi autonomi dall'esterno. Ogni tipologia di intervento ha inoltre un limite di spesa, ovvero un "massimale" che non può essere superato e che dipende da diversi fattori come la tipologia di edificio, le sue dimensioni e la regione

Anche in questo caso i controlli relativi agli interventi sono oggetto di interesse del servizio di gestione delle pratiche.

Cessione del credito

Riguardo alle modalità a disposizione del cosiddetto "beneficiario", ovvero colui che beneficia dei miglioramenti derivanti dagli interventi, esso ha a disposizione due opzioni:

- **Cessione del credito:** Il cittadino beneficiario può sostenere direttamente il costo dei lavori, maturando il credito d'imposta che ne deriva e decidere poi se utilizzarlo per la detrazione in compensazione per pagare meno tasse o se cederlo a terzi (istituti di credito compresi).
- **Sconto in fattura:** l'impresa o le imprese che hanno effettuato i lavori applicano uno sconto fino al 100% del valore della fattura. Questo permette al cittadino beneficiario dei lavori di effettuare i lavori senza alcun esborso monetario. All'impresa invece viene riconosciuto il diritto di beneficiare della detrazione fiscale, ottenendo credito d'imposta pari al 110% dell'ammontare dello sconto applicato. In sostanza quindi il cittadino beneficiario paga

² Un'unità immobiliare può ritenersi "funzionalmente indipendente" qualora sia dotata di almeno tre delle seguenti installazioni o manufatti di proprietà esclusiva: impianti per l'approvvigionamento idrico, impianti per il gas, impianti per l'energia elettrica, impianto di climatizzazione invernale.

³ Per "accesso autonomo" si intende invece un ingresso indipendente non comune ad altre unità immobiliari, chiuso da cancello o portone di ingresso che consenta l'accesso dalla strada o da cortile o giardino, anche di proprietà non esclusiva.

quanto dovuto all'impresa con il proprio credito d'imposta. Questo credito può essere poi utilizzato, come descritto precedentemente, per le detrazioni fiscali in cinque (o quattro per le spese sostenute nel 2022) quote annuali di pari importo. Ad esempio, se il valore complessivo dei lavori è pari a 10.000€ e l'impresa decidesse di applicare uno sconto pari al 100% della fattura, si vedrà riconosciuto un credito di 11.000€.

Si noti come lo sconto in fattura sia una sorta di cessione del credito "facilitata" per il beneficiario dei lavori che in questo modo non deve interessarsi di tutto il processo burocratico di trasferimento del credito d'imposta. Inoltre l'impresa che ottiene il credito d'imposta attraverso la modalità di sconto in fattura può cederlo a sua volta in maniera analoga al beneficiario.

Questa misura crea un meccanismo virtuoso di mercato che offre benefici a tutti i soggetti coinvolti: il cittadino può ristrutturare casa gratuitamente, ridurre il costo delle bollette e valorizzare il proprio patrimonio immobiliare; l'impresa può aumentare il proprio fatturato grazie al maggior volume di lavori; lo Stato può rendere più efficienti e più sicure le abitazioni e sostenere l'aumento dell'occupazione e del reddito. [1][2]

La progettazione dell'applicazione (su cui verte questa tesi) si concentra sull'opzione di cessione del credito essendo quella più vantaggiosa e utilizzata (in particolar modo in progetti più grandi e con costi elevati da sostenere). L'obiettivo è quello di facilitare l'incontro tra due "attori" i cui bisogni sono "complementari" cioè coloro che vogliono cedere il credito e coloro che intendono acquistarlo e di gestire il processo di cessione stesso:

- **Cedente:** colui che ha sostenuto delle spese in un progetto di ristrutturazione che rispetta tutti requisiti previsti dal Superbonus 110% e che quindi ha "maturato" del credito d'imposta. Il cedente vuole rientrare delle spese sostenute in tempi brevi oppure dal momento che non vuole, non può, o non gli conviene usufruire della detrazione fiscale derivante dal bonus.
- **Acquirente:** colui che è disposto ad acquistare il credito maturato dal cedente e ad ottenere quindi il diritto alle detrazioni fiscali relative a quel progetto di ristrutturazione.

Blockchain

Questo capitolo contiene una introduzione ai concetti fondamentali riguardanti la tecnologia della blockchain, utile per comprendere il perchè è stata adottata in questo progetto e le scelte fatte a riguardo che sono descritte nell'ultima parte del capitolo.

Definizione

La **blockchain** (letteralmente "catena di blocchi") è un struttura dati pensata per essere condivisa, distribuita e "immutabile". Per questo motivo, è una struttura dati utilizzata da numerosi **registri digitali distribuiti** (Distributed Ledger Techonology - DLT). Nello specifico, le voci del registro digitale sono raggruppate in **blocchi concatenati e ordinati in ordine cronologico**, la cui integrità è assicurata mediante l'utilizzo della crittografia. Le voci del registro di cui si costituiscono i blocchi vengono in genere denominate **transazioni**.

Caratteristiche

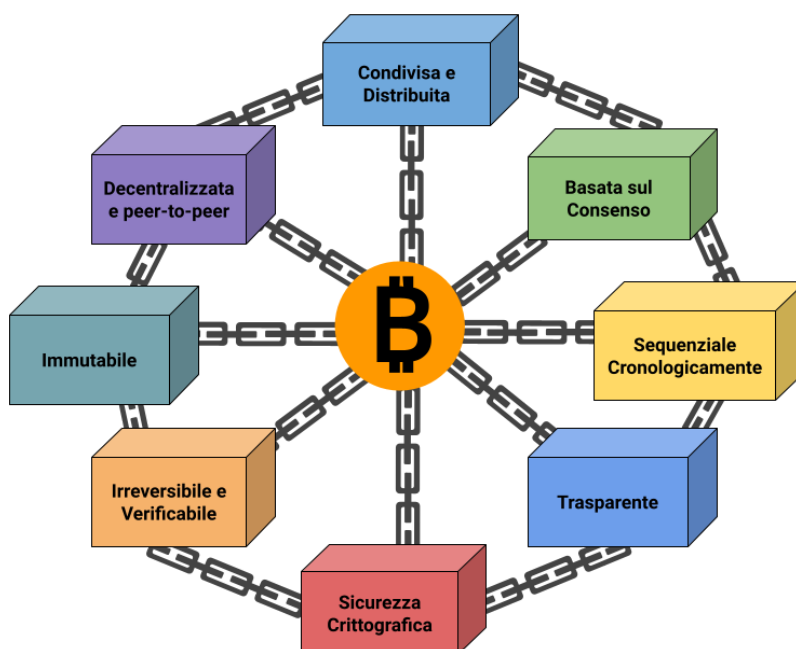


Figura 2.1 - Caratteristiche distintive della blockchain

L'immutabilità della blockchain consiste nel fatto che l'unica modifica concessa su di esse è l'aggiunta di voci. Ciò fa sì che le voci già presenti non possono essere modificate o eliminate a meno di invalidare l'intera struttura e che la dimensione della blockchain può solo aumentare. Questa tecnologia rientra nella più ampia famiglia dei Distributed Ledger, ovvero di sistemi che si basano su un registro distribuito, che può essere letto e modificato dai nodi di una rete. I nodi della rete conservano una copia del registro che aggiornano utilizzando un protocollo condiviso in modo da garantirne la coerenza. Nelle blockchain pubbliche, in particolare, il protocollo per l'aggiornamento delle copie del registro consente di garantire la coerenza senza che i nodi si conoscano e si fidino l'uno dell'altro. Ulteriori caratteristiche di questa tecnologia sono la digitalizzazione dei dati, l'anonimato, la decentralizzazione, la tracciabilità, la trasparenza e la verificabilità.

La natura **distribuita** della rete e il **protocollo di aggiornamento** che vede la cooperazione dei nodi rendono da una parte più robusto il processo di validazione dei blocchi aggiunti, mentre

dall'altra fanno aumentare i tempi in cui questo avviene a causa della necessità di sincronizzare i nodi della rete su una stessa versione della blockchain. I nodi che modificano la blockchain, detti **miner**, validano le nuove transazioni e le aggiungono al blocco che stanno costruendo dopo aver verificato l'intera blockchain. Una volta completato il blocco, lo trasmettono agli altri nodi della rete. A causa della natura concorrente del processo di validazione può capitare che alcuni nodi della rete producano simultaneamente più blocchi concorrenti (ovvero blocchi collegati ad un blocco già esistente ma che differiscono tra di loro per le transazioni che li costituiscono): questo dà origine ad una **biforcazione (o fork)** nella catena. In questo caso è il protocollo di aggiornamento che permette di stabilire quali blocchi delle ramificazioni i nodi andranno ad includere nella blockchain e quali verranno esclusi andando a costituire dei **"blocchi orfani"** in modo da mantenere le copie della blockchain consistenti.

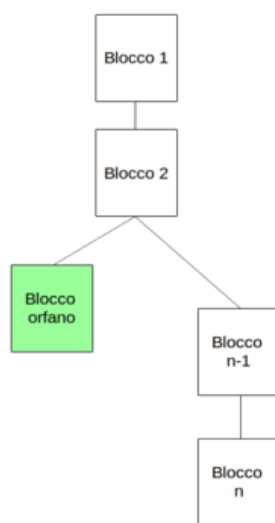


Figura 2.2 - Biforcazione della catena principale e generazione di un blocco orfano ([3])

Altro aspetto importante è quindi il decentramento dei dati memorizzati: la massiva replicazione dei blocchi in un network di nodi evita di avere un single point of failure e aumenta la qualità e la robustezza delle transazioni memorizzate. Non essendoci una centralizzazione è molto più difficile se non impossibile per un eventuale attaccante corrompere o invalidare un blocco o anche una singola transazione, considerando anche che la difficoltà aumenta all'aumentare del numero di nodi e che nessuna copia della blockchain è "più ufficiale" o "credibile" delle altre, essendo tutti gli utenti allo stesso livello di credenziali. Altra caratteristica importante della blockchain è l'**anonimato**. Nella blockchain ogni utente che vuole eseguire e quindi registrare una transazione è identificato mediante un indirizzo. L'indirizzo è in genere un insieme di caratteri esadecimale derivati da una chiave pubblica (o la chiave pubblica stessa). La chiave pubblica è essa stessa derivata da una chiave privata che l'utente tiene segreta e che utilizza per "firmare digitalmente" le transazioni. La chiave pubblica, oltre a "identificare" l'utente viene utilizzata per la verifica della firma digitale (garantire che la transazione è stata firmata dal possessore della chiave privata che ha generato quella specifica chiave pubblica). La coppia **chiave pubblica - chiave privata** costituisce il cosiddetto **digital wallet** (portafoglio digitale). La chiave privata è generata randomicamente e non è collegata alle informazioni personali dell'utente. Inoltre un utente può avere più portafogli digitali nella stessa blockchain (eventualmente anche uno diverso per transazione). Utilizzando quindi la crittografia asimmetrica la blockchain garantisce da un lato

privacy e anonimato agli utenti e dall'altro un modo efficace per verificare l'autenticità delle transazioni registrate [3].

Classificazione

Considerando alcuni criteri, i sistemi basati su blockchain possono essere classificati in pubblici, privati e consorzi.

- **Blockchain privata:** una blockchain privata è pensata per facilitare lo scambio di informazioni tra gruppi di individui facenti parte di una stessa organizzazione o di un insieme di organizzazioni. L'organizzazione o il gruppo di organizzazioni controlla in maniera selettiva la partecipazione di un utente alla blockchain e le operazioni che ciascun partecipante può eseguire, in particolar modo il diritto di scrivere sulla blockchain e di minare i blocchi. Gli utenti in genere sono invitati a partecipare o devono esplicitamente richiederlo all'organizzazione che controlla e possiede la blockchain. Questa caratteristica porta ad una maggiore centralizzazione e chiusura della blockchain rispetto ad una pubblica. Inoltre il fatto che gli utenti debbano essere identificati in qualche modo per accedere alla rete fa perdere la caratteristica di anonimato nella blockchain privata. Una blockchain privata può essere "**permissioned**" (**con permessi**) o "**permissionless**" (**senza permessi**): in una blockchain permissioned agli utenti sono associati dei permessi in merito alle transazioni che possono eseguire (generalmente basato su un sistema di ruoli che gli utenti ricoprono all'interno della rete), mentre in una blockchain permissionless non ci sono limitazioni in questo senso.
- **Blockchain Pubblica:** una blockchain pubblica costituisce una piattaforma alla quale "chiunque" può unirsi, e nella maggior parte dei casi può eseguire e minare transazioni, senza restrizione di alcun tipo su questo genere di operazioni, (cosa che la rende permissionless). Ogni partecipante ha la piena capacità di leggere e scrivere transazioni dalla e sulla blockchain, revisionare e verificare le transazioni già registrate in ogni momento. In questo senso quindi la blockchain è aperta e trasparente, non ci sono nodi a cui è assegnato il compito specifico di "validatori". Ogni utente può replicare il contenuto dell'intera blockchain sincronizzandosi con i nodi su cui è distribuita diventando esso stesso un nodo, garantendo indirettamente l'immutabilità della blockchain. Infatti sebbene la pubblica disponibilità della blockchain può inizialmente sembrare una grande vulnerabilità, l'utilizzo di un protocollo o meccanismo/ algoritmo di consenso unito a quelli di validazione e verifica crittografica per l'aggiornamento della blockchain, consente di avere una blockchain tanto più robusta e sicura all'aumentare del numero di nodi. Il grande vantaggio di una blockchain pubblica è che non è necessaria alcuna protezione specifica verso utenti malintenzionati (essendo questa garantita dalle caratteristiche intrinseche della blockchain) e di nessun controllo degli accessi. Queste caratteristiche fanno sì che né il possesso né tantomeno il controllo della blockchain si possa attribuire a una singola persona o organizzazione.
- **Blockchain Consorzio:** una blockchain consorzio può essere considerata come una blockchain parzialmente privata e permissioned, nella quale non c'è il controllo da parte di una organizzazione ma bensì la responsabilità della validazione dei blocchi e dell'applicazione dell'algoritmo di consenso è data ad un insieme predefinito di nodi. Questi nodi stabiliscono quali utenti possono partecipare alla rete e minare i blocchi. In questo tipo di blockchain la validazione di un blocco fa uso di schemi a firma multipla, in cui un blocco viene considerato valido se e solo se è stato firmato da uno di questi nodi. Ciò rende questo sistema parzialmente decentralizzato in quanto il controllo della blockchain è in

mano a un sottoinsieme di nodi (o consorzio). Il consorzio nodi può stabilire se i permessi in scrittura sulla blockchain siano pubblici o se limitati ai nodi che fanno parte della rete [5].

Algoritmi di consenso

Come già accennato affinché l'aggiornamento della blockchain avvenga in maniera corretta, sicura e coerente, i nodi della rete devono in qualche modo accordarsi su quale sia la nuova versione della blockchain da replicare in ogni nodo ogni qual volta un blocco di transazioni viene aggiunto, devono insomma raggiungere un consenso. Per fare ciò i nodi seguono un determinato protocollo che prende il nome di **algoritmo di consenso**. Esistono diversi tipi di algoritmi di consenso [4][6]:

- **Proof-of-work:** il primo algoritmo per il consenso decentralizzato proposto da Satoshi Nakamoto attualmente ancora utilizzato nella rete Bitcoin. Questo algoritmo prevede che i nodi miner siano in competizione con lo scopo di calcolare l'hash del prossimo blocco di transazioni (che hanno già validato) da aggiungere alla blockchain in modo tale che questo sia inferiore a un valore target stabilito dalla rete e che varia a secondo l'algoritmo. Il primo nodo che risolve questa sorta di sfida crittografica trasmette il blocco alla rete, attende la conferma degli altri nodi e aggiunge il blocco alla blockchain. Lavorando in parallelo è possibile che più di un nodo miner trovi una "soluzione" valida alla **sfida crittografica** e pertanto che sia generato più di un blocco valido da aggiungere in coda allo stesso blocco della blockchain. Questo come già detto crea una biforcazione temporanea della blockchain (fork e creazione di un branch). In questo casi i nodi miner lavorano per la validazione su entrambe le biforcazione (accettando temporaneamente entrambe le versioni), ma appena in una delle due viene validato ed aggiunto un nuovo blocco, tutti i miner si spostano su quel ramo rendendo il o i blocchi dell'altro ramo orfani. Il proof-of-work infatti prevede che i nodi accettino in ogni momento la più lunga versione della blockchain disponibile per risolvere la biforcazione [7].
- **Proof-of-stake:** un algoritmo proposto per superare lo svantaggio dell'eccessivo consumo di energia dal proof-of-stake applicato al Bitcoin. Invece di investire in risorse (potenza di calcolo) per poter eseguire il calcolo degli hash del proof-of-work, nel proof-of-stake si propone di utilizzare il possesso di criptovaluta da parte del miner che viene utilizzata come puntata (stake). Il valore della puntata è direttamente proporzionale alla probabilità di essere selezionato come nodo validatore del prossimo blocco che viene selezionato randomicamente e non è quindi predeterminato. Il nodo che producono blocchi validi vengono ricompensati, tuttavia se il blocco non viene aggiunto alla blockchain perdono l'ammontare puntato. Esistono diverse varianti del proof-of-stake che sono stati sviluppati ad hoc tenendo conto di alcuni fattori quali:
 - **Tipologia:** permissioned o permissionless
 - **Frequenza delle transazioni:** la frequenza alla quale le transazioni vengono confermate, che viene di fatto stabilita dall'algoritmo di consenso. Nel bitcoin, che invece utilizza il proof-of-work, la frequenza è di circa 7 transazioni/sec a causa del tempo necessario per il calcolo dell'hash che limita l'aggiunta a di un blocco a circa uno ogni 10 minuti.
 - **Scalabilità:** si intende la capacità del sistema di continuare a raggiungere il consenso all'aumentare del numero di nodi
 - **Costi di partecipazione:** in alcune blockchain viene imposto un costo di partecipazione iniziale. Nel proof-of-stake questo si traduce nell'investimento in criptovaluta, necessaria per poter partecipare alla validazione dei blocchi. Nel proof-of-work non è previsto un costo di partecipazione a meno che non si voglia

anche minare le transazioni, in quel caso l'energia utilizzata per minare le transazioni può essere considerata come un costo indiretto.

- **Fiducia dei nodi:** se i nodi della blockchain si possono considerare conosciuti e affidabili, come nelle blockchain private e consorzi o sconosciuti e non affidabili, come nelle blockchain pubbliche [8].
- **Practical Byzantine Fault Tolerance Algorithm (PBFT):** il PBFT è stato proposto come soluzione al **problema dei generali bizantini**. Questo è un problema informatico su come raggiungere un consenso in una situazione in cui le informazioni scambiate e gli interlocutori non sono totalmente affidabili che può essere esemplificato in questo modo: l'esercito bizantino cinge d'assedio una città nemica. L'esercito è comandato dai generali che possono comunicare unicamente attraverso messaggeri, alcuni dei generali bizantini sono leali, altri sono traditori. Affinché l'esercito bizantino vinca tutti i generali devono collaborare per stabilire un piano comune e attaccare simultaneamente. In aggiunta, indipendentemente da ciò che fanno i traditori, i generali leali devono proseguire con il piano stabilito mentre i traditori devono rovinarlo. Analogamente nella blockchain il PBFT permette di stabilire il consenso tra i nodi partecipanti attraverso lo scambio di messaggi che permette ad ogni nodo di raggiungere una decisione. La decisione con più nodi rappresenta il consenso raggiunto. Il PBFT è in grado di garantire che la "corretta" decisione venga presa fin quando il numero di nodi "traditori" non supera $\frac{1}{3}$ **del numero totale di nodi** [9][10].

Applicazioni e casi d'uso

Sebbene non si debba intendere come una sorta di panacea ad ogni problema, come spesso ultimamente viene considerata, grazie alle sue caratteristiche di sicurezza, decentralizzazione, trasparenza, immutabilità, ecc. nel tempo questa tecnologia è diventata la base per molte applicazioni in diversi settori quali economia, finanza, sanità, industria e altri, in cui tutte o in parte queste caratteristiche costituiscono un requisito o un vantaggio rispetto ad altre tecnologie. Di seguito sono descritti alcuni casi d'uso:

- **Criptovalute:** questo è sicuramente il caso d'uso che ha reso popolare la blockchain a partite dal bitcoin a cui poi sono seguite svariate altre criptovalute quali ethereum, ripple, dash ecc... In questo caso la blockchain rappresenta il registro sul quale vengono memorizzate transazioni che rappresentano uno scambio di criptovaluta tra due indirizzi/utenti della blockchain senza bisogno di intermediari. In questo senso la blockchain si "sostituisce" al ruolo delle banche o degli istituti di credito con l'importante differenza (almeno in quelle pubbliche) di essere decentralizzata e garantendo l'anonimato degli utenti.
- **Gestione e tracciamento delle risorse:** in questo caso la blockchain viene utilizzata per gestire il trasferimento in maniera sicura e tracciabile di una risorsa all'interno di una rete commerciale o una filiera di produzione. Per risorsa si può intendere sia qualcosa di fisico sia qualcosa di intangibile come un software o un servizio. Serializzando la risorsa tutti i suoi spostamenti e/o utilizzi possono essere registrati dai partner della rete da quando viene creata fino al suo utilizzo finale[13].
- **Mercato Immobiliare:** le transazioni nel mercato immobiliare sono spesso lente, poco trasparenti e costose principalmente a causa della presenza di intermediari quali brokers, catasti nazionali, ispettori, notai ecc. In questo campo la blockchain può essere utile per

registrare ogni proprietà associandola ad un indirizzo digitale che contenga tutte le caratteristiche di quella proprietà, le sue informazioni legali e finanziarie e mantenere uno storico di tutte le transazioni in cui viene coinvolta [14].

- **Finanza:** i processi di pagamento, in particolar modo quelli transnazionali sono spesso lenti e costosi a causa della presenza di intermediari. Un pagamento potrebbe passare attraverso diverse banche e valute, cosa che ne aumenta le tempistiche e i costi. La blockchain permette di ridurre i tempi ed abbattere i costi eliminando gli intermediari non necessari rendendo così per i trasferimenti di denaro all'estero più convenienti.
- **Sanità:** la blockchain può essere utilizzata per mantenere la confidenzialità delle informazioni sanitarie dei pazienti, consentendo l'accesso e il diritto di aggiornarle solo a determinati utenti quali medici curanti o fornitori di assicurazioni sanitarie [11].
- **Notarizzazione:** la blockchain può essere utilizzata per certificare la data certa di un documento e il fatto che questo non abbia subito alcuna variazione. Questo può essere fatto calcolando l'hash del documento da certificare e registrarlo nella blockchain. Qualsiasi modifica futura del documento può essere individuata ricalcolando l'hash del documento e confrontandolo con quello registrato in passato.
- **Non-fungible token (NFT):** un token non fungibile (o non usabile) è un token ("gettone") che rappresenta qualcosa di unico. Questo li rende appunto non fungibili in quanto non reciprocamente intercambiabili, a differenza della criptovaluta o della valuta in generale che è per natura fungibile e quindi intercambiabile. I token non fungibili sono utilizzati per superare il problema della potenzialmente infinita riproducibilità di un bene digitale, creando scarsità digitale verificabile e introducendo il concetto di proprietà di un bene digitale. La blockchain viene quindi utilizzata per certificare la proprietà di un bene digitale che può essere un'opera d'arte (crypto art), un oggetto da collezione, una risorsa scambiabile tra utenti in un videogioco (senza gli sviluppatori come intermediari) [12].

Problematiche

Nonostante i notevoli punti di forza di questa tecnologia, esistono alcuni svantaggi di cui si deve tener conto quando si sceglie di utilizzarla. Uno dei problemi più importanti è la **scalabilità**: il processo di raggiungimento del consenso e validazione di un blocco implica che i nodi debbano poter avere a disposizione l'intera blockchain, ovvero tutte le transazioni che sono avvenute fino a quel momento, cosa che richiederebbe sempre maggior memoria. Per mitigare questo problema si introducono limiti alle dimensioni dei blocchi (cosa che inoltre velocizza la trasmissione dei blocchi tra i nodi) e si specializzano i nodi in ruoli differenti:

- **Full nodes:** contengono tutte le transazioni registrate nella blockchain a partire dal primo blocco (genesi), possono inoltre memorizzare informazioni sul saldo dei wallet e inviare nuove transazioni.
- **Light nodes:** contengono una versione parziale della blockchain, possono inoltre memorizzare informazioni sul saldo dei wallet e inviare nuove transazioni. Sono connessi con uno o più full nodes per poter verificare le transazioni già aggiunte in un determinato blocco.

- **Mining nodes:** non contengono transazioni della blockchain essendo il loro unico compito quello di validare blocchi di transazioni e aggiungerli alla catena. Per farlo necessitano di avere accesso alle transazioni precedenti e pertanto devono essere connessi con uno o più full nodes [15].

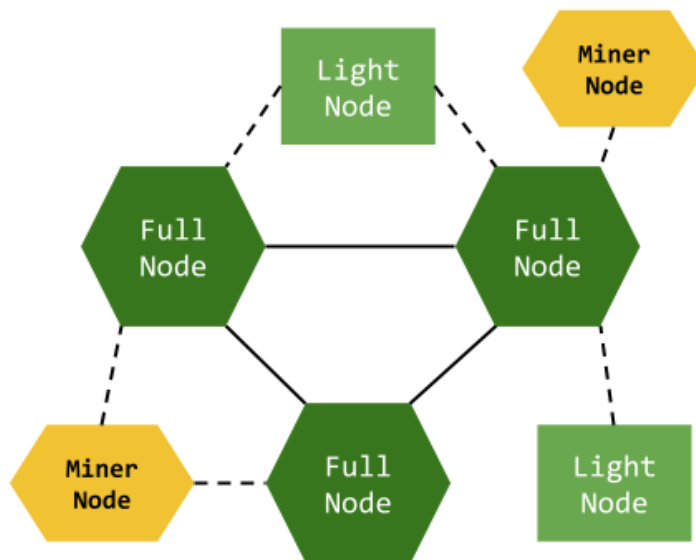


Figura 2.3 - Differenti tipi di nodi nella blockchain

Un altro problema è il cosiddetto "**51% attack**" per blockchain con proof-of-work: questo si può presentare quando più del 51% dei nodi (o meglio il 51% della potenza di calcolo della rete) si mette insieme per poter generare blocchi "falsi" o per annullare transazioni già convalidate. Una tale potenza di calcolo infatti permette di generare blocchi ad una velocità mediamente maggiore del resto della rete rendendo impossibile agli altri nodi aggiungere la versione legittima, (o meglio il branch legittimo). Infine, un altro grande problema della blockchain che utilizzano il proof-of-work è il consumo di energia dovuto al mining (in particolare alla soluzione dell'hash dei blocchi). Si stima che per il mining di 1 bitcoin sia necessario l'equivalente di due anni di consumi di una casa media (USA) e che una singola transazione bitcoin richieda 80.000 volte l'energia utilizzata per un per una transazione con carta di credito.

Smart Contract

Uno **smart contract** è un programma o un protocollo informatico che può essere utilizzato per verificare, negoziare o imporre il rispetto delle clausole contrattuali in maniera automatica. In questo senso nascono con l'obiettivo di eliminare la necessità di intermediari fidati nella stipulazione di un contratto diminuendo da una parte i costi (sia di gestione, sia di contrattazione sia quelli legati al rischio) e dall'altro la possibilità di errori, frodi, divulgazioni di informazioni riservate, ecc. Nonostante il nome uno smart contract non costituisce un accordo con valore legale ma è più da intendere come un mezzo attraverso il quale far rispettare gli obblighi derivanti dalle clausole di un contratto vero e proprio, precedentemente stipulato. La maggior parte delle applicazioni che coinvolgono gli smart contract si basano sull'esecuzione automatica di determinate azioni come conseguenza del soddisfacimento di determinate condizioni (es. il trasferimento di criptovaluta ad un determinato indirizzo una volta che tutte le condizioni stabilite sono soddisfatte). *Secondo i sostenitori degli smart contract molte tipi di clausole contrattuali possono essere rese parzialmente o integralmente automatizzate, auto-ottemperanti, o entrambe le cose, ciò renderebbe addirittura più sicuri gli smart contract rispetto ai contratti tradizionali oltre che, come già detto, meno costosi.*

Gli smart contract hanno assunto sempre maggior popolarità a partire dal 2015, quando è stata lanciata la blockchain Ethereum che per prima ha consentito il rilascio e l'esecuzione di programmi su blockchain. Da questo momento il termine smart contract è stato sempre più utilizzato in maniera più generica per indicare un programma, un insieme di codice (funzioni) e dati (stato), rilasciato attraverso transazioni firmate digitalmente ed eseguito sui nodi di una rete blockchain, svincolandosi dal concetto di contratto. In questo senso uno smart contract permette di memorizzare e manipolare in maniera sicura dei dati in quanto le operazioni che si possono compiere su di essi (ovvero l'esecuzione di un contratto) sono anch'esse memorizzate, verificate ed eseguite sulla blockchain e non su programmi connesse ad essa. Gli utenti interagiscono con uno smart contract eseguendo transazioni che provocano l'esecuzione del contratto che a sua volta provoca l'aggiornamento del proprio stato e/o l'esecuzione di un altro smart contract. A causa della natura immutabile della blockchain uno smart contract non può essere modificato successivamente al suo rilascio cosa che richiede particolare attenzione nella scrittura di un contratto e nella fase di testing. [16]

Perchè la Blockchain?

La tecnologia blockchain rappresenta senza dubbio la componente fondamentale ed innovativa di questo progetto. Negli ultimi tempi l'utilizzo al tecnologia della blockchain è diventato un tema caldo sia nel campo dell'informatica che in quello dell'economia. Tuttavia la maggior parte dell'interesse intorno a questa tecnologia, come si è già accennato, riguarda principalmente un singolo caso d'uso, ovvero quello delle criptovalute. Uno degli obiettivi di questo progetto è invece ribadire come questa tecnologia, unita a quella degli smart contract possa essere in campi di applicazione diversi da quello prettamente finanziario. La gestione delle pratiche di cessione del credito comporta la presenza di **numerosi documenti e certificazioni** a partire dai quali devono poter essere fatti dei controlli e delle verifiche che condizionano l'ottenimento del credito. I progetti legati alle pratiche di cessione hanno un valore economico che va dalle **decine alle centinaia di migliaia di euro**, pertanto un errore nelle verifiche o la perdita o la corruzione di dati relativi ad una pratica può comportare un danno economico ai contraenti di un accordo di cessione. Appare chiaro quindi come fosse necessario uno strumento in grado sia di garantire **l'integrità e l'incorruttibilità** delle informazioni e dei documenti legati ad una pratica sia di mantenere la tracciabilità di tutto il flusso della cessione, dal momento della creazione della pratica al momento della cessione vera e propria del credito. Questo strumento doveva inoltre essere in grado di fornire un certo livello di sicurezza nell'assegnazione delle pratiche ai determinati acquirenti in particolar modo per quel che riguarda tutto il meccanismo di "asta". Alla luce delle considerazioni fatte precedentemente si può capire come le caratteristiche della tecnologia blockchain siano perfettamente compatibili con questi requisiti. Bisogna però analizzare bene il ruolo della blockchain in questa applicazione: la blockchain non è da intendere semplicemente come un database dell'applicazione ma bensì come un supporto al database vero e proprio. La blockchain di questa applicazione è pensata per mantenere per ogni pratica e per ogni accreditamento di un utente un insieme di informazioni chiave e di gestirne il corretto aggiornamento in accordo alle regole stabilite. Le pratiche e gli accreditamenti costituiscono di fatto degli smart contract eseguiti sulla blockchain che va a coprire almeno due dei tanti possibili casi d'uso (in parte elencati precedentemente) ovvero gli **NFT** e la **notarizzazione di documenti**:

- L'insieme delle informazioni che costituiscono una pratica di cessione rappresenta di fatto un NFT. Infatti ogni pratica di cessione rappresenta una "risorsa" unica che viene generata da un utente cedente dimostrando di avere tutti i requisiti necessari. Questa risorsa viene acquistata o tramite un asta o in maniera diretta da un utente acquirente. Il possesso di questa risorsa da diritto, una volta terminata correttamente la pratica, al credito generato dal relativo alla progetto.




- A causa delle dimensioni dell'insieme dei documenti legati ad una pratica di cessione o di accreditamento di un utente, essi non possono essere memorizzati efficientemente sulla blockchain. Tuttavia essendo una parte importante delle pratiche si è giunti alla soluzione di conservare "l'essenza" del documento memorizzando l'hash. Sebbene ciò non permetta di recuperare un documento in caso di una eventuale perdita, consente di individuare eventuali contraffazioni e modifiche non autorizzate al documento originale tramite il confronto degli hash. I documenti di una pratica sono quindi notarizzati tramite la blockchain.

Il processo di cessione del credito necessita tra le altre cose della conoscenza delle informazioni specifiche riguardo all'identità dei vari utenti che, inoltre, ricoprono precisi ruoli nell'ambito dell'applicazione, ruoli a cui sono associati determinati permessi che determinano le operazioni che possono compiere e le informazioni a cui possono accedere. Queste due considerazioni hanno portato a rinunciare alla caratteristica di anonimato che contraddistingue la blockchain in favore di una assoluta trasparenza, orientandosi verso soluzioni **permissioned** e ad una **blockchain privata con controllo degli accessi**. Inoltre è stato deciso che, almeno per la fase iniziale del progetto, i nodi della blockchain non sono controllati dagli utenti stessi ma da una singola organizzazione (si considerano quindi fidati), con la possibilità di evolvere in un consorzio di organizzazioni. Per questo motivo e in considerazione degli obiettivi di cui si è parlato precedentemente, la scelta del meccanismo di consenso più appropriata è ricaduta su un algoritmo che fosse almeno **crash fault tolerant** (tollerante ai guasti sui nodi) e preferibilmente anche **byzant fault tolerant**, quale il PBFT.

Ci sono infine altri due requisiti rilevanti di cui si è dovuto tener conto:

- Le normative che regolano la cessione del credito, in particolar modo il SuperBonus 110% essendo il più recente, sono soggette a **soventi modifiche circa requisiti, documentazioni e certificazione varie**. D'altra parte, come già accennato, la natura degli smart contract è tendenzialmente statica e immutabile. Per questo motivo si è cercato di trovare una soluzione che fosse abbastanza flessibile da poter in qualche modo "aggiornare" uno smart contract.
- L'applicazione doveva essere sviluppata considerando che sarebbe entrata a far parte di una piattaforma più grande di cui costituiva un singolo servizio. Questa **necessità di integrazione** ha spinto verso una soluzione che consentisse di separare il livello applicativo da quello relativo ai meccanismi specifici della gestione della blockchain.

Alla luce di tutte queste considerazioni si è scelto di sviluppare la blockchain utilizzando uno dei framework nel progetto **Hyperledger: Sawtooth**. Nella tabella successiva viene messo a confronto Sawtooth con altre popolari soluzioni blockchain.

| |  Bitcoin |  Ethereum |  Hyperledger Sawtooth |
|----------------------|---|--|--|
| Tipologia blockchain | pubblica | pubblica e privata (con quorum) | pubblica, privata, consorzio |
| Supporto smart | No | Si | Si |

| | | | |
|-------------------------------|---------------|--|--|
| contract | | | |
| Aggiornabilità smart contract | - | No | Si, aggiornando un Transaction Processor |
| Linguaggio smart contract | - | Solidity, Vyper | Python, C++, Go, Java, JavaScript, e Rust (Solidity supportato tramite la transaction family "Seth") |
| Algoritmo di consenso | Proof-of-work | - Raft (CFT*) - Istanbul Byzantine (BFT**) | - Devmode - Practical Byzantine Fault Tolerance (BFT**) - Proof of Elapsed Time (CFT*) - Proof of Elapsed Time SGX (BFT**) - Raft (CFT*) |
| Parallelizzazione transazioni | No | No | Si |
| Costo transazioni | Si | Sì su blockchain pubblica. No su blockchain privata (in quorum il costo del gas è 0) | No |

* Crash Fault Tolerant

** Byzantine Fault Tolerant

Hyperledger Sawtooth

Questo capitolo contiene una descrizione delle caratteristiche principali del framework scelto per lo sviluppo della blockchain [17].



Figura 3.1 - Logo del progetto Hyperledger Sawtooth
(<https://cn.hyperledger.org/sawtooth-logo-2>)

Introduzione

Hyperledger Sawtooth è una **piattaforma blockchain open-source enterprise**, nata dal multi-progetto Hyperledger che permette di creare applicazioni e reti basate su registro distribuito. Gli obiettivi che si propone di raggiungere questa piattaforma sono mantenere i registri distribuiti e rendere i contratti sicuri, in particolar modo per il caso d'uso aziendale. Si noti che in Sawtooth "registro distribuito" è utilizzato come sinonimo di blockchain: con questo termine si intende infatti un database di transazioni che abbia le caratteristiche di essere distribuito in una rete di nodi o peer, senza il bisogno di un amministratore o un' autorità centrale, immutabile a fronte di possibili tentativi di modifica dello storico di transazioni e sicuro, in quanto ogni transazione è firmata da una identità conosciuta. Sawtooth semplifica lo sviluppo di un'applicazione blockchain separando nettamente il sistema "core" del registro dallo specifico dominio applicativo. In questo modo gli sviluppatori possono lavorare sui requisiti specifici della loro applicazione e utilizzando il linguaggio che ritengono più appropriato senza dover necessariamente conoscere la struttura e i meccanismi dietro al sistema core del registro. Sawtooth è anche altamente modulare. La modularità permette ad un'organizzazione o un insieme di organizzazioni di implementare le particolari regole di business e policy, permettendo di scegliere e configurare tra le altre cose: le regole che governano le transazioni, il sistema di permessi e l'algoritmo di consenso. In questo modo Sawtooth tenta di venire incontro ad ogni particolare e specifica necessità di cui l'organizzazione può aver bisogno nello sviluppo dell'applicazione.

Caratteristiche distintive

Di seguito vengono descritte alcune caratteristiche peculiari di questo framework:

- **Separazione tra livello applicativo e sistema core.** Come già accennato, Sawtooth semplifica lo sviluppo di un'applicazione separando nettamente il livello applicativo che contiene le specifiche regole di business e il sistema core della blockchain che riguarda invece tutti i meccanismi dietro la gestione del registro distribuito (validazione dei blocchi, aggiornamento del registro, comunicazione tra i nodi, algoritmo di consenso ecc.). Sawtooth astrae il concetto di smart contract, permettendo allo sviluppatore di definirne la logica utilizzando uno tra molteplici linguaggi supportati e non vincolandolo a un modello o un linguaggio specifico (come accade ad esempio in Quorum/Ethereum con Solidity o Vyper). La separazione, spostando i dettagli implementativi di una determinata applicazione al cosiddetto "transaction-processing" layer, consente inoltre di avere molteplici diverse applicazioni che coesistono sulla stessa istanza di blockchain. Ogni applicazione definisce un proprio transaction processor, che di fatto rappresenta uno smart contract in Sawtooth,

modellato a seconda delle proprie esigenze che va ad interfacciarsi con il sistema core attraverso un validatore. Sawtooth mette a disposizione SDKs disponibili in diversi linguaggi (Python, Go, JavaScript, Java C++ e Rust) che permettono di sviluppare i transaction processor.

- **Sistema di autorizzazione per blockchain private.** I transaction processor Identity e Settings, sviluppati dal team di Sawtooth, permettono di gestire i permessi di esecuzione delle transazioni dei vari utenti, definendo identità e policy basate sulle chiavi pubbliche (il modo in cui un utente o un nodo viene identificato in sawtooth). In questo modo le informazioni di autorizzazione non sono centralizzate e affidate ad un unico servizio, cosa che necessiterebbe ulteriori accorgimenti in merito alla sicurezza, ma decentralizzate e distribuite come le altre informazioni sulla blockchain (on-chain configurations).
- **Esecuzione parallela delle transazioni.** Sawtooth include un avanzato scheduler che consente di eseguire transazioni parallelamente. Lo scheduler permette, quando necessario, di definire esplicitamente l'ordine di esecuzione delle transazioni in modo da garantire comunque la consistenza sui vari nodi. Lo scheduler si basa sia sugli indirizzi nello stato che le transazioni vanno a leggere o scrivere sia sulle informazioni di esplicita dipendenza tra transazioni, isolando l'esecuzione della singola transazione dalle altre ma conservando le modifiche che genera nello stato in maniera contestuale. Questo permette di capire quando è possibile parallelizzare l'esecuzione di transazioni (persino quando modificano lo stesso stato) evitando problemi di double-spending e aumentando le performance rispetto ad un'esecuzione sequenziale.
- **Algoritmo di consenso dinamico.** Diversamente da quanto accade nelle altre piattaforme blockchain, Sawtooth non è legato ad un determinato algoritmo di consenso. Come accade per quello di smart contract, anche il concetto di consenso viene astratto in Sawtooth isolandolo dalla semantica transazionale. In Sawtooth infatti il validatore è "fisicamente" (oltre che concettualmente) separato dal meccanismo di consenso: un determinato algoritmo di consenso è implementato come "consensus engine" (motore di consenso), un processo in esecuzione separato da quello di validazione ma che può interagire con quest'ultimo attraverso un'interfaccia di consenso che prende il nome di "consensus API". Questa separazione dà la possibilità di cambiare l'algoritmo di consenso anche dopo che la blockchain è stata creata ed è nella fase di produzione attraverso l'aggiornamento dell'appropriata on-chain setting. Attualmente sono disponibili diversi **consensus engine** che supportano il consensus API:
 - **Devmode:** Engine utilizzato per finalità di testing con un singolo validatore. Utilizza una versione semplificata dell'algoritmo random-leader. Non è adatto all'utilizzo in produzione
 - **PBFT:** Engine che implementa l'algoritmo di consenso Practical Byzantine Fault Tolerance, un algoritmo leader-based ideale per piccole reti private/consorzi che fornisce Byzantine Fault Tolerance senza bisogno di hardware specifico (vedi PoET SGX)
 - **PoET CFT:** Conosciuto anche come PoET Simulator. Il PoET (Proof of Elapsed Time) genera casualmente intervalli di tempo che i vari nodi della rete devono aspettare a fine di stabilire diritto di mining di questi sui blocchi. In questa versione l'algoritmo è eseguito in un ambiente Intel SGX simulato. Questo permette di

utilizzarlo su qualsiasi processore (anche non Intel e senza il supporto SGX), garantendo tuttavia la Crash Fault Tolerance ma non la Byzantine Fault Tolerance.

- **PoET SGX:** La versione del PoET che necessita di essere eseguita su processore Intel con supporto a SGX (Software Guard Extension). L'algoritmo sfrutta l'SGX per fornire Byzantine Fault Tolerance, come un algoritmo PoW, ma richiedendo un bassissimo utilizzo di CPU.
- **Raft:** Algoritmo di consenso leader-based che elegge un leader, al quale vengono concessi i diritti di mining, per un periodo di tempo casuale. Al termine del periodo di tempo o in caso vada in time-out, il leader viene sostituito. Raft è più veloce del PoET ma è solo CFT e non BFT [18].

Architettura della rete

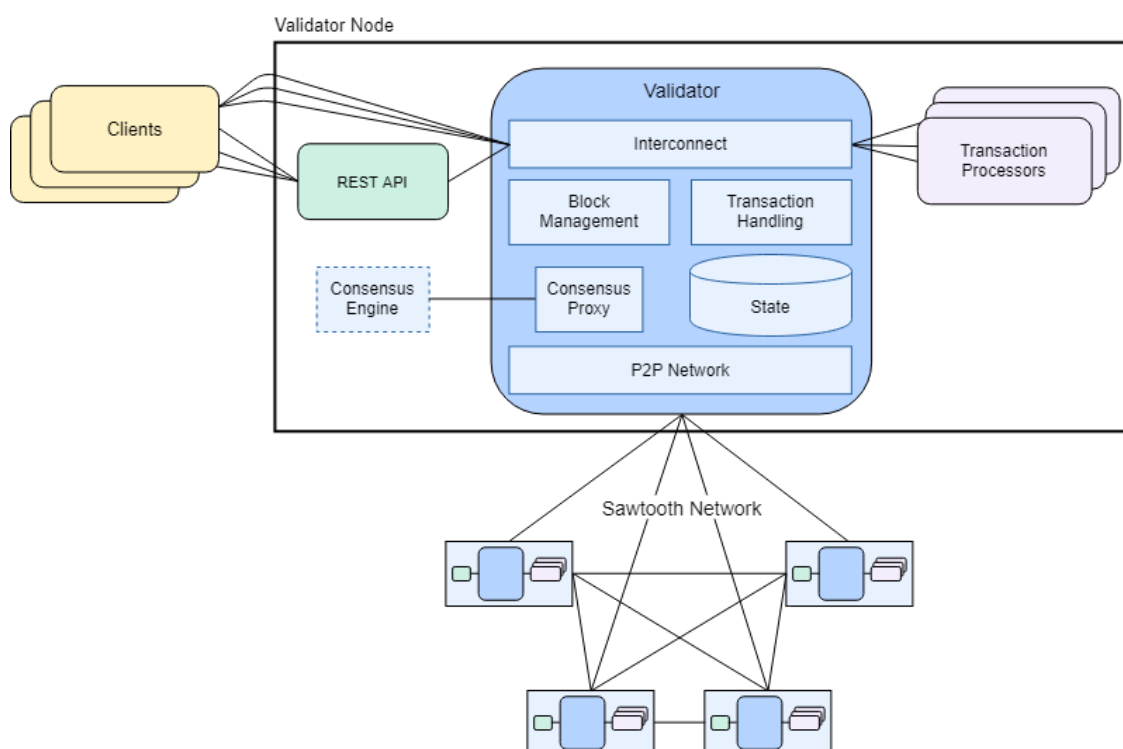


Figura 3.2 - Architettura della rete Sawtooth con dettaglio sul singolo nodo
(<https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture.html>)

Il diagramma in figura 3.2 mostra la struttura della blockchain sawtooth concentrandosi sulla struttura di un singolo nodo. Come si può notare un **nodo sawtooth** non è un blocco monolitico ma è formato da diversi componenti interconnessi tra di loro, ognuno dei quali ha un determinato ruolo e compito nei meccanismi di funzionamento della rete che può essere considerato come un processo/servizio a se stante ma che è in grado di comunicare con gli altri. Questo rende l'architettura altamente modulare e rispettosa del principio di separazione degli ambiti, aprendo molte possibilità in fase di implementazione della rete. In modo particolare, questo tipo di architettura non vincola l'intero nodo ad essere fisicamente istanziato su un'unica macchina: ad esempio il validatore di un nodo e il relativo consensus engine potrebbero essere in esecuzione su due server distanti migliaia di chilometri continuando, da un punto di vista logico, a far parte dello stesso nodo della rete. Questa caratteristica rende Sawtooth particolarmente adatto all'utilizzo in sistemi cloud ed è compatibile con i più famosi sistemi di containerizzazione e orchestrazione quali

Docker e Kubernetes. In sostanza quindi un nodo Sawtooth è un qualsiasi sistema host (computer fisico, virtual machine, insieme di container docker, Kubernetes pod) in cui è in esecuzione un validator, un consensus engine e, opzionalmente, una rest API e un insieme di transaction processor.

Come si vede in figura 3.2, tutte le altre componenti sono connesse ad uno stesso **validator** e i validator dei vari nodi sono connessi tra di loro a formare la rete. Il validator costituisce quindi il componente fondamentale e insostituibile di un nodo Sawtooth: esso è responsabile della validazione delle transazioni, della generazione e dell'aggiunta dei blocchi, del raggiungimento del consenso nella rete oltre che della memorizzazione della blockchain e dello stato globale, comunicando tra applicazioni client, transaction processor, consensus engine e i validatori. Lo scambio di messaggi tra i vari nodi è basato sul modello del "**gossip protocol**" (o **epidemic protocol**). Un "gossip protocol" è un paradigma di comunicazione ispirato al fenomeno di propagazione di un "pettegolezzo" all'interno di una rete sociale o equivalentemente di un virus in una comunità biologica. Questa tipologia di protocollo viene utilizzata spesso nelle reti peer-to-peer per garantire che un messaggio sia diffuso a nodi della rete in assenza di un registro centralizzato comune [19]. Ogni nodo deve chiaramente utilizzare lo stesso consensus engine, lo stesso insieme di transaction processor degli altri nodi che fanno parte della rete. Ogni nodo inoltre deve rendere pubblico un proprio indirizzo affinché non solo i vari componenti (in caso si usino i container) ma anche gli altri nodi siano in grado di raggiungerlo e sapere che fa parte della loro stessa rete. In genere **non c'è una gerarchia** (es. master/head node, slave node) una differenziazione di ruoli (es. full node, miner node, ecc.) tra i nodi di una stessa rete, solamente al momento della creazione della blockchain deve esserci un "primo nodo" che generi il blocco genesis stabilendo le configurazioni on-chain che verrà poi replicato dai restanti nodi della rete.

Transaction Family e Transaction processor

Uno delle caratteristiche peculiari di Sawtooth di cui si è fatta menzione è la netta separazione tra livello applicativo e sistema core. Questa separazione è ottenuta concretamente mediante le **transaction family**. Sebbene Sawtooth metta a disposizione alcune transaction family predefinite, ogni applicazione definisce le proprie transaction family in accordo ai propri unici requisiti.

Una transaction family è sostanzialmente un insieme di **logiche di business** specifiche dell'applicazione che definisce un insieme di operazioni e di tipologie di transazioni che sono consentite sulla blockchain. Essa è costituita da due componenti fondamentali:

- **Transaction Processor:** un processo/componente di un nodo della rete che implementa le operazioni definite dalla transaction family. Il transaction processor valida ed esegue le transazioni, applicando le relative modifiche allo stato della blockchain (vedi Global State). Esso di fatto rappresenta l'implementazione concreta di una transaction family e deve essere progettato dallo sviluppatore dell'applicazione in accordo ai requisiti specifici.
- **Modello di dati:** Sawtooth lascia ampia libertà riguardo al formato dei dati manipolati dalle transazioni di una transaction family. Per questo motivo la transaction family stessa deve definire un modello, un formato (es. CSV, JSON, CBOR, Protobuf, ecc.) e i relativi meccanismi di serializzazione/deserializzazione, sia per i dati che vengono memorizzati nello stato globale sia per il payload delle transazioni.

Una transaction family può essere aggiunta alla rete in qualsiasi momento eseguendo una istanza del suo transaction processor per ogni nodo della rete e connettendolo al corrispondente validator. Inoltre, come si vedrà nella sezione relativa al Global State, per evitare sovrapposizioni, ogni transaction family definisce un sottospazio nello stato globale della blockchain che può essere

modificato dalle relative transazioni. Le transaction family (e più in particolare i relativi transaction processor) sono quindi l'equivalente degli smart contract utilizzati in altri framework blockchain [20].

Appare evidente come, grazie a questo approccio, lo sviluppatore dell'applicazione debba preoccuparsi unicamente della logica di business e dei data memorizzati nello stato della propria applicazione, lasciando tutte le altre dinamiche della gestione della blockchain (validazione dei blocchi, consenso, aggiornamento della blockchain, scheduling delle transazioni, comunicazione con gli altri nodi, ecc.) al sistema core (validator).

Global State

Uno degli obiettivi, probabilmente quello più importante, dei registri distribuiti come Sawtooth è quello di mantenere copie consistenti dello stesso insieme di informazioni distribuite tra i nodi di una rete. In Sawtooth le transazioni non vengono semplicemente validate ed aggiunte alla blockchain ma vengono anche **"eseguite"** utilizzando l'appropriato transaction processor. L'esecuzione di una transazione provoca un cambiamento, un'**evoluzione dello stato della rete**. Questo stato, che prende il nome di **global state**, è una delle due importanti strutture dati (insieme alla catena di blocchi che costituisce la blockchain) che viene memorizzata e gestita da ogni validatore in ogni nodo. Lo stato è globale in quanto comprende tutti i dati di ogni transaction family utilizzata nella rete ed è suddiviso in namespace per garantire una certa flessibilità e riusabilità nella definizione delle transaction family. Il global state viene rappresentato con una singola istanza di un **Merkle-Radix tree indirizzabile**.

Un **Merkle tree o hash tree** è una particolare struttura ad albero molto utilizzata in crittografia. Consiste in un albero in cui ogni nodo foglia è etichettato con l'hash calcolato da un blocco di dati, mentre tutti gli altri nodi interni sono etichettati con l'hash calcolato a partire dagli hash dei suoi nodi figli, in particolare quindi il nodo radice sarà etichettato con un hash che dipende da quello di ogni nodo (direttamente o indirettamente). Questo tipo di struttura dati viene in genere utilizzata per verificare in maniera sicura ed efficiente l'integrità del contenuto di grandi strutture dati. La struttura ad albero permette soprattutto di verificare l'appartenenza o la variazione di un singolo blocco dati corrispondente ad un particolare nodo foglia, calcolando l'hash del nodo radice, senza dover conoscere il contenuto degli altri nodi [21].

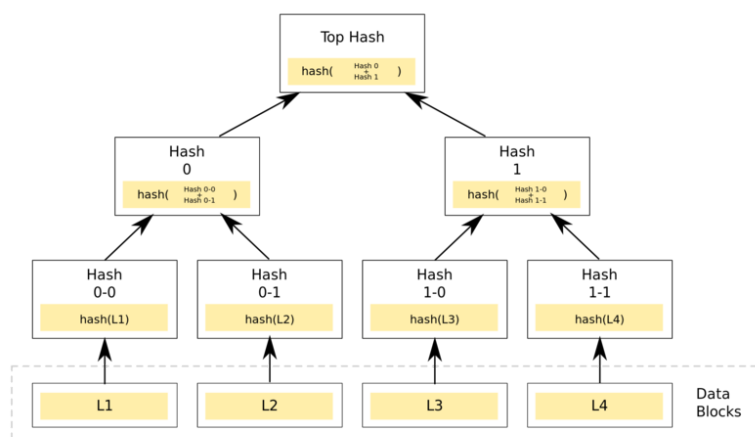


Figura 3.3 - Esempio di Merkle Tree ([21])

Nel caso del global state i blocchi di dati sono quelli relativi alle transaction family e l'hash del nodo radice viene utilizzato nei meccanismi di validazioni di blocchi di transazioni: dato un insieme di transazioni di stato derivanti da transazioni associate ad un blocco si calcola l'hash della radice del Merkle tree dello stato dopo che tutte le transazioni sono avvenute: questo hash, che rappresenta in maniera "condensata" una "versione" dell'albero e quindi uno stato dell'intera struttura dati, viene

inserito nell'header del blocco in questione. In questo modo i validatori raggiungono il consenso non solo sulle transazioni incluse nel blocco e sul loro ordinamento ma anche su quale sia lo stato globale dei dati delle transaction family dopo che quelle transazioni sono state eseguite. Questo significa che se l'hash dello stato calcolato da due nodi risulta diverso l'intero blocco viene considerato non valido.

L'albero del global state è anche un **Radix tree indirizzabile** in quanto ogni path dal nodo radice a un nodo foglia in cui sono memorizzate i dati è identificato univocamente da un indirizzo. L'indirizzo di un nodo è una stringa di 70 caratteri esadecimali che rappresenta 35 byte. Ogni byte (2 caratteri esadecimali) costituisce un segmento dell'indirizzo che a partire dalla radice identifica il nodo successivo nel percorso che conduce al nodo foglia indirizzato.

Da ciò si può dedurre che ogni nodo dell'albero può avere fino a 2^8 nodi figli. Il formato degli indirizzi prevede che i primi 3 byte (6 caratteri esadecimali) siano riservati per identificare il namespace relativo ad una transaction family, permettendo di avere 2^{24} (16,777,216) possibili namespace differenti in una stessa istanza di blockchain. I restanti 32 byte (64 caratteri esadecimali) possono invece essere codificati secondo le esigenze dello sviluppatore della transaction family ad esempio suddividendo ulteriormente il namespace o mappando porzioni di indirizzo con identificatori specifici del dominio di applicazione di quel determinato blocco di dati.

Namespace prefix is 3 bytes.

Namespace-specific address portion is 32 bytes. The encoding rules for this portion are determined by the namespace design.

```
97fd77f7eabdd2c77fd852cf6c16a7a4429ab27aff394493ab7c41609c759ddb696f2e
```

Full address is 35 bytes, represented as a 70 character hex string.

Figura 3.4 - Esempio di indirizzo del Merkle-Radix Tree

(https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/global_state.html)

L'indirizzo di nodo è utilizzato per referenziare determinato blocco di dati nelle operazioni di lettura (get) e scrittura (set) dei dati nello stato globale eseguite dal transaction processor.

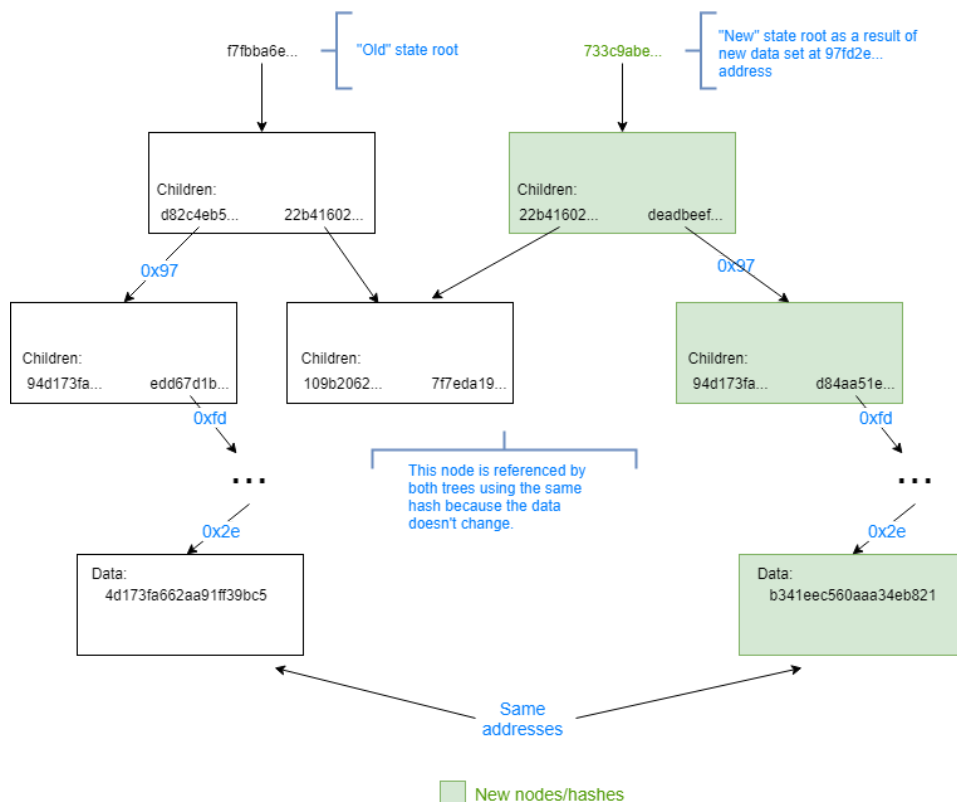


Figura 3.5 - Aggiornamento del Merkle-Radix Tree dopo una modifica dello stato globale
(https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/global_state.html)

Anche per quanto riguarda il formato del blocco di dati associato ad un node Sawtooth lascia ampia libertà allo sviluppatore della transaction family: i dati vengono memorizzati ad un dato indirizzo semplicemente come un array di byte; in questo senso quindi il contenuto dello stato è opaco al validatore ed è responsabilità dello sviluppatore fornire e utilizzare un proprio schema di serializzazione e deserializzazione dei dati specifico per il dominio applicativo che sia inoltre deterministico tra più esecuzione della stessa transazione per evitare che validatori di due nodi producano un stato globale diverso.

Journal e blockchain

Oltre al global state l'altra struttura dati fondamentale in Sawtooth è la blockchain stessa intesa proprio come la catena di blocchi di transazioni. Questa struttura dati viene gestita dal validatore attraverso il **journal**, ovvero l'insieme delle sue sottocomponenti che cooperano per verificare e validare i batch di transazioni e proporre i blocchi da aggiungere alla blockchain. I componenti del journal sono direttamente responsabili della costruzione dei blocchi, a partire dai batch di transazione, della loro validazione e "pubblicazione" sulla blockchain, avendo come obiettivo ultimo quello di **estendere la catena di blocchi**.

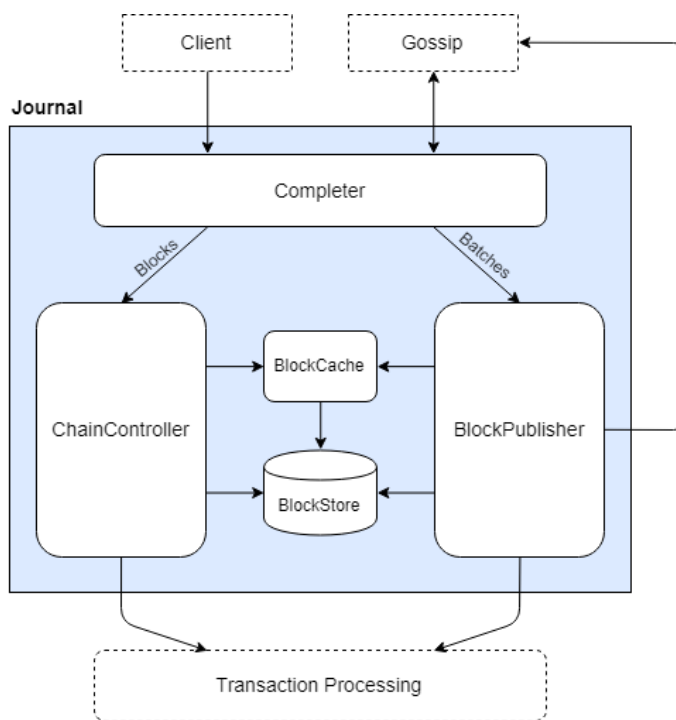


Figura 3.6 - Struttura interna del componente Journal
(<https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/journal.html>)

Sia i blocchi che i batch arrivano al validatore o direttamente come richieste del client o dagli altri nodi attraverso il gossip protocol e vengono gestiti in diverse pipeline. In linea di massima il workflow è questo:

- Il Completer riceve inizialmente i blocchi e i batch andando ad accertarsi che tutte le dipendenze dei blocchi e dei batch siano soddisfatte, potendo quindi considerarsi completi.
- I batch completi vengono passati al Block Publisher che li valida e li aggiunge ad un blocco.

- I blocchi completati vengono passati a Chain Controller per la validazione e la risoluzione di eventuali fork.
- BlockCache e Blockstore supportano gli altri componenti memorizzando i batch e i blocchi mentre vengono processati e dopo la loro aggiunta alla blockchain.

Il processamento dei blocchi e dei batch avviene in maniera asincrona e sul modello della pipeline. Questo consente al Chain Controller di processare in blocchi in arrivo in parallelo e al Block Publisher di crearli dai batch anche quando la frequenza dei blocchi in arrivo è elevata. Inoltre il workflow è abbastanza flessibile e generalizzato da permettere di lavorare con diversi algoritmi di consenso attraverso la consensus interface che permette ai componenti del journal di comunicare con il consensus engine.

Di seguito vengono analizzati più nel dettaglio i componenti del journal:

- **Completer:** Quando un blocco proposto viene propagato nella rete contiene solo un minima parte di informazioni, ad esempio sono presenti gli id dei batch di transazioni ma non i batch stessi. Il Completer, che riceve questi blocchi e i batch, ha il compito di assicurarsi che siano completi prima di passarli al BlockPublisher o al Chain Controller. Il Completer controlla la presenza di eventuali dipendenze nelle transazioni, si assicura che il blocco precedente esista e che il batch sia presente nel Blockstore o nella BlockCache.
 - Un batch è considerato completo quando tutte le transazioni da cui dipendono le transazioni che costituiscono il batch sono presenti nella blockchain o sono state consegnate al Block Publisher. Una volta che il batch è formalmente completo viene mandato al Block Publisher.
 - Un blocco è considerato completo quando tutti i blocchi che lo precedono sono stati inviati al Chain Controller e il campo "batches" contiene tutti i batch indicati dai relativi id nell'header del blocco. Una volta che il blocco è formalmente completo viene mandato al Chain Controller per la validazione.

Sia per i blocchi che per i batch viene stabilito un timeout per il loro completamento: il Completer invia una richiesta iniziale per ogni dipendenza non soddisfatta nei batch o per i blocchi che precedono il blocco processato; se non riceve una risposta entro il tempo stabilito il blocco o il batch viene messo da parte con la possibilità di riprendere il processo di completamento quando un blocco successivo lo richiede come dipendenza del proprio processo di completamento.

- **Block Publisher:** Il Block Publisher è responsabile della creazione del blocco candidato ad estendere la catena di blocchi seguendo le indicazioni dell'algoritmo di consenso riguardo in particolare a quando creare il blocco e quando pubblicarlo sulla blockchain. In sostanza interviene al verificarsi di questi eventi: inizio della creazione del blocco, ricezione di una batch di transazioni, sintesi del blocco, pubblicazione del blocco.
- **ChainController:** Il ChainController è vero e proprio responsabile della gestione della blockchain per conto del validatore. Questo implica validare i blocchi proposti e stabilire se i blocchi validati possano essere aggiunti alla blockchain. Inoltre il Chain Controller stabilisce, in caso di fork, il ramo della blockchain su cui il validatore deve lavorare e coordina tutte le operazioni di modifica che coinvolgono la blockchain.

- **BlockStore:** Il BlockStore costituisce la memoria persistente su cui è memorizzata la lista che contiene tutti i blocchi che costituiscono la blockchain, dall'ultimo blocco aggiunto al blocco genesis, esclusi i blocchi che fanno parte di branch orfani. Il contenuto del Blockstore è di fatto considerato come lo "stato ufficiale" in cui si trova la blockchain quando il validatore viene avviato. Oltre a questo il Blockstore conserva le informazioni che permettono di mappare ogni transazione e ogni batch al rispettivo blocco. In questo modo un blocco può essere referenziato sia con il suo id sia con l'id di uno dei suoi batch o l'id di una delle sue transazioni. Queste informazioni sono utili sia a scopo di consultazione della "storia" della blockchain sia in caso di corruzione o perdita dei blocchi della blockchain per ricostruirli. Quando si verificano fork, il Blockstore gestisce il passaggio da un branch ad un altro utilizzando liste di blocchi da aggiungere (commit) e liste di blocchi da rimuovere (de-commit) in modo da eseguire tutti gli aggiornamenti della blockchain in maniera atomica.
- **Block Cache:** La Block Cache è una struttura dati in-memory (quindi volatile) che memorizza temporaneamente i blocchi e i batch che vengono utilizzati nei diversi processi dagli altri componenti del journal in modo da garantire un accesso più rapido. La Block Cache tiene traccia dello stato di ogni blocco etichettandolo come:
 - Valid: un blocco che è stato validato con successo dal Chain Controller. Questo implica che ogni blocco nel BlockStore è necessariamente in questo stato.
 - Invalid: un blocco che ha fallito la validazione del Chain Controller o il cui predecessore è a sua volta un blocco non valido.
 - Unknown: un blocco che non è ancora stato validato.

La Block Cache tiene traccia di tutti i blocchi che si ritengono rilevanti in base al periodo di tempo trascorso dall'ultima richiesta di accesso a quel blocco. Periodicamente i blocchi che non sono stati utilizzati di recente vengono eliminati dalla cache a patto che non ci siano altri blocchi più recenti che non li referenziano come predecessori.
- **Consensus Interface:** Come già accennato tra le caratteristiche distintive di Sawtooth c'è quella dell'algoritmo di consenso dinamico: l'algoritmo di consenso viene stabilito alla creazione primo blocco della catena (blocco genesis) come on-chain setting e può essere cambiato in seguito utilizzando il transaction processor Settings. Questo genere di flessibilità si ottiene grazie all'utilizzo di un'interfaccia di consenso. Il validatore attraverso la consensus interface determina quale nodo ha il diritto di pubblicare un blocco, se un blocco è valido in base alle regole di consenso e quale branch deve essere considerato quello principale in caso di fork.

Questo è quindi il processo standard per la creazione, validazione e aggiunta di un blocco alla blockchain. Esiste tuttavia un'eccezione ovvero la creazione del blocco genesis. La presenza di un **blocco genesis** è fondamentale nel processo di creazione di una nuova blockchain Sawtooth in quanto stabilisce tutte le **on-chain settings iniziali** (tra cui l'algoritmo di consenso). Il blocco genesis contiene una lista di batch che il validatore processa al momento della creazione della blockchain. Queste transazioni inizializzano alcune configurazioni on-chain (ad esempio il consensus engine, gli utenti che possono modificare le queste impostazioni, le transaction family utilizzabili nella rete, il numero massimo di transazioni per blocco, ecc.) e sono eseguite da un utente amministratore. Questo permette alle transaction family di includere i propri batch di transazioni senza dover tenere conto dei dettagli del processo di creazione. Il processo di creazione segue questi 4 passaggi:

1. Creazione del batch genesis contenente le transazioni iniziali della blockchain

2. Creazione del blocco genesi utilizzando il batch genesi
3. Avvio dei transaction processor
4. Validazione e pubblicazione del blocco genesi

Transaction Scheduling

Sawtooth supporta lo **scheduling** delle transazioni sia in **parallelo** che **sequenziale**. Entrambi i tipi di scheduler hanno un comportamento deterministico ciò significa che la differenza tra i due scheduler è unicamente nelle prestazioni in quanto il risultato sarà lo stesso indipendentemente dallo scheduler utilizzato. Lo scheduler parallelo è uno dei valori aggiunti di sawtooth e consente di migliorare le performance del processo di validazione anche quando il carico di lavoro è minimo, andando a ridurre gli effetti di latenza che si ha quando lo scheduling delle transazioni è sequenziale. I veri benefici dello scheduler parallelo però si vedono quando la durata delle singole transazioni non è uniforme e il numero di transazioni più "rapide" è sensibilmente maggiore di quelle più "lente", dal momento che la parallelizzazione evita che queste ultime costituiscano un collo di bottiglia (cosa che accadrebbe con lo scheduling sequenziale).

In altre tipologie di registri distribuiti in genere, per evitare che le modifiche si sovrappongano, si ha il vincolo di al massimo una transazione per blocco che modifica lo stesso indirizzo nello stato, limitando considerevolmente le prestazioni. Una caratteristica peculiare dello scheduler di Sawtooth invece è di non avere questo tipo di limite in quanto in grado di gestire in maniera corretta ed efficiente le transazioni che modificano lo stesso indirizzo e che si trovano non solo nello stesso blocco ma addirittura nello stesso batch. Ciò è reso possibile dal fatto che l'evoluzione dello stato globale non è calcolata blocco per blocco ma transazione per transazione.

Scheduling Sequenziale vs Parallelo

Lo scheduler stabilisce la successiva transazione da eseguire in maniera dinamica basandosi sul grafo di dipendenza della transazione e sulle transazioni eseguite in precedenza nello stesso scheduling. Il modo in cui il grafo di dipendenza viene calcolato e in cui viene fornita stabilita la prossima transazione da eseguire varia a seconda del tipo di scheduler:

- **Scheduler Sequenziale:** Lo scheduler sequenziale fornisce la transazione successiva solo quando ha ricevuto il risultato dell'esecuzione di quella precedente. Il grafo di dipendenza in questo è molto semplice in quanto ogni transazione dipende da quella che è stata eseguita in precedenza, essendo le transazioni eseguite nella sequenza in cui vengono ricevute.
- **Scheduler Parallelo:** Lo scheduler parallelo genera il grafo di dipendenza delle transazioni determinando per ogni transazione in un batch la transazione che la precede (cioè quella che deve essere necessariamente eseguita prima che venga eseguita questa), via via che i batch sono aggiunti allo scheduler. Per fare ciò utilizza gli indirizzi definiti nei campi di input e output nell'header della transazione definiti dall'applicazione client. Questi indirizzi referenziano i blocchi di dati dello stato a cui la transazione deve poter accedere in lettura (input) e in scrittura (output) durante la sua esecuzione. Considerando transazioni che leggono e/o scrivono sullo stesso indirizzo, lo scheduler determina relazioni di dipendenza di una transazione da un'altra stabilendo i "predecessori" di ogni transazione. A differenza dello scheduler sequenziale, l'ordine delle transazioni fornite non è predeterminato: solo le transazioni che non hanno conflitti nell'accesso in lettura/scrittura allo stato vengono eseguite in parallelo. Quando viene richiesta la prossima transazione da eseguire lo scheduler fornisce la prima transazione in lista i cui predecessori hanno terminato la propria esecuzione. Nel caso non venga trovata alcuna transazione che soddisfa questo requisito,

lo scheduler si sospende e attende che una delle transazioni eseguite in parallelo abbia terminato la propria esecuzione, per poi controllare nuovamente.

Scheduling all'interno del validatore

Il processo di scheduling all'interno del validator avviene in due componenti di cui si è già parlato precedentemente: il Chain Controller e il Block Publisher. Queste due componenti utilizzano istanze dello scheduler, (che possono essere molteplici) create in maniera dinamica quando necessario che "istruiscono" un altro componente del processo di validazione, l'Executor, riguardo all'ordine stabilito per l'esecuzione delle transazioni.

- **Chain Controller:** Nel processo di scheduling il Chain Controller, ricevuto un blocco candidato dalla rete, istanzia uno scheduler per determinare l'evoluzione dello stato globale e il relativo Merkle tree, provocata dal blocco candidato. Compara quindi l'hash della radice del Merkle tree con quello presente nell'header del blocco: se l'hash corrisponde il blocco è considerato valido. Il Chain controller utilizza questa informazione in combinazione con l'algoritmo di consenso per decidere se aggiornare la blockchain.
- **Block Publisher:** Il Block Publisher è responsabile della creazione di nuovi blocchi candidati. Quando riceve i batch dal validatore o da altri nodi, li aggiunge ad una coda. Solo le transazioni valide possono essere aggiunte ad un blocco candidato. Per rendere il processo più tempestivo i batch vengono aggiunti sia alla coda di quelli in attesa sia ad uno scheduler istanziato dal Block Publisher, in questo modo le transazioni vengono processate in maniera incrementale non appena ricevute.
- **Executor:** L'Executor ha il compito di eseguire le transazioni inoltrandole al rispettivo transaction processor. Per garantire una corretta e deterministica evoluzione dello stato globale a seguito dell'esecuzione delle transazioni, l'Executor utilizza uno scheduler e un Context Manager che fornisce il contesto in cui le transazioni vengono eseguite. Il flusso di gestione dell'esecuzione è il seguente:
 1. L'Executor riceve la transazione da eseguire e il relativo contesto iniziale dallo scheduler.
 2. L'Executor riceve un nuovo contesto per la transazione dal Context Manager a partire da quello iniziale.
 3. L'Executor invia la transazione e il riferimento al contesto al transaction processor
 4. Il transaction processor esegue la transazione aggiornando lo stato del contesto contattando il context manager.
 5. Il transaction processor notifica l'Executor nel momento in cui l'esecuzione della transazione è terminata.
 6. L'Executor aggiorna lo scheduler con il risultato della transazione che contiene il contesto aggiornato.

Nel caso di scheduling sequenziale lo step 1 per la transazione attuale si sospende fin quando lo step 6 di quella precedente non è stato completato. Nel caso di scheduling parallelo lo step 1 non si blocca fin quando ci sono transazioni da eseguire che abbiano le relative dipendenze soddisfatte. Questo perché gli step 2-6 avvengono in parallelo per ogni transazione che viene eseguita.

Transactions and Batches

Le **transazioni** sono definibili come funzioni che cambiano lo stato globale della blockchain. L'applicazione client crea una transazione e la invia al validatore direttamente o indirettamente tramite la Rest API. Il validatore si preoccupa di eseguire la transazione che causa il cambiamento di stato. In Sawtooth le transazioni sono sempre accorpate a formare il cosiddetto batch. Il **batch** rappresenta una modifica atomica dello stato: l'evoluzione dello stato globale evolve se tutte le transazioni di un batch sono valide come conseguenza dell'esecuzione di ogni singola transazione oppure non evolve affatto se anche una sola delle transazioni non risulta valida.

Transaction

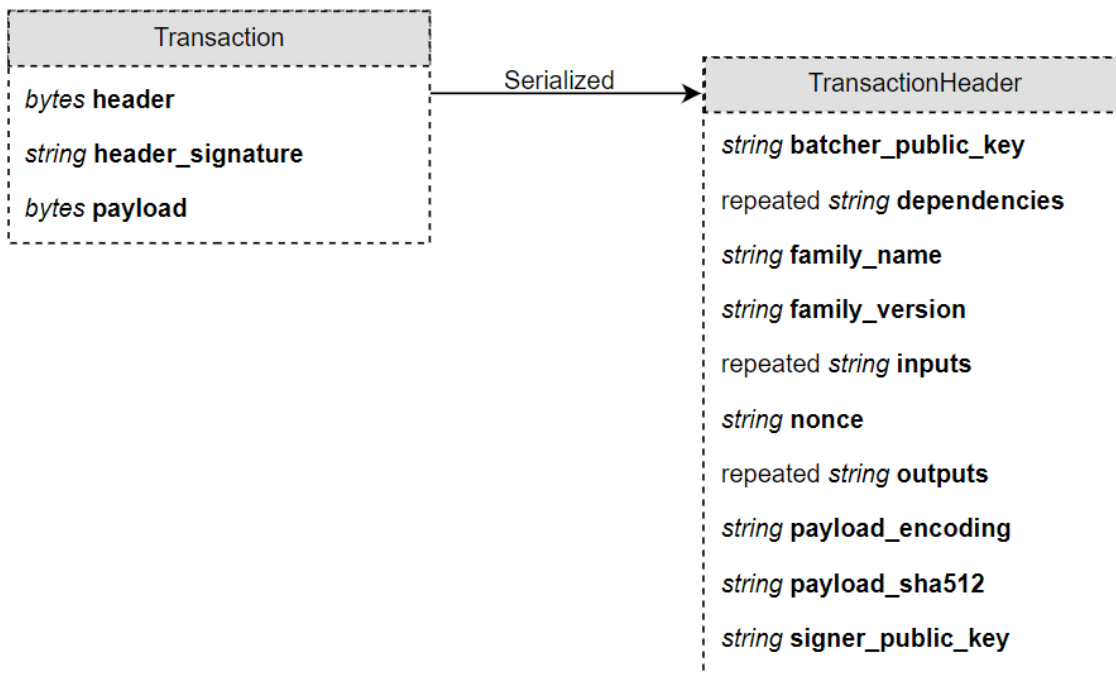


Figura 3.7 - Struttura di una transazione

https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/transactions_and_batches.html

In figura 3.7 viene mostrata sinteticamente la struttura generale di una transazione e del relativo header. Il campo header contiene una versione serializzata del **Transaction Header**. L'header viene inoltre firmato digitalmente utilizzando la chiave privata dell'utente e il la firma viene memorizzata nel campo **header_signature**. L'header è fornito serializzato in maniera tale da poter verificare la firma digitale non appena la transazione viene ricevuta, questo comporta:

- Verificare che la chiave pubblica presente nel campo `signer_public_key` sia quella derivata dalla chiave privata che ha firmato l'header.
- Verificare che la chiave pubblica presente nel campo `batcher_public_key` sia quella derivata dalla chiave privata che ha firmato il batch in cui la transazione è contenuta.

Riguardo alla firma digitale Sawtooth supporta unicamente l' **Elliptic Curve Digital Signature Algorithm (ECDSA)** con chiavi che usano la curva **secp256k1**. Il validatore si aspetta firme digitali compatte normalizzate di 64-byte (ovvero il risultato della concatenazione dei campi R ed S della firma). I campi **family_name** e **family_version** contengono rispettivamente il nome univoco della transaction family relativa alla transazione e la versione. Il campo versione è stato introdotto per

gestire possibili upgrade di una transaction family. I campi **input**, **output** e **dependencies** vengono invece utilizzati dallo scheduler, come spiegato precedentemente, per determinare l'ordine di esecuzione delle transazioni. I campi input e output contengono una lista di indirizzi dello stato a cui la transazione accede in lettura (input) e in scrittura (output) quando viene eseguita. Gli indirizzi possono essere completi o parziali, questi ultimi specificano un sottoalbero del Merkle-Radix Tree dello stato globale anziché un singolo nodo foglia. Il campo dependencies viene utilizzato per indicare esplicite dipendenze della transazioni da altre transazioni (predecessori) e contiene una lista di id di queste transazioni. Il campo **payload** contiene il contenuto vero e proprio della transazione serializzato, ovvero l'insieme di informazioni che il transaction processor utilizza per aggiornare lo stato eseguendo la transazione. Il campo **payload_sha512** contiene l'hash SHA-512 dei byte del payload e permette di verificare l'integrità del payload quando viene ricevuto. A differenza del payload (non incluso nell'header) il campo payload_sha512 viene firmato e verificato in quanto parte dell'header. Infine il campo **nonce** contiene un valore random generato dal client: la presenza di questo campo garantisce che due transazioni identiche (a meno del nonce) generino due firme dell'header diverse

Batch

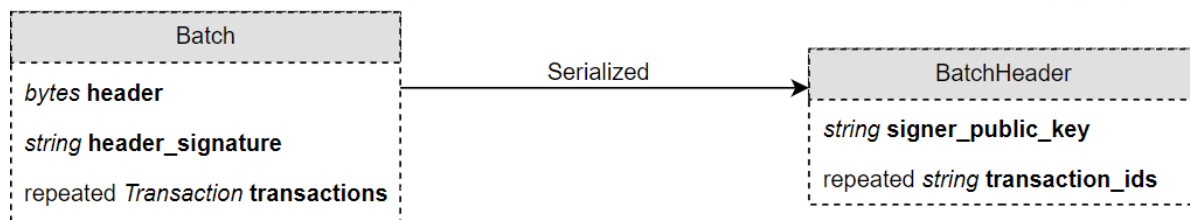


Figura 3.8 - Struttura di un Batch di transazioni

(https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture/transactions_and_batches.html)

In figura 3.8 viene mostrata sinteticamente la struttura generale di un batch e del relativo header. La struttura del Batch segue lo stesso pattern descritto per la Transaction. L'header viene firmato utilizzando la chiave privata dell'utente e il risultato della firma è inserito nel campo **header_signature**. Anche in questo caso l'header è fornito serializzato in maniera tale da poter verificare la firma digitale non appena il batch viene ricevuto e comporta la stessa serie di controlli elencati per la Transaction. Il campo transactions contiene la lista di transazioni che costituiscono il batch. Il campo **transaction_ids** contiene una lista di Transaction **header_signature** che corrispondono ordinatamente alle transazioni del campo transactions.

Oltre al meccanismo di modifica atomica dello stato di cui si è parlato precedentemente esistono ulteriori motivi per l'utilizzo dei batch:

- Il batch semplifica notevolmente la gestione delle dipendenze dal punto di vista del client, poiché non è necessario dichiarare esplicitamente le dipendenze per le transazioni di uno stesso batch. Infatti l'unico caso in cui la dipendenza esplicita (si intende il campo dependencies del Transaction header) deve essere fornita da client è quello fra transazioni che non appartengono allo stesso batch.
- I batch risolvono un importante problema non risolvibile con le dipendenze esplicite. Si supponga ad esempio di avere le transazioni A, B e C tali che l'ordine in cui devono essere eseguite è $A \rightarrow B \rightarrow C$ e che se anche solo una non è valida nessuna delle altre deve generare un cambiamento di stato. Se si cerca di indicare l'ordine con le dipendenze esplicite si può provare stabilendo queste relazioni: C dipende da B, B dipende da A e A

dipende da C. Come si può notare si stabilisce però una dipendenza ciclica tra le transazioni che non permette di stabilire un ordine di esecuzione delle transazioni.

- Il batch consente di aggregare insieme transazioni di transaction family diverse favorendo in questo modo il loro riutilizzo (in particolar modo di quelle predefinite di Sawtooth)
- I batch e le transazioni possono essere firmati con chiavi private diverse e quindi da utenti/componenti che potrebbero avere ruoli diversi nell'applicazione. Questo apre a scenari e pattern interessanti: ad esempio un'applicazione web potrebbe dare la possibilità ad un utente di firmare le singole transazioni mentre un componente del server potrebbe essere responsabile di creare i batch delle transazioni degli utenti e firmarli.

L'applicazione

In questo capitolo verranno descritti i punti salienti della progettazione dell'applicazione, dalla raccolta e analisi dei requisiti fino alla realizzazione. L'applicazione è stata sviluppata utilizzando la **metodologia Scrum**⁴. Lo team scrum ha visto partecipare:

- **Product Owner:** Michele Mencarelli
- **Scrum Master:** Mattia Luzi
- **Team Sviluppo:** Mattia Luzi, Giacomo Giuliani (come sviluppatori e progettisti software), Margherita Bolletta e Agnese Giacometti (come esperti di dominio in merito alla cessione del credito edilizio e al SuperBonus 110)

Si è scelto di pianificare **sprint** (cicli di rilascio) settimanali, in modo da avere quasi sempre una versione dell'applicazione funzionante, accompagnati da sprint meeting della durata di 2-3 ore circa in cui discutere le funzionalità prioritarie del rilascio successivo. Inoltre per coordinare i membri del team nello sviluppo è stata di fondamentale importanza una kaban gestita mediante il software gestionale **Trello**.

Raccolta dei requisiti

Definizione delle fonti

Lo sviluppo dell'applicazione è avvenuto in collaborazione con l'azienda **MC Energy GTS srl** che, nelle persone del CEO Michele Mencarelli, dell'Ingegnere Edile Margherita Bolletta e del tecnico Agnese Giacometti, ha fornito le conoscenze di dominio riguardanti il SuperBonus 100 e la cessione del credito. Altra fonte importante è stata sicuramente il Professor Luca Spalazzi in merito a sicurezza informatica e tecnologia blockchain. I requisiti sono stati raccolti durante le riunioni settimanali dello sprint meeting, avvenute da remoto.

Classificazione dei requisiti

Inizialmente sono stati raccolti dei requisiti preliminari che descrivono in generale le caratteristiche desiderate per l'applicazione, i problemi che deve aiutare a risolvere e quali sono gli utenti che la utilizzano. Questi requisiti preliminari sono stati classificati, raggruppandoli in requisiti utente e requisiti di sistema.

Di seguito vengono elencati sinteticamente i principali requisiti utente⁵:

- Il software deve poter essere utilizzato da utenti con modeste capacità tecniche, che tengono più alla praticità nella user experience piuttosto che al design e all'estetica dell'applicazione.
- L'utilizzo dell'applicazione avviene unicamente indoor (in ufficio) attraverso un browser.
- Gli utenti che utilizzano l'applicazione sono raggruppabili in 4 categorie:
 - **Cedente:** l'utente che utilizza l'applicazione per cedere il credito che ha maturato o maturerà in seguito a un progetto che rispetta i requisiti previsti per il SuperBonus

⁴ Scrum è una metodologia per la gestione del ciclo di sviluppo del software, iterativo ed incrementale, concepito per gestire progetti e prodotti software o applicazioni di sviluppo.

⁵ I requisiti utente fanno riferimento agli utenti che devono interagire con il software, tenendo presente quindi le loro competenze tecniche (utente esperto, occasionale, etc.), l'ambiente in cui dovranno utilizzarlo (per quanto concerne luminosità, rumorosità, indoor o outdoor) ed infine ovviamente l'obiettivo che ha l'utente nell'utilizzo del software.

110. In genere parliamo di imprese edilizie che realizzano i lavori o privati (es. amministratori condominiali per conto di condomini).

- **Acquirente:** l'utente che acquista il credito da un cedente e usufruisce del rimborso fiscale previsto per il SuperBonus 110. In genere parliamo di grandi contribuenti (es. Banche, Istituti Finanziari) o di privati con discreti volumi di affari. L'acquirente può far parte di un determinato gruppo acquirente per conto del quale riceve le proposte di cessione del credito e le acquista oppure acquistare attraverso il meccanismo di borsa del credito.
- **Revisore Fiscale:** un utente che svolge la professione di revisore fiscale per conto di un gruppo acquirente. Il revisore fiscale interviene nel processo di validazione della proposta di cessione. Questa figura potrebbe essere non essere presente in un gruppo acquirente, in questo caso il suo ruolo viene ricoperto da un acquirente.
- **Validatore:** un utente che ha come unico compito di validare le richieste di accreditamento dei cedenti per la borsa del credito controllando la loro eleggibilità a cedere il credito.

Di seguito vengono elencati sinteticamente i principali requisiti di sistema⁶:

- **Requisiti funzionali:**

- L'applicazione ha come obiettivo principale quello di permettere la vendita del credito d'Imposta (cessione del credito) relativo al Super Bonus 110, tracciando in maniera sicura e verificabile tutte le informazioni relative ad una cessione del credito e garantendo il più possibile trasparenza. L'applicazione deve costituire un mezzo che permette quindi a queste due figure il cedente e l'acquirente di mettersi in contatto e gestire la cessione.
- L'applicazione deve permettere di gestire la cessione del credito secondo due modalità:
 - **Cessione diretta:** il cedente propone la cessione direttamente a un determinato gruppo acquirente che ha già precedentemente dichiarato le proprie condizioni di cessioni e che il cedente può consultare. Le condizioni di cessioni riguardano principalmente la percentuale che il gruppo è disposto a pagare sul valore totale del progetto in cambio del credito da esso generato e i documenti da presentare nella proposta.
 - **Cessione borsa del credito:** il cedente propone la cessione alla borsa del credito (questa viene di fatto considerata come un gruppo acquirente "speciale"). Il meccanismo di acquisto della proposta di cessione segue all'incirca le regole di un' asta: gli acquirenti avanzano delle offerte di acquisto per una proposta di cessione e il cedente può decidere quali tra queste accettare. Gli acquirenti possono vedere le offerte altrui e modificare/eliminare la propria fino a quando il cedente non decide quale accettare.
- L'applicazione deve garantire che i cedenti siano eleggibili per la cessione del credito e che i progetti presentati con le proposte siano in regola con il SuperBonus 110. Questo implica che ci sia un meccanismo di accreditamento in cui il cedente

⁶ I requisiti di sistema descrivono invece le caratteristiche che deve possedere il sistema. Essi si dividono in: requisiti funzionali, requisiti non funzionali e requisiti di dominio. I requisiti funzionali sono le funzionalità che deve possedere il sistema, i requisiti non funzionali non sono delle funzionalità, bensì dei vincoli imposti sul suo sviluppo, ed infine i requisiti di dominio tengono conto del dominio operativo.

presenta tutti i documenti che attestano la sua eleggibilità presso un determinato gruppo acquirente (in quanto il gruppo ha la possibilità di richiedere documenti specifici aggiuntivi per l'accredimento) e che nel processo di l'acquisto del credito sia presente una fase in cui l'acquirente valida la proposta di cessione sulla base della documentazione fornita dal cedente.

- L'applicazione deve permettere di gestire due tipologie di accordo tra cedente e acquirente di una proposta di cessione:
 - **Preistruttoria:** cedente e acquirente si accordano sulla cessione del credito di un progetto che è stato verificato essere in regola con le norme del SuperBonus 110 ma che non è ancora stato realizzato. In questo caso il cedente si impegna a cedere il credito dell'intero progetto una volta che è stato realizzato e l'acquirente si impegna ad acquistarlo.
 - **Istruttoria:** cedente e acquirente si accordano sulla cessione del credito relativo a un determinato S.A.L. (stato avanzamento dei lavori) a cui è arrivato un progetto in regola con il SuperBonus 110. In questo caso l'acquirente al termine del processo d'acquisto paga il cedente e riceve il credito generato.

- **Requisiti non funzionali:**

- L'applicazione deve essere integrata nella preesistente piattaforma SuperBonus Manager di cui andrà a costituire un servizio specifico per la cessione del credito (Cessione Credito DOC)
- L'integrazione con la piattaforma vincola all'uso dell' architettura client-server rientrando nella tipologia di web app accessibile tramite browser e nel riutilizzo del database della piattaforma preesistenti per la memorizzazione completa delle informazioni relative alle cessione e agli accreditamenti.
- Il servizio deve sfruttare e riutilizzare il più possibile i componenti base già presenti (es. API per caricamento dei documenti, servizio di autenticazione, strutture del database)

- **Requisiti di Dominio:**

- I dati essenziali delle proposte di cessione e delle richieste di accredimento devono essere memorizzati in maniera decentralizzata, sicura e tracciabile utilizzando una rete blockchain con algoritmo di consenso che garantisca almeno la tolleranza ai guasti (crash fault tolerant).
- Ogni utente deve disporre di un portachiavi costituito da chiave pubblica e chiave privata protetto da password che gli consente di eseguire transazioni sulla blockchain utilizzando l'applicazione come client.
- I documenti e i contratti firmati digitalmente devono essere inseriti mediante il caricamento di file con estensione "p7m" che fa riferimento a un file firmato digitalmente in modalità CADES.

Analisi dei requisiti

Story Card (Storie Utente)

I requisiti iniziali sono stati dettagliati e ampliati durante le riunioni settimanali durante lo sprint meeting attraverso la definizione di storie utente che sono state poi state sintetizzate in story card. Di seguito vengono riportate alcune delle **story card** più significative.

| Storia Utente | | |
|--|-------------------------|--|
| Release: - | Data: 11/06/2021 | Utente: Cedente, Acquirente, Validatore, Revisore Fiscale |
| Nome storia: Panoramica Cessione Credito | | Storia ID: 1 |
| Priorità: - | | Accettato: - |
| Stima durata sviluppo: - | | |
| Descrizione: Dopo aver acceduto al servizio di cessione del credito dell'applicazione, ogni utente deve poter visualizzare le varie sezioni in accordo al proprio ruolo: <ul style="list-style-type: none">- Il cedente può visualizzare la sezione di gestione delle proposte di cessione in cui trova tutte le proposte create e la sezione relativa alle richieste di accreditamento in cui trova le proprie richieste ai vari gruppi acquirente.- L'acquirente può visualizzare la sezione di gestione delle proposte di cessione in cui trova tutte le proposte che ha "acquistato" o che gli sono state direttamente proposte al proprio gruppo acquirente e la sezione relativa alle richieste di accreditamento in cui trova le richieste di accreditamento fatte dai cedenti al proprio gruppo acquirente.- Il revisore fiscale può visualizzare la sezione di gestione delle proposte di cessione in cui trova tutte le proposte del proprio gruppo acquirente.- Il validatore può visualizzare la sezione relativa alle richieste di accreditamento fatte dai cedenti per la borsa del credito. | | |
| Osservazioni: <ul style="list-style-type: none">- Il cedente può accedere alla sezione di gestione delle cessioni solo dopo essere stato accreditato da almeno un gruppo acquirente o per la borsa del credito.- Ogni utente può accedere al servizio di cessione del credito solo dopo aver registrato con successo il proprio account blockchain. | | |

| Storia Utente | | |
|--|-------------------------|--|
| Release: - | Data: 23/07/2021 | Utente: Cedente, Acquirente, Validatore, Revisore Fiscale |
| Nome storia: Registrazione Account Blockchain | | Storia ID: 1 |
| Priorità: - | | Accettato: - |
| Stima durata sviluppo: - | | |
| <p>Descrizione: Ogni utente che intende utilizzare il servizio di cessione del credito deve registrare/attivare il proprio account Blockchain. Per farlo accede alla sezione dei dati personali dove può visualizzare lo stato di attivazione dell'account blockchain e registrare l'account tramite un apposito pulsante. Il processo di registrazione deve richiedere l'inserimento della password e di una serie di risposte a domande di sicurezza per il recupero della stessa. Una volta che ha inserito tutte e le informazioni l'utente conferma la registrazione e attende che l'account sia registrato.</p> | | |
| <p>Osservazioni:</p> <ul style="list-style-type: none"> - L'inserimento della password deve essere richiesto due volte per evitare errori di battitura. - La password deve essere sicura. - Lo stato di registrazione passa da "Non Registrato" a "Registrato" se il processo è andato a buon fine. | | |

| Storia Utente | | |
|--|-------------------------|--|
| Release: - | Data: 23/07/2021 | Utente: Cedente, Acquirente, Validatore, Revisore Fiscale |
| Nome storia: Modifica Password Account Blockchain | | Storia ID: 1 |
| Priorità: | | Accettato: - |
| Stima durata sviluppo: - | | |
| <p>Descrizione: Ogni utente che ha registrato un account blockchain deve poter modificare la password del proprio account Blockchain. Per farlo accede alla sezione dei dati personali e modifica la password tramite l'apposito pulsante. Il processo di modifica deve richiedere sia l'inserimento della vecchia password che della nuova password. La modifica deve avvenire solo se la vecchia password è corretta.</p> | | |
| <p>Osservazioni:</p> <ul style="list-style-type: none"> - L'inserimento della nuova password deve essere richiesto due volte per evitare errori di battitura. - La nuova password deve essere sicura. | | |

| Storia Utente | | |
|---|-------------------------|--|
| Release: - | Data: 23/07/2021 | Utente: Cedente, Acquirente, Validatore, Revisore Fiscale |
| Nome storia: Reset Password Account Blockchain | | Storia ID: 1 |
| Priorità: | | Accettato: - |
| Stima durata sviluppo: - | | |
| <p>Descrizione: Ogni utente che ha registrato un account blockchain deve poter resettare la password dell'account in caso di smarrimento. Per farlo accede alla sezione dei dati personali e resetta la password tramite l'apposito pulsante. Il processo di reset della password deve richiedere sia l'inserimento delle risposte alle domande di sicurezza date in fase di registrazione sia la nuova password. La modifica deve avvenire solo se le risposte alle domande di sicurezza sono corrette.</p> | | |
| <p>Osservazioni:</p> <ul style="list-style-type: none"> - L'inserimento della nuova password deve essere richiesto due volte per evitare errori di battitura. - La nuova password deve essere sicura. | | |

| Storia Utente | | |
|---|-------------------------|------------------------|
| Release: - | Data: 11/06/2021 | Utente: Cedente |
| Nome storia: Nuova Proposta di Cessione | | Storia ID: 2 |
| Priorità: - | | Accettato: - |
| Stima durata sviluppo: - | | |
| <p>Descrizione: Il cedente deve poter creare una nuova proposta di cessione. Per farlo accede alla sezione di gestione delle cessioni e inizia la creazione attraverso l'apposito pulsante. Il processo di creazione deve permettere la scelta della tipologia di cessione tra istruttoria e preistruttoria, il gruppo acquirente al quale inviare la proposta o altrimenti la possibilità di presentarla in borsa del credito. A seconda della tipologia selezionata il cedente dovrà poi scegliere una tra le sue pratiche da inserire nella proposta. Una volta inserite tutte le informazioni necessarie, il cedente può confermare la creazione della proposta. La nuova proposta verrà aggiunta all'elenco della sezione di gestione delle cessioni.</p> | | |
| <p>Osservazioni:</p> <ul style="list-style-type: none"> - Nel caso si scelga la tipologia preistruttoria si potranno associare le opportunità, nel caso si scelga invece la tipologia istruttoria si potranno associare le combinazioni sal-commessa. - Lo stato iniziale della nuova proposta è "Preparazione". - Questa operazione comporta una modifica allo stato della blockchain, pertanto l'utente dovrà inserire la password corretta del proprio account Blockchain al momento della conferma dell'operazione. - Affinché un sal di una commessa sia disponibile per una cessione è necessario che sia stato completato. Analogamente un'opportunità è cedibile quando è completata. | | |

| Storia Utente | | |
|---|-------------------------|------------------------|
| Release: - | Data: 11/06/2021 | Utente: Cedente |
| Nome storia: Invio Proposta di Cessione | | Storia ID: 3 |
| Priorità: - | | Accettato: - |
| Stima durata sviluppo: - | | |
| <p>Descrizione: Il cedente deve poter inviare una nuova proposta di cessione al gruppo acquirente dopo aver inserito tutti i documenti richiesti. Per farlo accede alla sezione di gestione delle cessioni, seleziona la proposta e la visualizza tramite l'apposito pulsante. Quindi inserisce tutti i documenti richiesti dal gruppo acquirente presenti nella sezione documentazione: sia quelli aggiornati relativi alla pratica (generici) sia quelli relativi alla proposta vera e propria. Quando tutti i documenti sono stati inseriti il cedente ha la possibilità di inviare tramite la proposta il pulsante "Proponi". Gli acquirenti del gruppo acquirente della proposta devono ricevere una notifica che informa della nuova proposta disponibile.</p> | | |
| <p>Osservazioni:</p> <ul style="list-style-type: none"> - Lo stato della proposta passa da "Preparazione" a "Proposta". - Questa operazione comporta una modifica allo stato della blockchain, pertanto l'utente dovrà inserire la password corretta del proprio account Blockchain al momento della conferma dell'operazione. - Gli acquirenti possono richiedere che alcuni dei documenti associati alla proposta o alla pratica siano firmati digitalmente. | | |

| Storia Utente | | |
|---|-------------------------|---------------------------|
| Release: - | Data: 18/06/2021 | Utente: Acquirente |
| Nome storia: Presa in Carico della Proposta | | Storia ID: 4 |
| Priorità: - | | Accettato: - |
| Stima durata sviluppo: - | | |
| <p>Descrizione: L'acquirente deve poter segnalare al cedente di aver preso in carico la proposta di cessione che è stata inviata al suo gruppo acquirente. Per farlo accede alla sezione di gestione delle cessioni, seleziona la proposta e la visualizza tramite l'apposito pulsante. L'acquirente ha la possibilità di visualizzare le informazioni relative alla proposta e alla pratica nella sezione "riepilogo" per decidere se prenderla in carico o meno. Se decide di farlo conferma l'operazione attraverso il pulsante "Prendi in Carico". Il cedente della proposta deve ricevere una notifica che lo informa dell'avvenuta presa in carico</p> | | |
| <p>Osservazioni:</p> <ul style="list-style-type: none"> - Lo stato della proposta passa da "Proposta" a "Presa in Carico". - Questa operazione comporta una modifica allo stato della blockchain, pertanto l'utente dovrà inserire la password corretta del proprio account Blockchain al momento della conferma dell'operazione. - Le proposte che vengono "prese in carico" sono quelle proposte direttamente al gruppo acquirente e che non vengono quindi acquistate tramite il meccanismo della borsa del credito | | |

| Storia Utente | | |
|--|-------------------------|---|
| Release: - | Data: 18/06/2021 | Utente: Acquirente, Revisore Fiscale |
| Nome storia: Validazione Proposta | | Storia ID: 5 |
| Priorità: - | | Accettato: - |
| Stima durata sviluppo: - | | |
| <p>Descrizione: L'acquirente/revisore fiscale deve poter validare una proposta di cessione che il proprio gruppo acquirente ha preso in carico. Per farlo accede alla sezione di gestione delle cessioni, seleziona la proposta e la visualizza tramite l'apposito pulsante. L'acquirente/revisore fiscale ha la possibilità di visualizzare i documenti della proposta nella sezione Documenti (inseriti dal cedente precedentemente) e stabilire se la proposta è valida o meno. L'acquirente/revisore fiscale valida o invalida la proposta tramite i pulsanti "Valida" e "Invalida". In caso di invalidazione, al momento della conferma, l'acquirente deve anche indicare le motivazioni. Una proposta invalidata può essere rivalidata dall'acquirente/revisore fiscale dopo che cedente ha corretto gli errori, nel frattempo la proposta rimane nello stato "Invalidata". Il cedente della proposta deve ricevere una notifica che lo informa dell'esito della valutazione (e le motivazioni in caso di invalidazione).</p> | | |
| <p>Osservazioni:</p> <ul style="list-style-type: none"> - Lo stato della proposta passa da "Presa in Carico/Invalidata" a "Validata/Invalidata". - Questa operazione comporta una modifica allo stato della blockchain, pertanto l'utente dovrà inserire la password corretta del proprio account Blockchain al momento della conferma dell'operazione. - L'utente ha sia la possibilità di visualizzare nel browser i documenti della proposta (purché il formato sia supportato dal browser) sia di scaricarli sul suo dispositivo. | | |

| Storia Utente | | |
|--|-------------------------|---------------------------|
| Release: - | Data: 02/07/2021 | Utente: Acquirente |
| Nome storia: Caricamento Contratto da Firmare | | Storia ID: 6 |
| Priorità: - | | Accettato: - |
| Stima durata sviluppo: - | | |
| <p>Descrizione: L'acquirente deve poter caricare il contratto da far firmare al cedente di una proposta di cessione che è stata validata correttamente da un acquirente o un revisore fiscale del suo stesso gruppo acquirente. Per farlo accede alla sezione di gestione delle cessioni, seleziona la proposta e la visualizza tramite l'apposito pulsante. L'acquirente ha la possibilità di caricare il documento del contratto nella sezione Bonifici e Contratti, sottosezione Contratti. Nel momento in cui carica il file del contratto all'acquirente viene chiesta conferma dell'operazione. Il cedente deve ricevere una notifica che lo informa che c'è un contratto da firmare per quella proposta di cessione.</p> | | |
| <p>Osservazioni:</p> <ul style="list-style-type: none"> - Lo stato della proposta passa da "Validata/Acquistata" a "Contratto da Firmare". - Questa operazione comporta una modifica allo stato della blockchain, pertanto l'utente dovrà inserire la password corretta del proprio account Blockchain al momento della conferma dell'operazione. - La tipologia del contratto varia in base alla tipologia di cessione: in caso di Preistruttoria si avrà una "Lettera d'Impegno", in caso di Istruttoria si avrà un "Contratto di Cessione". | | |

| Storia Utente | | |
|--|-------------------------|------------------------|
| Release: - | Data: 02/07/2021 | Utente: Cedente |
| Nome storia: Caricamento Contratto Firmato | | Storia ID: 7 |
| Priorità: - | | Accettato: - |
| Stima durata sviluppo: - | | |
| <p>Descrizione: Il cedente deve poter scaricare il contratto da firmare e ricaricarlo firmato. Per farlo accede alla sezione di gestione delle cessioni, seleziona la proposta e la visualizza tramite l'apposito pulsante. Il cedente ha la possibilità di scaricare il documento del contratto nella sezione Bonifici e Contratti, sottosezione Contratti. Dopo averlo scaricato il file del contratto, firma il contratto e deve poterlo ricaricare nella stessa sottosezione. Nel momento in cui carica il file del contratto firmato al cedente viene chiesta conferma dell'operazione. Gli acquirenti del gruppo acquirente della proposta devono ricevere una notifica che informa che il contratto per quella proposta di cessione è stato firmato.</p> | | |
| <p>Osservazioni:</p> <ul style="list-style-type: none"> - Lo stato della proposta passa da "Contratto da Firmare" a "Contratto Firmato". - Questa operazione comporta una modifica allo stato della blockchain, pertanto l'utente dovrà inserire la password corretta del proprio account Blockchain al momento della conferma dell'operazione. - Il file del contratto di firmato deve essere anche firmato digitalmente. | | |

| Storia Utente | | |
|--|-------------------------|---------------------------|
| Release: - | Data: 09/07/2021 | Utente: Acquirente |
| Nome storia: Liquidazione della Proposta di cessione | | Storia ID: 8 |
| Priorità: - | | Accettato: - |
| Stima durata sviluppo: - | | |
| <p>Descrizione: L'acquirente deve poter caricare i bonifici che confermano il pagamento del credito per una proposta di cessione di tipologia preistruttoria, potendo in questo modo liquidare il contratto. Per farlo accede alla sezione di gestione delle cessioni, seleziona la proposta e la visualizza tramite l'apposito pulsante. Il cedente ha la possibilità di caricare uno o più bonifici nella sezione Bonifici e Contratti, sottosezione Bonifici. Dopo aver caricato tutti i file relativi ai bonifici, può liquidare la proposta di cessione tramite il pulsante "Liquida". Il cedente deve ricevere una notifica che lo informa che la proposta è stata liquidata.</p> | | |
| <p>Osservazioni:</p> <ul style="list-style-type: none"> - Il cambiamento di stato avviene solo in caso di istruttoria: lo stato della proposta passa da "Contratto da Firmare" a "Da Liquidare". - Questa operazione comporta una modifica allo stato della blockchain, pertanto l'utente dovrà inserire la password corretta del proprio account Blockchain al momento della conferma dell'operazione. | | |

| Storia Utente | | |
|--|-------------------------|---------------------------|
| Release: - | Data: 16/07/2021 | Utente: Acquirente |
| Nome storia: Nuova Offerta | | Storia ID: 11 |
| Priorità: - | | Accettato: - |
| Stima durata sviluppo: - | | |
| <p>Descrizione: L'acquirente della borsa acquirente deve poter fare un'offerta ad una proposta di cessione. Per farlo accede alla sezione di gestione delle cessioni, seleziona la proposta e la visualizza tramite l'apposito pulsante. Nella sezione "Offerte" della proposta, l'acquirente può visualizzare tutte le offerte fatte da altri acquirenti e avanzare la propria offerta. Il processo di creazione di una nuova offerta prevede semplicemente di indicare l'ammontare che l'acquirente offre per l'acquisto del credito legato alla proposta. La nuova offerta verrà aggiunta all'elenco della sezione offerte della proposta. Il cedente deve ricevere una notifica che lo informa che della nuova offerta.</p> | | |
| <p>Osservazioni:</p> <ul style="list-style-type: none"> - Lo stato iniziale della nuova offerta è "Da Valutare". - Il meccanismo delle offerte è disponibile solo per le proposte della borsa del credito - Per le proposte in borsa del credito non è prevista una fase di validazione esplicita, pertanto l'acquirente dovrebbe controllare la validità dei documenti prima di fare l'offerta. - Questa operazione comporta una modifica allo stato della blockchain, pertanto l'utente dovrà inserire la password corretta del proprio account Blockchain al momento della conferma dell'operazione. | | |

| Storia Utente | | |
|--|-------------------------|---------------------------|
| Release: - | Data: 16/07/2021 | Utente: Acquirente |
| Nome storia: Aggiornamento Offerta | | Storia ID: 12 |
| Priorità: - | | Accettato: - |
| Stima durata sviluppo: - | | |
| <p>Descrizione: L'acquirente della borsa acquirente deve poter aggiornare una propria offerta fatta ad una proposta di cessione. Per farlo accede alla sezione di gestione delle cessioni, seleziona la proposta e la visualizza tramite l'apposito pulsante. Nella sezione "Offerte" della proposta, l'acquirente può visualizzare tutte le offerte fatte tra cui la propria. L'acquirente può a questo punto aggiornare l'offerta tramite il pulsante "Aggiorna Offerta". Il processo di aggiornamento prevede semplicemente di indicare il nuovo ammontare dell'offerta. Il cedente deve ricevere una notifica che lo informa dell'aggiornamento dell'offerta.</p> | | |
| <p>Osservazioni:</p> <ul style="list-style-type: none"> - Un'offerta può essere aggiornata se il suo stato è "Da Validare o Rifiutata". Se l'offerta era stata rifiutata dopo l'operazione il suo stato passa da "Rifiutata" a "Da Validare", altrimenti lo stato rimane immutato. - Il meccanismo delle offerte è disponibile solo per le proposte della borsa del credito - Questa operazione comporta una modifica allo stato della blockchain, pertanto l'utente dovrà inserire la password corretta del proprio account Blockchain al momento della conferma dell'operazione. | | |

| Storia Utente | | |
|--|-------------------------|---------------------------|
| Release: - | Data: 16/07/2021 | Utente: Acquirente |
| Nome storia: Eliminazione Offerta | | Storia ID: 13 |
| Priorità: - | | Accettato: - |
| Stima durata sviluppo: - | | |
| <p>Descrizione: L'acquirente della borsa acquirente deve poter eliminare una propria offerta fatta ad una proposta di cessione. Per farlo accede alla sezione di gestione delle cessioni, seleziona la proposta e la visualizza tramite l'apposito pulsante. Nella sezione "Offerte" della proposta, l'acquirente può visualizzare tutte le offerte fatte tra cui la propria. L'acquirente può a questo punto eliminare l'offerta tramite il pulsante "Elimina Offerta". Il cedente deve ricevere una notifica che lo informa dell'eliminazione dell'offerta.</p> | | |
| <p>Osservazioni:</p> <ul style="list-style-type: none"> - Il meccanismo delle offerte è disponibile solo per le proposte della borsa del credito - Questa operazione comporta una modifica allo stato della blockchain, pertanto l'utente dovrà inserire la password corretta del proprio account Blockchain al momento della conferma dell'operazione. | | |

| Storia Utente | | |
|---|-------------------------|------------------------|
| Release: - | Data: 16/07/2021 | Utente: Cedente |
| Nome storia: Valutazione Offerta | | Storia ID: 14 |
| Priorità: - | | Accettato: - |
| Stima durata sviluppo: - | | |
| <p>Descrizione: Il cedente deve poter valutare un'offerta fatta ad una propria proposta di cessione. Per farlo accede alla sezione di gestione delle cessioni, seleziona la proposta e la visualizza tramite l'apposito pulsante. Nella sezione "Offerte" della proposta, il cedente può visualizzare tutte le offerte fatte e decidere se accettarne o rifiutarne una in particolare tramite i pulsanti "Accetta" e "Rifiuta" presenti su ogni proposta nell'elenco. In particolare se l'utente accetta una proposta le altre vengono automaticamente rifiutate. Una volta che il cedente accetta un'offerta gli acquirenti non possono più avanzare ulteriori offerte o aggiornare quelle esistenti. Ogni acquirente deve ricevere una notifica che lo informa dell' accettazione/rifiuto della propria offerta.</p> | | |
| <p>Osservazioni:</p> <ul style="list-style-type: none"> - Lo stato dell'offerta valutata passa da "Da Validare" a "Accettata/Rifiutata". - Se un'offerta viene accettata lo stato della proposta passa da "Proposta" ad "Acquistata". Una proposta acquistata è visibile solo dal Cedente e dall'Acquirente che ha vinto "l'asta" (la cui offerta è stata accettata) nelle rispettive sezioni di gestione delle proposte. - Il meccanismo delle offerte è disponibile solo per le proposte della borsa del credito - Questa operazione comporta una modifica allo stato della blockchain, pertanto l'utente dovrà inserire la password corretta del proprio account Blockchain al momento della conferma dell'operazione. | | |

| Storia Utente | | |
|---|-------------------------|------------------------|
| Release: - | Data: 25/06/2021 | Utente: Cedente |
| Nome storia: Nuova Richiesta Accredimento | | Storia ID: 9 |
| Priorità: - | | Accettato: - |
| Stima durata sviluppo: - | | |
| <p>Descrizione: Il cedente deve poter creare una nuova richiesta di accredimento presso un gruppo acquirente o la borsa del credito. Per farlo accede alla sezione accreditamenti e inizia la creazione attraverso l'apposito pulsante. La richiesta di accredimento prevede la selezione del gruppo acquirente presso il quale accreditarsi o in alternativa la borsa del credito e l'inserimento di due categorie di documenti</p> <ul style="list-style-type: none"> - Documenti del cedente. Questi documenti sono condivisi da ogni accredimento del cedente e non devono essere ricaricati per accreditamenti successivi. - Documenti specifici richiesti dall'acquirente. Questi documenti variano in base al gruppo acquirente selezionato e quindi da richiesta a richiesta. La proposta deve essere "salvata" Una volta inserite tutti i documenti necessari il cedente può confermare la creazione della richiesta e inviarla al gruppo acquirente tramite il pulsante "Accredita". La richiesta verrà aggiunta all'elenco della sezione accreditamenti. Gli acquirenti del gruppo acquirente o il selezionato o il validatore (per borsa del credito) devono ricevere una notifica che li informa della nuova richiesta di accredimento. | | |
| <p>Osservazioni:</p> <ul style="list-style-type: none"> - Lo stato iniziale della richiesta di accredimento è "Preparazione" che passa a "Da Validare" nel momenti in cui il cedente invia la richiesta. - Una volta inviata la richiesta sarà visualizzabile da ogni acquirente del gruppo acquirente selezionato (in caso di richiesta presso gruppo acquirente) oppure dall'utente validatore della borsa (nel caso di accredimento presso borsa del credito). - Questa operazione comporta una modifica allo stato della blockchain, pertanto l'utente dovrà inserire la password corretta del proprio account Blockchain al momento della conferma dell'operazione. | | |

| Storia Utente | | |
|---|-------------------------|---------------------------------------|
| Release: - | Data: 25/06/2021 | Utente: Acquirente, Validatore |
| Nome storia: Validazione Richiesta di Accredитamento | | Storia ID: 10 |
| Priorità: - | | Accettato: - |
| Stima durata sviluppo: - | | |
| <p>Descrizione: L'acquirente/validatore devono poter validare una richiesta di accredитamento presso il proprio gruppo acquirente (borsa del credito nel caso del validatore). Per farlo accede alla sezione di gestione delle cessioni, seleziona la proposta e la visualizza tramite l'apposito pulsante. L'acquirente/validatore ha la possibilità di visualizzare i documenti della richiesta (inseriti dal cedente precedentemente) e stabilire se la proposta è valida o meno. L'acquirente/validatore valida o invalida la richiesta di accredитamento tramite i pulsanti "Valida" e "Invalida". In caso di invalidazione, al momento della conferma, l'acquirente/validatore deve anche indicare le motivazioni. Una richiesta invalidata può essere rivalidata dall'acquirente/validatore dopo che cedente ha corretto gli errori, nel frattempo la richiesta rimane nello stato "Non valida". Il cedente che ha fatto la richiesta deve ricevere una notifica che lo informa dell'esito della valutazione (e le motivazioni in caso di invalidazione).</p> | | |
| <p>Osservazioni:</p> <ul style="list-style-type: none"> - Lo stato della richiesta passa da "Da Validare/Non Valida" a "Valida/Non Valida". - Una richiesta valida dà il diritto al cedente di creare una nuova proposta di cessione presso quel gruppo acquirente. - Questa operazione comporta una modifica allo stato della blockchain, pertanto l'utente dovrà inserire la password corretta del proprio account Blockchain al momento della conferma dell'operazione. - L'utente ha sia la possibilità di visualizzare nel browser i documenti della proposta (purché il formato sia supportato dal browser) sia di scaricarli sul suo dispositivo. | | |

Diagramma di Gantt delle storie utente

Il diagramma di Gantt in figura 4.1 mostra come le varie storie utente siano state sviluppate nei diversi sprint settimanali:

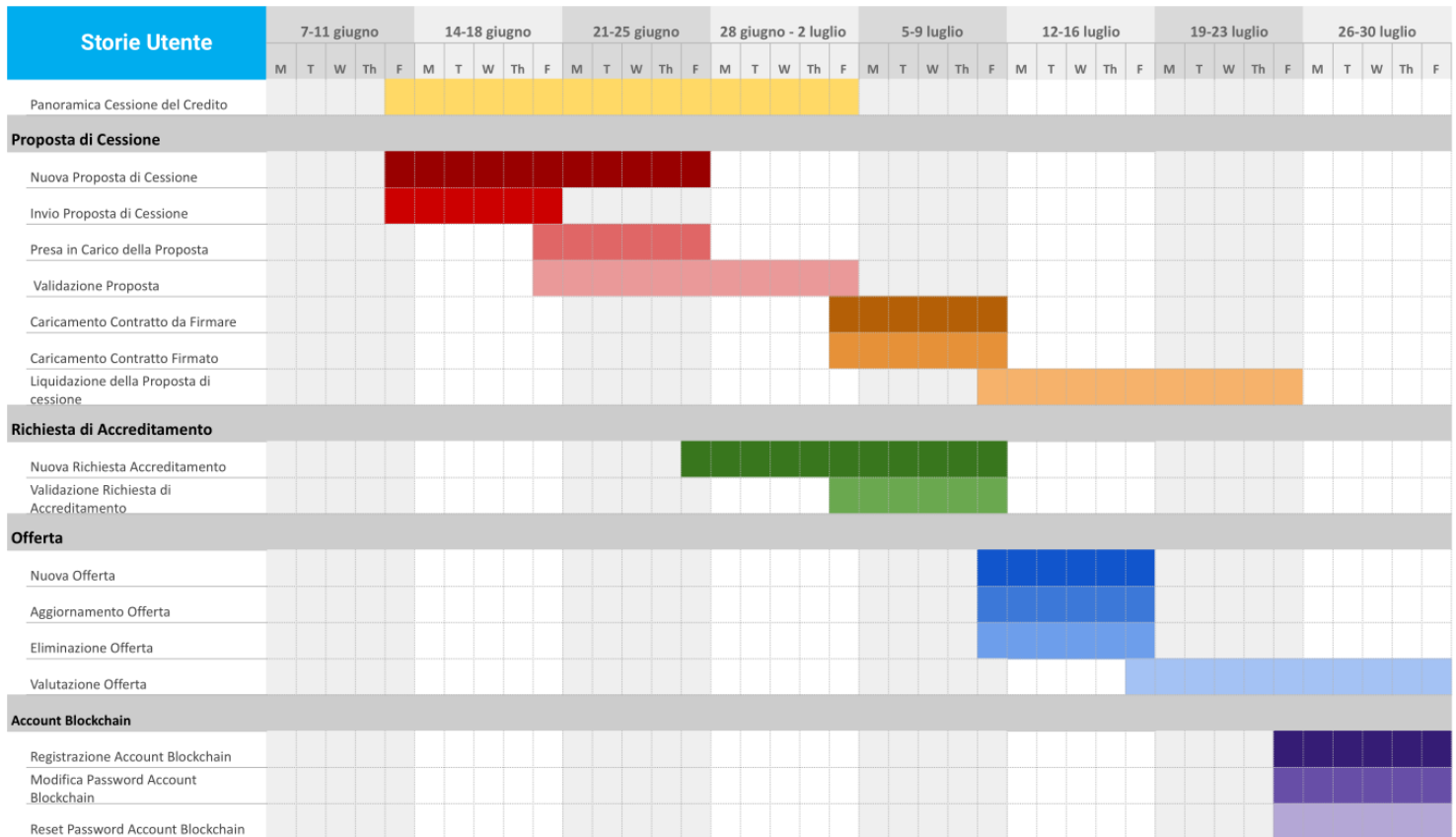


Figura 4.1 - Diagramma di Gantt delle storie utente

Task Card

Le varie storie utente sono state in seguito analizzate e suddivise in task da eseguire una volta assegnate ai vari membri dello Scrum Team. Ogni task è stato dunque riportato su una task card inserita nella kaban per monitorare il suo completamento. A titolo di esempio, di seguito sono mostrate alcune di queste card.

| | |
|---|---|
| Story ID: 1-n | Task ID: 2 |
| | Software Engineer: Mattia Luzi, Giacomo Giuliani |
| Descrizione: Valutare il framework blockchain da utilizzare per la memorizzazione dei dati rilevanti dell'applicazione in relazione ai requisiti forniti | |

| | |
|--|---------------------------------------|
| Story ID: 1-n | Task ID: 25 |
| | Software Engineer: Mattia Luzi |
| Descrizione: Creare una classe helper che permetta al back end di interfacciarsi con i nodi della blockchain nelle operazioni di invio delle transazioni e di lettura dei dati dallo stato. La classe deve anche fornire, in maniera generica e riutilizzabile(per tutte le transaction family), le funzionalità di creazione e firma delle transazioni, creazione e firma dei batch, gestione del portachiavi degli utenti (serializzazione e cifratura delle chiavi). | |

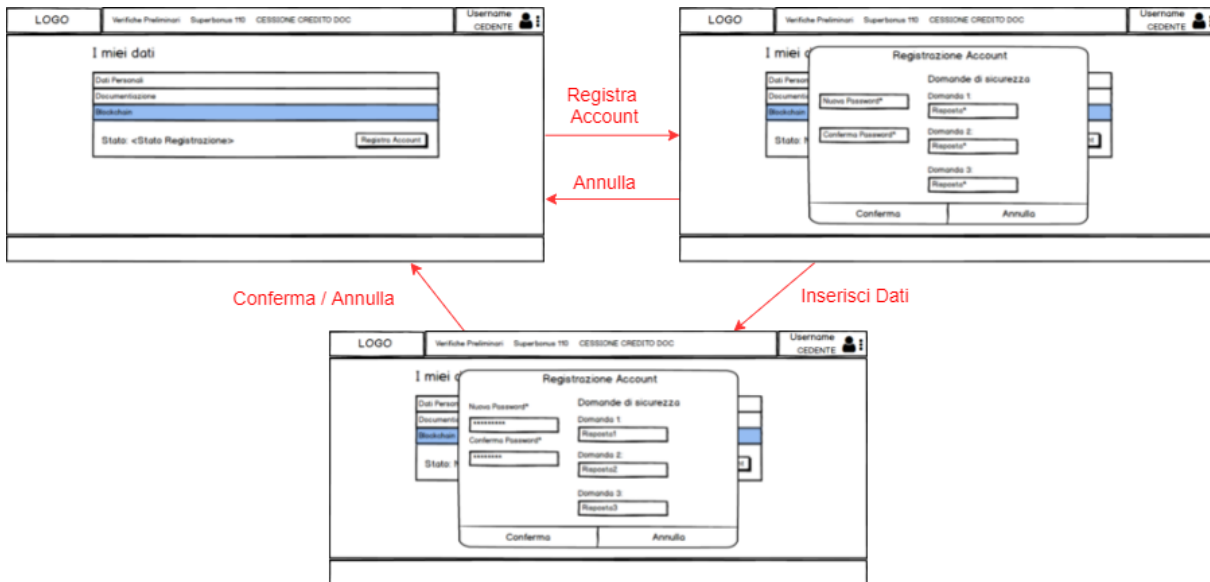
| | |
|--|---|
| Story ID: 1-n | Task ID: 25 |
| | Software Engineer: Mattia Luzi, Giacomo Giuliani |
| Descrizione: Creare il controller Yii che funga da da API REST per il nuovo servizio di cessione del credito. | |

| | |
|--|--|
| Story ID: 4-14 | Task ID: 25 |
| | Software Engineer: Mattia Luzi, |
| Descrizione: Realizzare la sezione dello schema ER relativo alla Proposta di Cessione tenendo conto delle entità del database preesistenti. | |

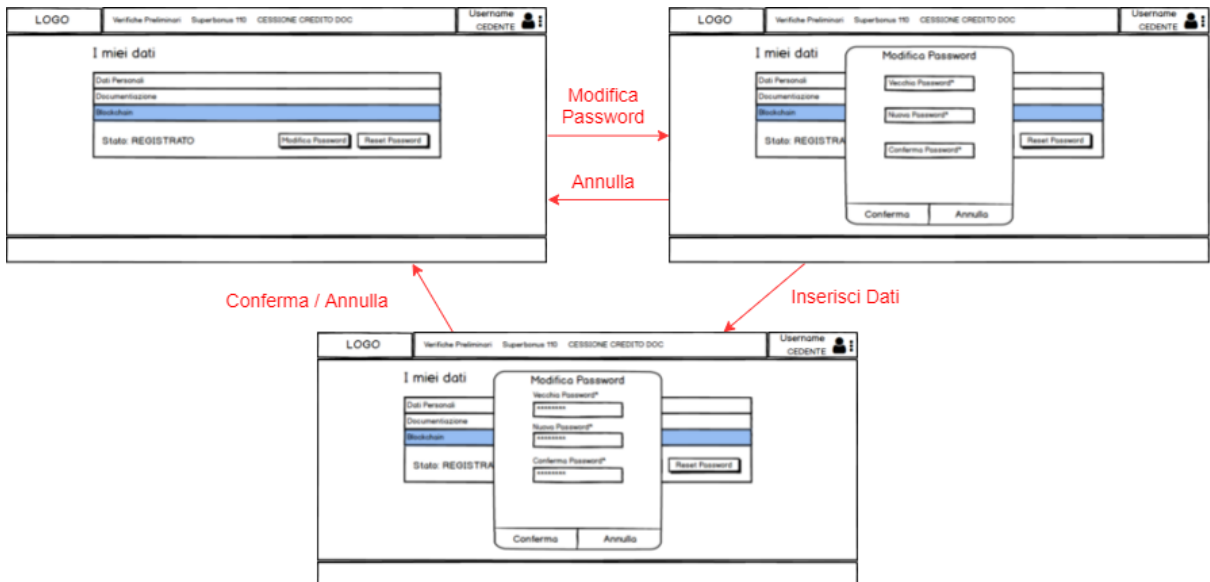
Mockup (Modelli di Navigazione)

L'analisi delle storie utente ha portato alla creazione di mockup interattivi realizzati con lo strumento **Balsamiq Mockup**, i quali sono stati discussi e perfezionati con il product owner e gli esperti di dominio dello Scrum Team. Per documentare questo processo di seguito vengono riportati i **modelli di navigazione** di alcune storie utente.

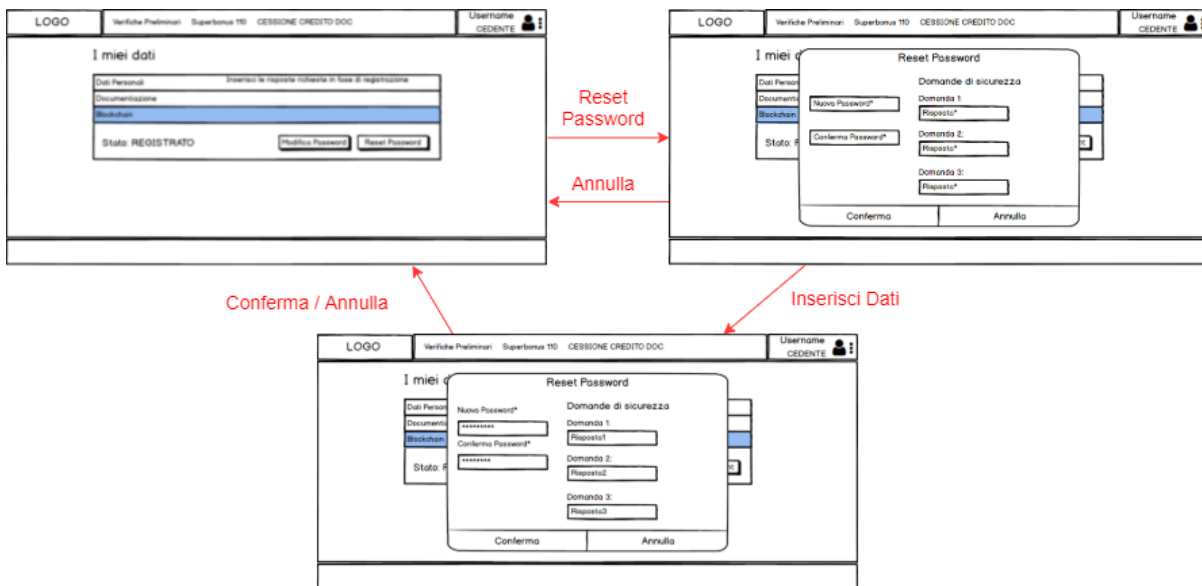
Registrazione Account Blockchain - Cedente, Acquirente, Validatore, Revisore Fiscale



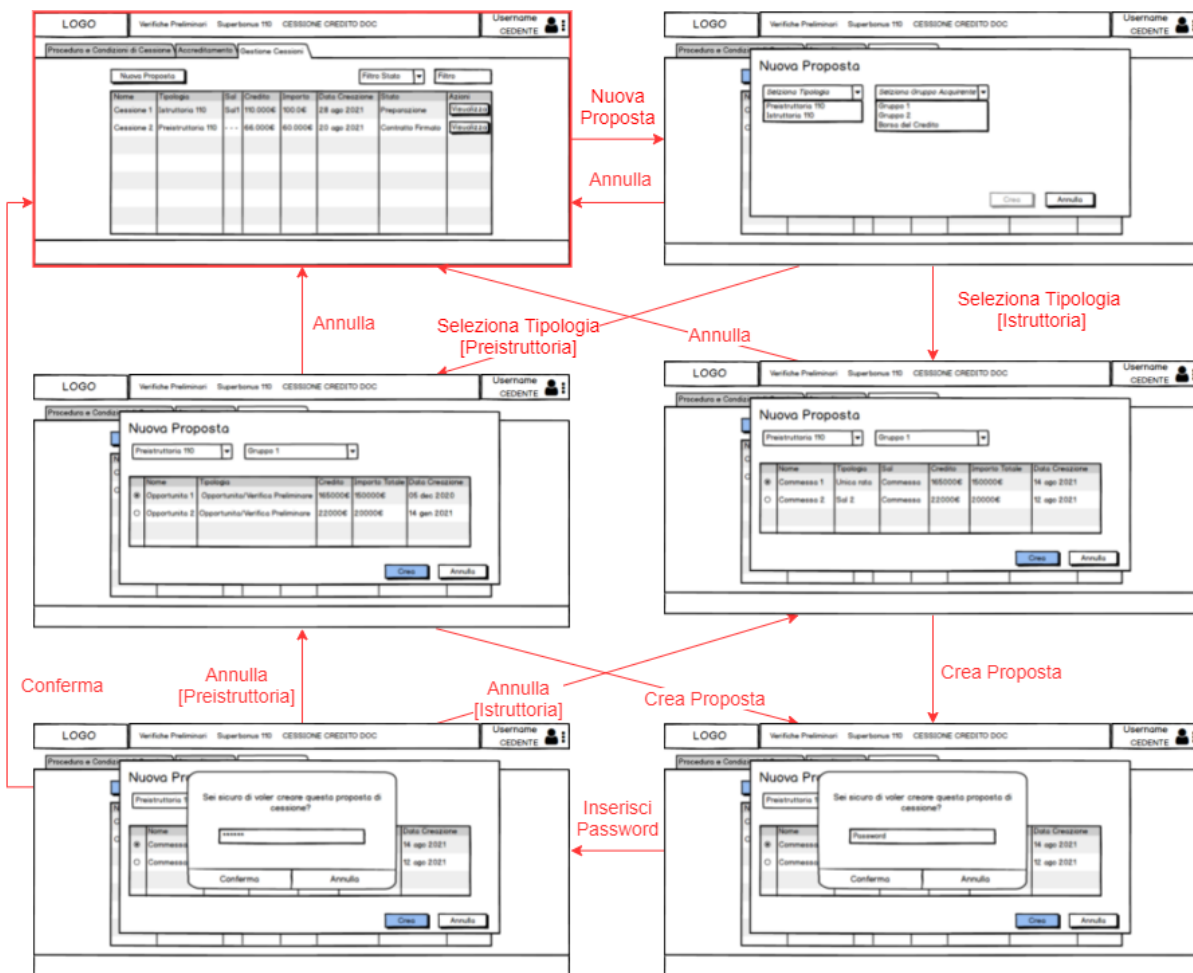
Modifica Password Account Blockchain - Cedente, Acquirente, Validatore, Revisore Fiscale



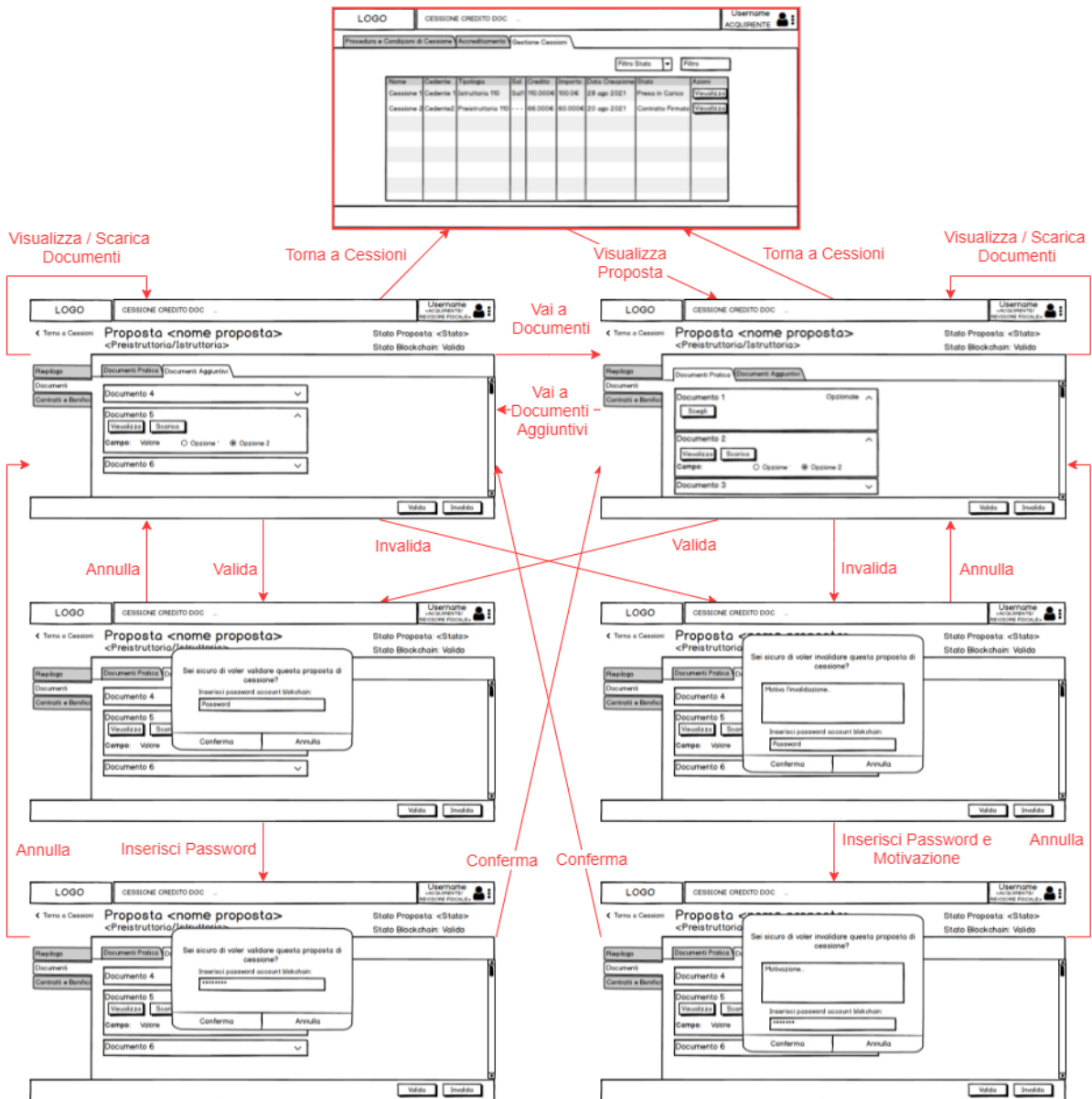
Reset Password Account Blockchain - Cedente, Acquirente, Validatore, Revisore Fiscale



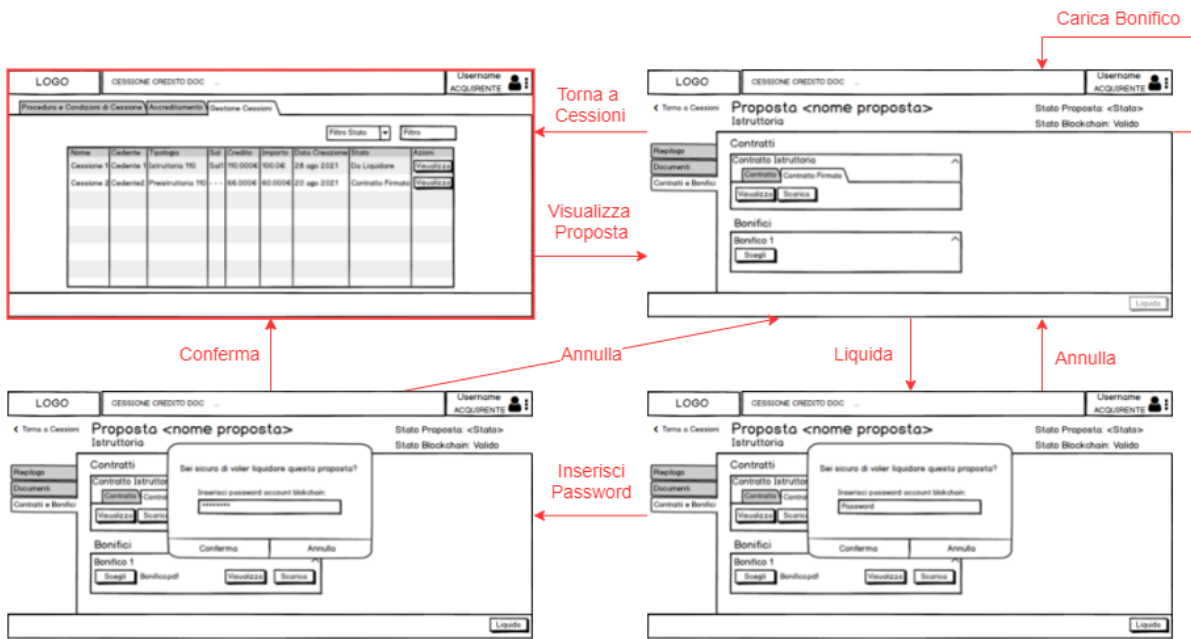
Nuova Proposta di Cessione - Cedente



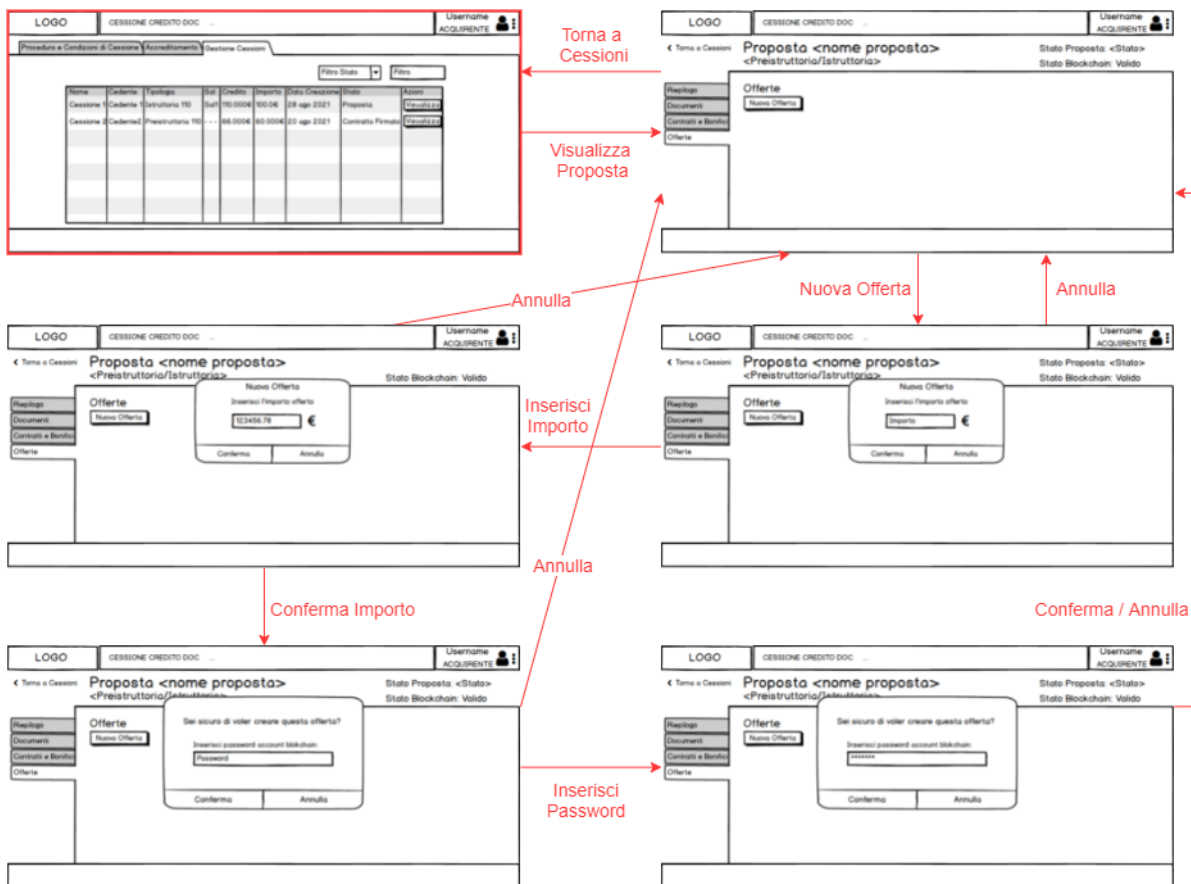
Validazione Proposta - Acquirente, Revisore Fiscale



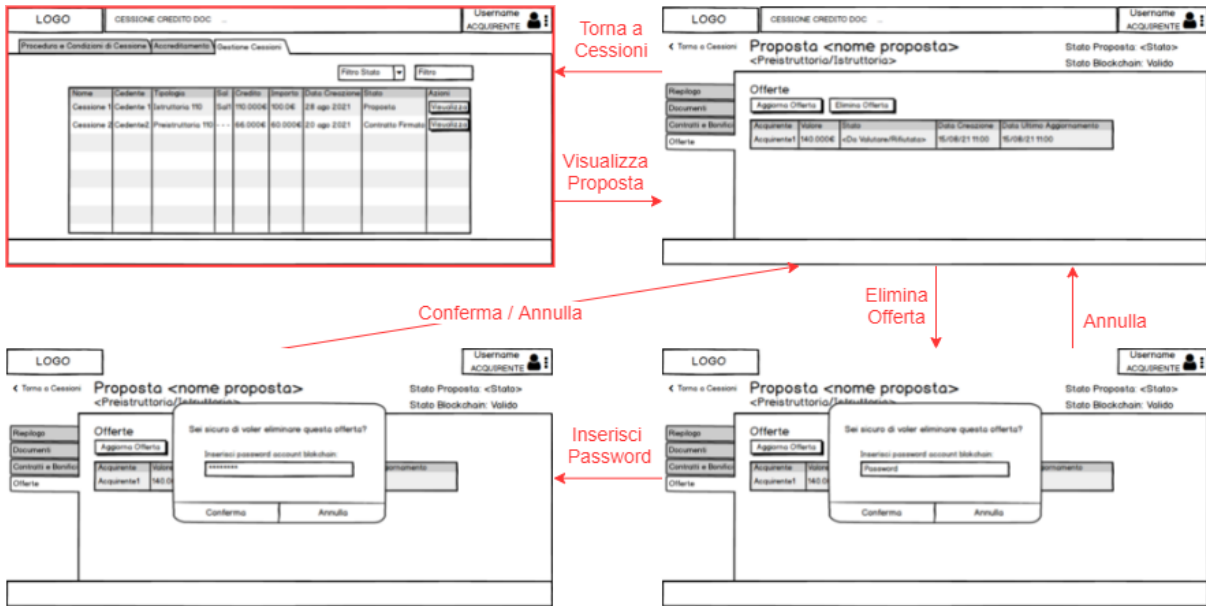
Liquidazione Proposta - Acquirente



Nuova Offerta - Acquirente



Eliminazione Offerta - Acquirente



Nuova Richiesta di Accredimento - Cedente

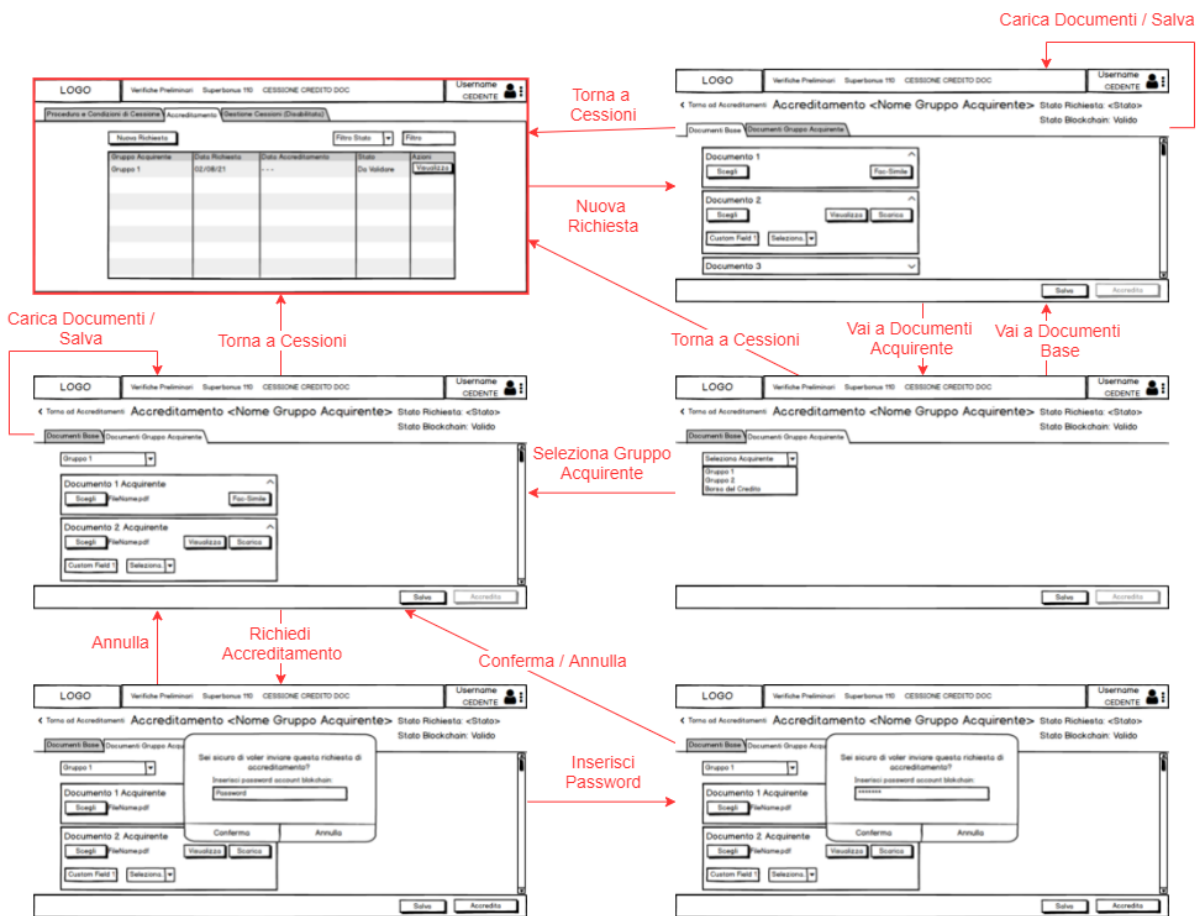


Diagramma di Stato

Nell'applicazione sono tre le entità che subiscono delle significative evoluzioni del proprio stato: la proposta di cessione, la richiesta di accreditamento e l'offerta. Per questo motivo è stato utile

analizzarle dal punto di vista dei cambiamenti di stato che subiscono. La figure 4.1 e 4.2 riportano i **diagrammi di stato** relativi alle singole entità separatamente. Riguardo alla proposta di cessione è possibile notare come l'evoluzione dello stato differisca a seconda della tipologia di cessione (preistruttoria o istruttoria) e della modalità acquisto (borsa del credito o proposta diretta a gruppo acquirente).

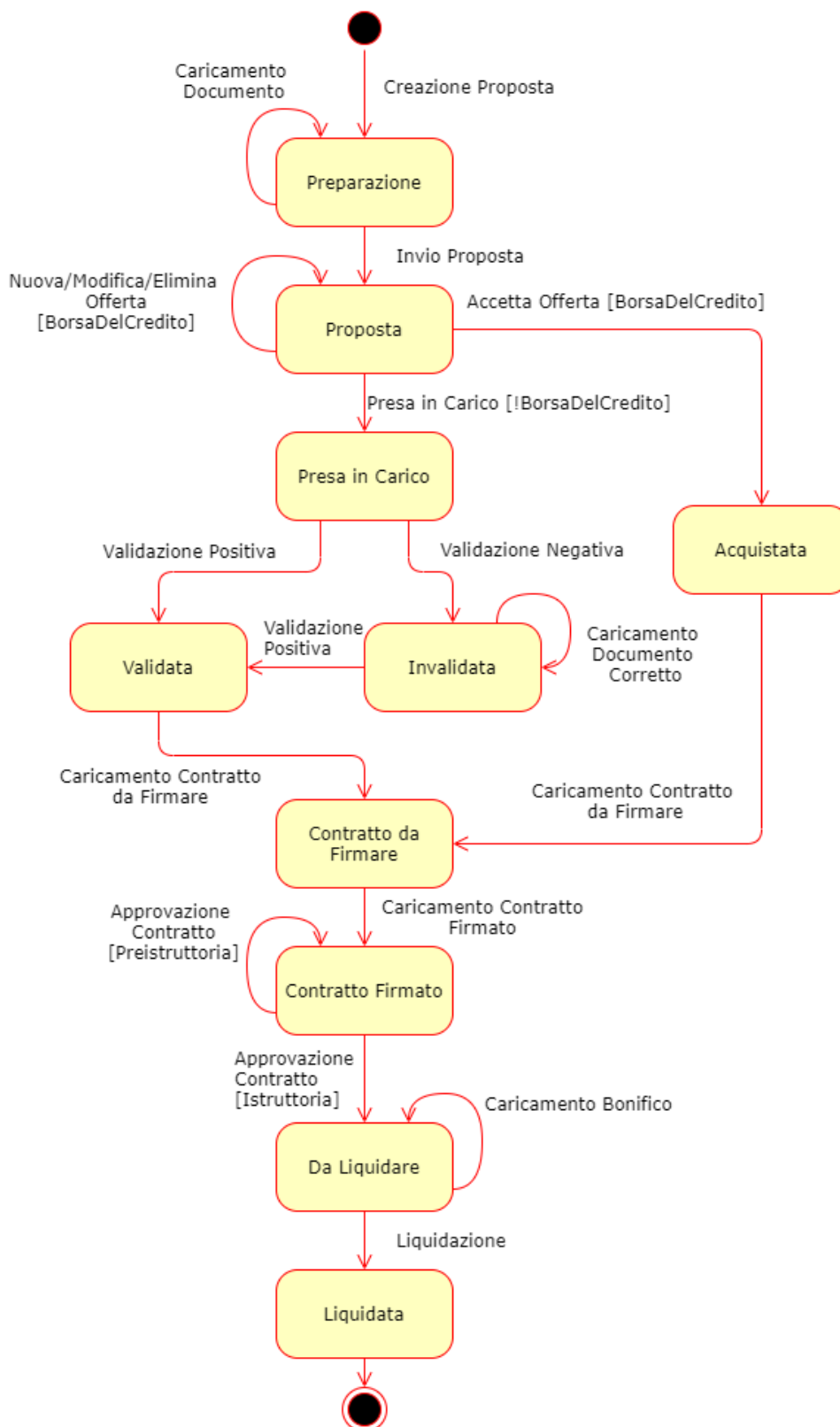


Figura 4.2 - Diagramma di Stato di una Proposta di cessione

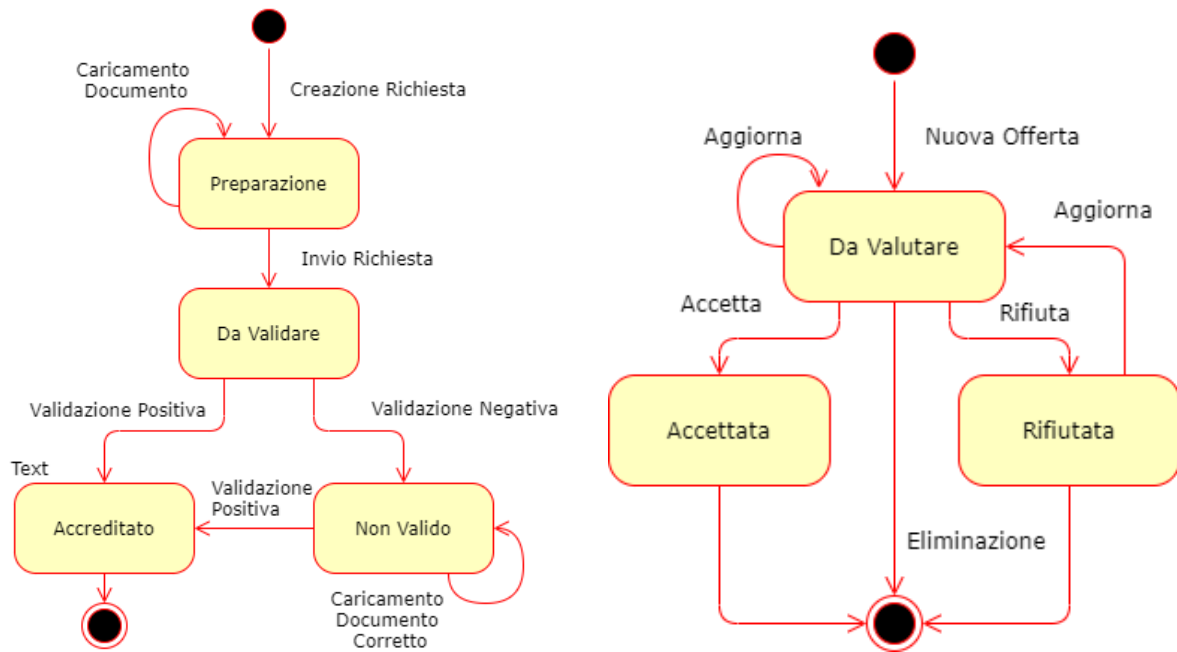


Figura 4.3 - Diagramma di Stato di una Richiesta di accreditamento (sinistra) e di una Offerta (destra)

Progettazione

Diagramma delle classi

Dopo aver analizzato le storie utente e approvato i mockup si è proceduto alla definizione delle classi che sono riportate nel diagramma in figura 4.3.

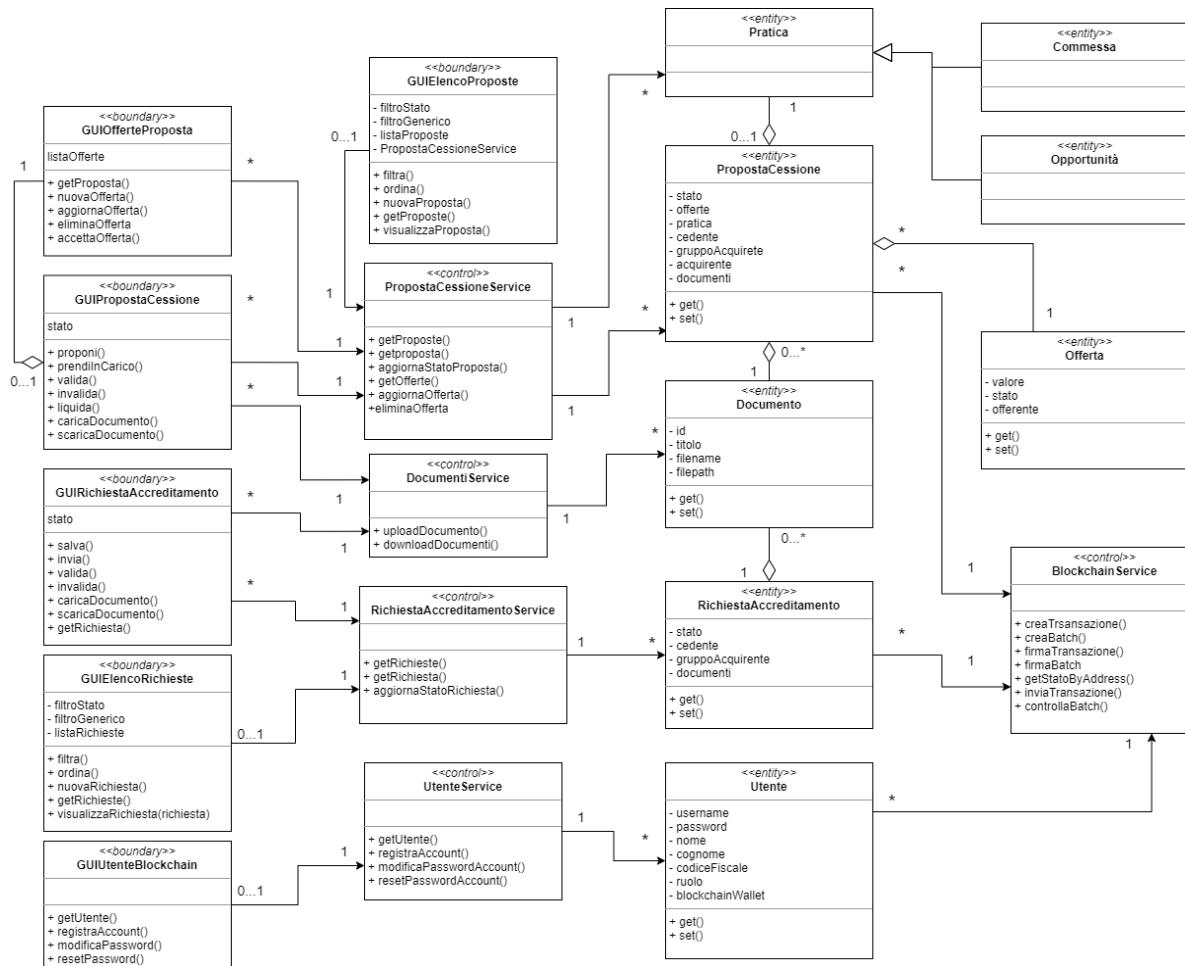


Figura 4.4 - Diagramma delle Classi

Progettazione del Database

Un'importante fase della progettazione è stata quella della realizzazione della struttura del database. Riguardo alla **progettazione del database** bisogna fare tre considerazioni importanti che hanno condizionato questa fase:

- Riguardo alla memorizzazione dei dati ci si è trovati di fronte al problema di stabilire quali informazioni dovevano essere memorizzate nel database e quali nella blockchain; se mantenere tutti i dati sul db e tenere traccia delle modifiche come transazioni nella blockchain o se memorizzare tutte le informazioni sulla blockchain. Per motivi di sicurezza ed efficienza si è optato per un approccio ibrido in cui il database conserva l'informazione per intero e la blockchain lo affianca memorizzando e tenendo traccia della parte essenziale delle informazioni delle due entità più importanti: proposta di cessione e richiesta di accreditamento. I motivi di questa scelta sono approfonditi nel capitolo relativo alla sicurezza.

- La progettazione del database non è avvenuta partendo da zero ma ha richiesto, come da requisito, il riutilizzo del database relazionale della piattaforma a cui sono state aggiunte ulteriori strutture data per la memorizzazioni dei dati del nuovo servizio costituito da questa applicazione. Nei diagrammi che seguono vengono riportate solo queste nuove strutture e, sinteticamente, alcune di quelle preesistenti da cui queste dipendono.
- Utilizzando una metodologia agile, la struttura finale è stata ottenuta attraverso modifiche incrementali per tenere conto dell'evoluzione delle storie utente e delle nuove e più specifiche esigenze venute fuori nel corso dello sviluppo. I diagrammi che seguono riportano solamente la struttura finale.

La progettazione è iniziata realizzando il **diagramma ER (entity-relationship)** che viene mostrato in figura 4.4. Le entità Opportunità, Sal, Commessa, Utente e Modello Documento fanno riferimento a strutture già esistenti nel database della piattaforma e pertanto di queste non sono state elencati tutti gli attributi. Come si può vedere, inoltre, le entità più importanti sono Proposta Cessione, Richiesta Accreditamento e Gruppo Acquirente che presentano tra tutte le altre il maggior numero di associazioni. In particolare la quadrupla relazione tra Proposta Cessione, Opportunità, Sal e Commessa è dovuta al fatto che una proposta di cessione può essere creata a partire da un'opportunità o alternativamente da una combinazione di sal-commessa. La centralità dell'entità Gruppo Acquirente è dovuta invece al fatto che, ad esclusione del meccanismo di borsa del credito, le interazioni tra un acquirente e le altre entità sono quasi sempre indirette, passando dal gruppo acquirente di cui fa parte.

Dal diagramma ER è stato poi ricavato il **diagramma relazionale** in figura 4.5.

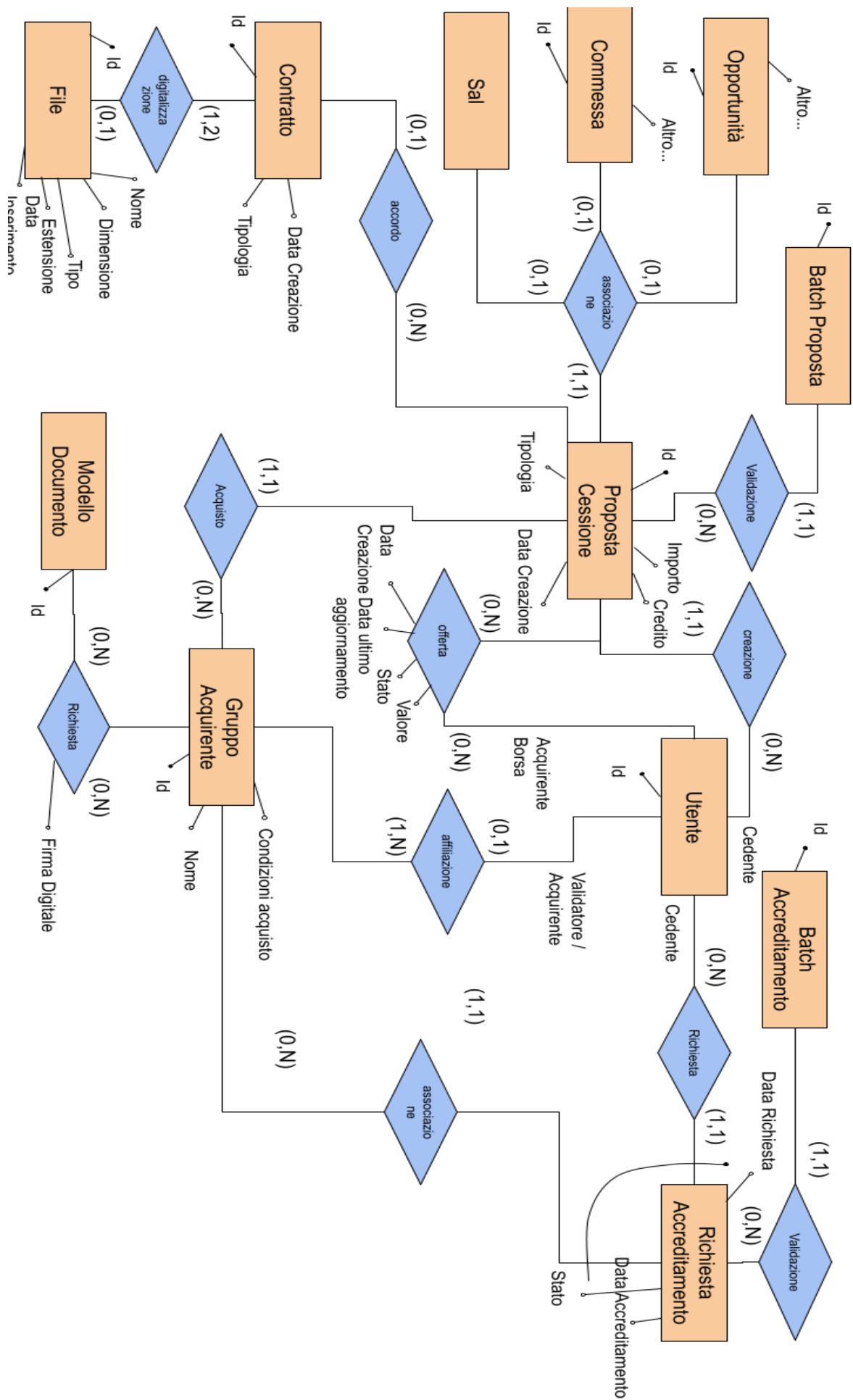


Figura 4.5 - Diagramma ER

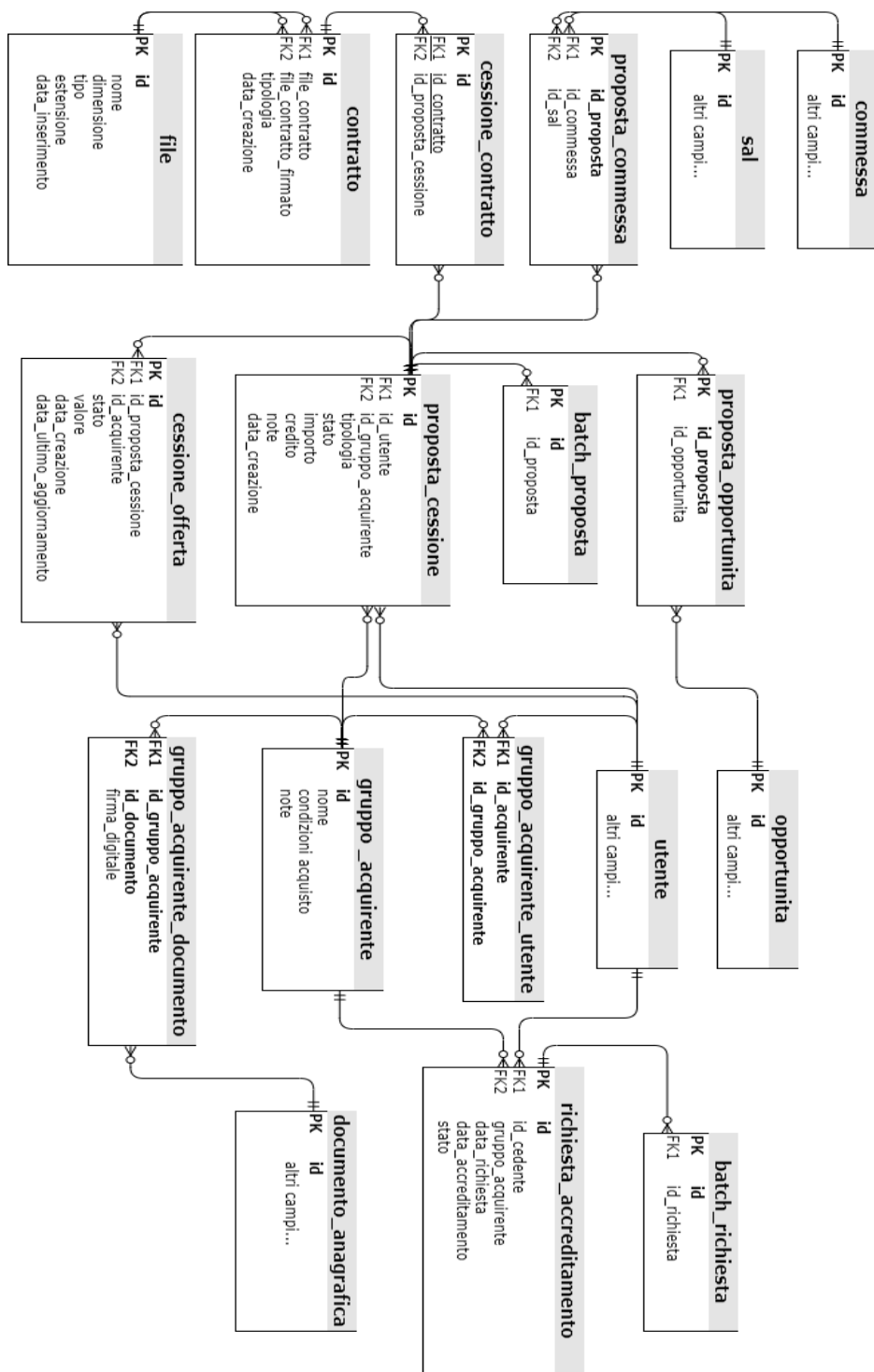


Figura 4.6 - Diagramma Relazionale

Realizzazione

Applicazione Client e Database

Come già accennato lo sviluppo dell'applicazione client è consistito in realtà in una integrazione di un nuovo servizio (cessione del credito) all'interno di una **piattaforma web preesistente** per la gestione delle pratiche relative al credito d'imposta. Questo requisito ha influito sulla scelta delle tecnologie utilizzate nell'applicazione client e nel suo sviluppo. La piattaforma in questione utilizza il noto framework per lo sviluppo di Single Page Web Application **Angular 10** per la parte front end dell'applicazione e il framework PHP per lo sviluppo di applicazioni web basate sul pattern MVC **Yii**

2.0 per il back end. In figura 4.6 è mostrato il **diagramma dell'architettura** della piattaforma in cui sono evidenziati i componenti integrati per il nuovo servizio di cessione del credito.

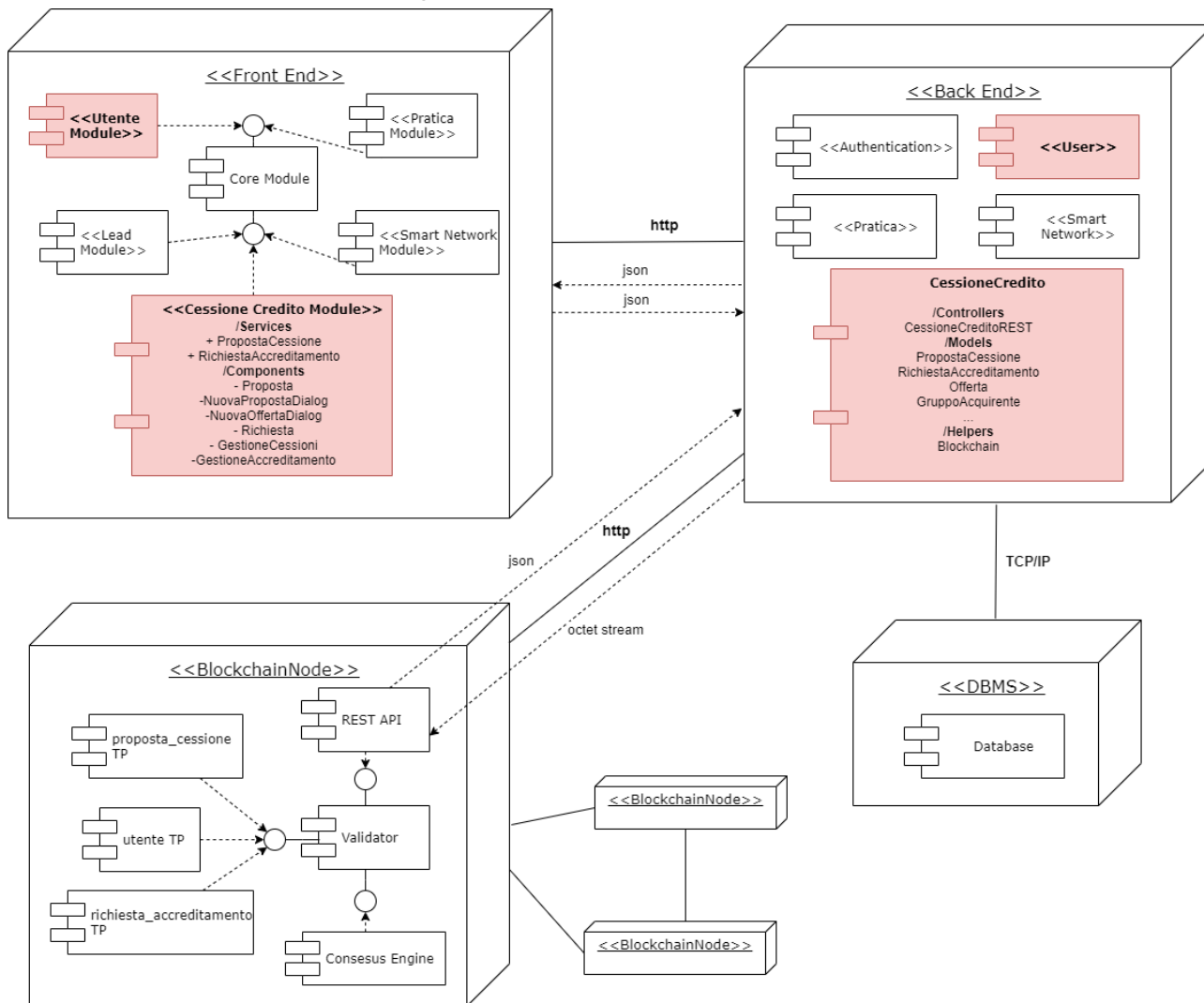


Figura 4.7 - Diagramma dell'architettura della piattaforma

Come si può vedere il front end, realizzato con Angular, è strutturato in diversi moduli che mappano le funzionalità richieste dai vari servizi della piattaforma. L'integrazione nel front end è avvenuta aggiungendo un nuovo modulo "CessioneCredito" che è costituito da vari component per l'interfaccia web dell'applicazione (costituiscono di fatto delle View nel pattern MVC) che comunicano con il back end attraverso i due service definiti per questo modulo: "PropostaCessioneService" e "RichiestaAccreditamentoService". Le funzionalità relative alla gestione dell'account Blockchain sono invece state aggiunte al modulo "Utente" preesistente per facilitare un possibile riutilizzo in sviluppi futuri. Il backend è similmente suddiviso in moduli per i vari servizi della piattaforma. In tutti i moduli è presente un controller che espone una **REST API** per la comunicazione con il front end. Ogni modulo inoltre raccoglie una serie di **ActiveRecord** (classi ORM di Yii 2.0 che rappresentano il Model nel pattern MVC) che gestiscono i modelli dei dati per conto dei controller. Anche in questo caso l'integrazione è avvenuta aggiungendo un modulo Yii 2.0 specifico per il nuovo servizio di cessione del credito

Per quel che riguarda la persistenza dei dati, data la dipendenza di questo nuovo servizio dalle informazioni gestite dal resto della piattaforma, (vedi ad esempio le pratiche da cui creare nuove proposte, i relativi documenti, le informazioni degli utenti registrati, ecc.) è stato riutilizzato e ampliato il **database SQL MariaDB** della piattaforma implementato nel cloud aziendale, affiancandolo alla blockchain. Yii 2.0 si occupa della connessione al database SQL tramite

opportuni driver, mentre l'applicazione client si connette alla rete blockchain attraverso l'API rest per l'invio delle transazioni e la lettura dei dati dello stato globale.

Blockchain

Come già accennato nei capitoli precedenti per aderire ai requisiti di decentralizzazione, sicurezza e trasparenza dei dati è stata sviluppata una rete blockchain privata e permissioned. I due strumenti fondamentali per lo sviluppo della blockchain sono stati il framework Hyperledger Sawtooth e Docker. Di Sawtooth si è già parlato in modo approfondito nel capitolo precedente mentre **Docker** è un progetto open-source che automatizza il processo di deployment attraverso la virtualizzazione del software all'interno di unità standardizzate che prendono il nome di container. I container sono isolati gli uni dagli altri e contengono il software e tutto ciò che è necessario alla sua corretta esecuzione: librerie, i file di configurazione, strumenti sistema, codice, runtime, ecc; pur essendo isolati i container possono comunicare attraverso canali ben definiti. Inoltre il fatto che i container condividano lo stesso kernel del sistema operativo li rende fa sì che necessitino di meno risorse rispetto ad una "classica" macchina virtuale.

Docker è stato particolarmente utile nella fase di sviluppo e di test per creare e testare la rete. Infatti anziché creare una macchina virtuale per ogni singolo nodo della rete, i vari componenti (validator, consensus engine, rest API, transaction processor) dei nodi sawtooth sono stati containerizzati con docker e successivamente i container sono stati configurati e connessi tra di loro attraverso un file yaml per creare la rete. In questo modo i singoli componenti sono riutilizzabili per provare diverse topologie di reti ed è stato possibile avere degli ambienti di sviluppo "usa e getta" disponibili in pochi secondi. Di seguito è riportato, a titolo di esempio, il contenuto del **file yaml** per una rete ad un singolo nodo con un consensus engine semplificato per lo sviluppo e i transaction processor creati ad hoc per questa applicazione:

```
version: '3.6'

services:
# transaction processor che gestisce le configurazioni on-chain
settings-tp:
  image: hyperledger/sawtooth-settings-tp:chime
  container_name: sawtooth-settings-tp
  depends_on:
    - validator
  command: |
    bash -c "
      settings-tp -vv -C tcp://validator:4004
    "
  stop_signal: SIGKILL

# transaction processor che gestisce le proposte di cessione
proposta-cessione-tp-python:
  image: wall94/proposta-cessione-tp-python-local
  container_name: proposta-cessione-tp-python-local
  depends_on:
    - validator
  command: |
    bash -c "
      proposta-cessione-tp-python -vv -C tcp://validator:4004
    "
  stop_signal: SIGKILL

# transaction processor che gestisce le richieste di accreditamento
richiesta-accreditamento-tp-python:
```

```

image: wall94/richiesta-accreditamento-tp-python-local
container_name: richiesta-accreditamento-tp-python-local
depends_on:
  - validator
command: |
  bash -c "
    richiesta-accreditamento-tp-python -vv -C tcp://validator:4004
  "
stop_signal: SIGKILL

```

transaction processor che memorizza le identità degli utenti sulla blockchain

```

utente-tp-python:
  image: wall94/utente-tp-python-local:latest
  container_name: utente-tp-python-local
  depends_on:
    - validator
  command: |
    bash -c "
      utente-tp-python -vv -C tcp://validator:4004
    "
  stop_signal: SIGKILL

```

validatore del nodo

```

validator:
  image: wall94/sawtooth-validator:latest

  container_name: sawtooth-validator
  expose:
    - 4004
    - 8800
    - 5050
  ports:
    - "4004:4004"
  command: |
    bash -c "
      sawadm keygen
      sawset genesis \
        -k /etc/sawtooth/keys/validator.priv \
        -o config-genesis.batch && \
      sawset proposal create \
        -k /etc/sawtooth/keys/validator.priv \
        sawtooth.consensus.algorithm.name=Devmode \
        sawtooth.consensus.algorithm.version=0.1 \
        -o config.batch && \
      sawadm genesis config-genesis.batch config.batch && \
      sawtooth-validator -vv \
        --endpoint tcp://validator:8800 \
        --bind component:tcp://eth0:4004 \
        --bind network:tcp://eth0:8800 \
        --bind consensus:tcp://eth0:5050 \
    "
  stop_signal: SIGKILL

```

API Rest per l'applicazione client

```

rest-api:
  image: hyperledger/sawtooth-rest-api:chime
  container_name: sawtooth-rest-api
  ports:

```

```

    - "8008:8008"
depends_on:
  - validator
command: |
  bash -c "
    sawtooth-rest-api -v --connect tcp://validator:4004 --bind rest-api:8008
  "
stop_signal: SIGKILL

# Consensus Engine di sviluppo
devmode-rust:
  image: hyperledger/sawtooth-devmode-engine-rust:chime
  container_name: sawtooth-devmode-engine-rust
  depends_on:
    - validator
  command: |
    bash -c "
      devmode-engine-rust -v --connect tcp://validator:5050
    "
  stop_signal: SIGKILL

```

Transaction Families (Smart Contract)

Nel capitolo relativo a Sawtooth è stato più volte sottolineato come la netta separazione tra livello applicativo e sistema core della blockchain garantisca una grande flessibilità nella definizione delle **transaction family** (l'equivalente di uno smart contract). Questa grande flessibilità comporta inevitabilmente maggiori responsabilità da parte dello sviluppatore dal momento che per creare una transaction family è necessario:

- Definire un prefisso e un pattern per gli indirizzi dello stato globale in cui memorizzare i dati della transaction family.
- Definire la struttura, il meccanismo di serializzazione e deserializzazione dei dati memorizzati nello stato e del payload delle transazioni di una determinata transaction family.
- Creare un transaction processor che definisca le operazioni che possono essere eseguite sui dati di una transaction family, stabilendo anche la validità delle transazioni che le attivano.

Prima di descrivere le transaction family definite per questa applicazione è bene fare alcune considerazioni riguardo ai meccanismi di serializzazione e allo sviluppo dei transaction processor:

- Tutte le transaction family definite per questa applicazione utilizzano i protocol buffer (o protobuf) per la definizione e serializzazione della struttura dati memorizzata nello stato e del payload delle transazioni. **Protobuf** è una libreria open source cross-platform per la serializzazione di dati strutturati, particolarmente utile nello sviluppo di programmi che devono comunicare attraverso una rete o per la memorizzazione di dati. Protobuf si basa su un interface description language⁷ (o interface definition language) che descrive la struttura di dati e su un programma compilatore che, partendo da questa descrizione, genera il codice sorgente e permette di generare uno stream di byte che rappresenta questa

⁷ Un interface description language è un linguaggio che consente ad un programma, scritto in un certo linguaggio, di comunicare con un altro programma scritto in linguaggio diverso.

struttura (serializzazione). Le strutture dati (chiamate *message*) vengono definite in file con estensione `.proto`: un message è costituito da un insieme di campi indicizzati e tipizzati con tipi di dato primitivi quali ad esempio `string`, `int32`, `int64`, `float`, `bytes`, `bool`, `repeated` (per i vettori), `map` o altri message precedentemente definiti. Questi file vengono compilati con programma `protoc` generando delle classi di codice sorgente in un determinato linguaggio di programmazione che permettono di istanziare e gestire queste strutture dati come oggetti, fornendo dei metodi per serializzarle e la deserializzarle in maniera indipendente dallo specifico linguaggio. In questo modo due programmi che devono scambiarsi dei dati compilano lo stesso file `proto` nei rispettivi linguaggi di programmazione e inviano i dati serializzati che entrambi sono in grado di deserializzare utilizzando il codice sorgente ottenuto dalla compilazione. Nel caso delle *transaction family* i protocol buffer vengono utilizzati dal *transaction processor* per serializzare i dati nel *global state* e dal client dell'applicazione per serializzare il payload delle transazioni che invia alla blockchain (in particolar modo al *transaction processor* che è in grado di deserializzarlo).

- Per quanto riguarda i *transaction processor* invece è bene precisare che *sawtooth* mette a disposizione dei **SDK (software developer kit)** per diversi linguaggi di programmazione che aiutano non solo sviluppare il *transaction processor* ma forniscono anche le interfacce per comunicare con il componente validatore del nodo in cui il *transaction processor* è in esecuzione. Grazie all'SDK sviluppare un *transaction process* si riduce a strutturare e scrivere la funzione `apply(context, transaction)` della classe *handler*: questa funzione viene invocata su richiesta del *validator* del nodo per ogni transazione della *transaction family* che il *transaction processor* è in grado di gestire; e ha la responsabilità di aggiornare lo stato globale, a cui può accedere attraverso l'oggetto `context`, utilizzando le informazioni dell'oggetto `transaction` e di controllare eventuali transazioni non valide (relativamente alle logiche interne della *transaction family*) che segnala al *validator* come eccezione `InvalidTransaction`. Per questa applicazione si è deciso di utilizzare il `sawtooth-sdk-python` e sviluppare quindi i *transaction processor* in linguaggio `python`.

Di seguito vengono descritte le caratteristiche sopracitate delle tre *transaction family* definite per questa applicazione: **proposta_cessione**, **richiesta_accREDITAMENTO** e **utente**.

proposta_cessione

Questa *transaction family* permette di memorizzare le informazioni più importanti legate all'entità proposta di cessione e di gestirne l'aggiornamento man mano che essa evolve, a partire dalla creazione di una nuova proposta fino alla firma del contratto e/o alla sua liquidazione. *proposta_cessione* dipende dalla *transaction family* *utente* per quanto riguarda i controlli sull'autorizzazione delle operazioni di una proposta di cessione.

Indirizzi

Gli indirizzi della *transaction family* *proposta_cessione* sono costruiti in questo modo: un prefisso costituito dai caratteri "7e049a" (i primi 6 caratteri hex dell'hash sha512 della stringa "proposta_cessione") seguito dai primi 64 caratteri hex dell'hash sha512 dell'id della proposta di cessione memorizzata (o da memorizzare) a quel determinato indirizzo.

Strutture dati

Le strutture dati per lo stato e il payload delle transazioni della *transaction family* *proposta_cessione* sono definite attraverso i message `protobuf` "PropostaCessioneState" e "PropostaCessionePayload" riportati di seguito.

```

message PropostaCessioneState {

    enum StatoPropostaCessione {
        PREPARAZIONE = 0;
        PROPOSTA = 1;
        PRESA_IN_CARICO = 2;
        VALIDATA = 3;
        ACQUISTATA = 4;
        INVALIDATA = 5;
        CONTRATTO_DA_FIRMARE = 6;
        CONTRATTO_FIRMATO = 7;
        DA_LIQUIDARE = 8;
        LIQUIDATA = 9;
    }
    int32 id = 1;
    TipologiaCessione tipologia = 2;
    StatoPropostaCessione stato = 3;
    // id del cedente
    int32 id_cedente = 4;
    // id del gruppo acquirente
    int32 id_gruppo_acquirente = 5;
    // id del sal selezionato per la proposta
    int32 id_sal = 6;
    // se la proposta è gestita con meccanismo di borsa credito o meno
    bool borsa_credito = 7;
    // id dell'acquirente che si aggiudica la proposta (caso borsa del credito)
    int32 id_acquirente_borsa = 8;
    string data_creazione = 9;
    string note = 10;
    // valore in cent di €
    uint32 importo = 11;
    // valore in cent di €
    uint32 credito = 12;
    repeated OffertaProposta offerte = 13;
    repeated HashedFile documenti = 14;
    repeated HashedFile contratti = 15;
}

enum TipologiaCessione {
    PREISTRUTTORIA_110 = 0;
    ISTRUTTORIA_110 = 1;
    ISTRUTTORIA_BONUS = 2;
}

enum StatoOffertaProposta {
    DA_VALUTARE = 0;
    ACCETTATA = 1;
    RIFIUTATA = 2;
}

message OffertaProposta {
    int32 id = 1;
    int32 id_acquirente = 2;
    StatoOffertaProposta stato = 3;
}

```

```

// entità dell'offerta in cent di €
uint32 valore = 4;
string data_creazione = 5;
string data_ultimo_aggiornamento = 6;
}

message HashedFile {
    int32 id = 1;
    string hash = 2;
}

```

Come è possibile notare PropostaCessioneState memorizza le informazioni più importanti di una proposta di cessione quali la tipologia, lo stato, l'id del gruppo acquirente e dell'acquirente borsa (in caso la proposta sia stata presentata su borsa del credito). Nel caso di tipologia e stato sono state create delle enumerazioni che riportano rispettivamente le due tipologie di cessione e tutti i possibili stati in cui si può trovare la proposta. Inoltre è prevista la memorizzazione degli hash dei file dei vari documenti (tra cui i bonifici) e dei contratti nei campi documenti e contratti che di fatto costituiscono delle liste hash di file. Le offerte invece vengono memorizzate nella lista offerte e ogni offerta ha una struttura più complessa anch'essa presente in figura. Infine è bene notare come il valore relativo all'importo e al credito siano memorizzati in centesimi (quindi interi) per evitare problemi di rappresentazione con in numeri in virgola mobile (float).

```

message PropostaCessionePayload {
    // payload per aggiornare lo stato della proposta
    message AggiornamentoStato {
        int32 id_proposta = 1;
        PropostaCessioneState.StatoPropostaCessione nuovo_stato = 2;
        string note = 3;
        int32 id_acquirente_borsa = 4;
    }
    // payload per caricare nuove offerte o aggiornare quelle esistenti (a seconda)
    message AggiornamentoOfferte {
        int32 id_proposta = 1;
        repeated OffertaProposta offerte_aggiornate = 2;
    }
    // payload per eliminare un'offerta
    message EliminazioneOfferte {
        int32 id_proposta = 1;
        repeated int32 id_offerte = 2;
    }
    // payload per caricare nuove hash di file o aggiornare quelli esistenti
    message AggiornamentoFile {
        int32 id_proposta = 1;
        repeated HashedFile file_aggiornati = 2;
    }
    enum PayloadType {
        PAYLOAD_TYPE_UNSET = 0;
        CREAZIONE_PROPOSTA = 1;
        AGGIORNAMENTO_STATO = 2;
        AGGIORNAMENTO_OFFERTE = 3;
        AGGIORNAMENTO_DOCUMENTI = 4;
        AGGIORNAMENTO_CONTRATTI = 5;
        ELIMINAZIONE_OFFERTE = 6;
    }
    PayloadType type = 1;
}

```



```

    bytes data = 2;
}

```

PropostaCessionePayload descrive invece la struttura del payload di una transazione di questa famiglia. Per gestire l'eterogeneità dei dati utilizzati nelle varie operazioni di modifica il payload è costituito da due campi: type, un'enumerazione che definisce tutte le possibili operazioni della proposta di cessione e data, un campo di tipo bytes che contiene la versione serializzata dei dati utilizzati nell'operazione.

richiesta_accreditamento

Questa transaction family permette di memorizzare le informazioni più importanti legate all'entità richiesta di accreditalmento e di gestirne l'aggiornamento man mano che essa evolve, a partire dalla creazione di una nuova richiesta fino all'accreditamento dell'utente.

richiesta_accreditamento dipende dalla transaction family utente per quanto riguarda i controlli sull'autorizzazione delle operazioni di una richiesta di accreditalmento.

Indirizzi

Gli indirizzi della transaction family richiesta_accreditamento sono costruiti in questo modo: un prefisso costituito dai caratteri "fd80" (i primi 6 caratteri hex dell'hash sha512 della stringa "richiesta_accreditamento") seguito dai primi 64 caratteri hex dell'hash sha512 dell'id della richiesta di accreditalmento memorizzata (o da memorizzare) a quel determinato indirizzo.

Strutture dati

Le strutture dati per lo stato e il payload delle transazioni della transaction family proposta_cessione sono definite attraverso i message protobuf "RichiestaAccreditamentoState" e "RichiestaAccreditamentoPayload" riportati di seguito.

```

message RichiestaAccreditamentoState {
    enum StatoRichiestaAccreditamento {
        PREPARAZIONE = 0;
        DA_VALIDARE = 1;
        ACCREDITATO = 2;
        NON_VALIDO = 3;
    }
    int32 id = 1;
    StatoRichiestaAccreditamento stato = 2;
    // id del cedente
    int32 id_cedente = 3;
    // id del gruppo acquirente a cui viene fatta la richiesta
    int32 id_gruppo_acquirente = 4;
    // se la richiesta è per accreditarsi alla borsa del credito
    bool borsa_credito = 5;
    string data_creazione = 6;
    string data_accreditamento = 7;
    string note = 8;
    repeated HashedFile documenti = 9;
}

```

RichiestaAccreditamentoState descrive la struttura dello stato per questa transaction family, riportando le informazioni essenziali nei campi id_cedente che contiene l'id dell'utente che ha creato la richiesta e id_gruppo_acquirente che contiene l'id del gruppo acquirente a cui la richiesta è stata inviata. Come per PropostaCessioneState tutti i documenti relativi ad una richiesta vengono notarizzati salvando gli 'hash dei documenti nel campo documenti.

```

message RichiestaAccreditamentoPayload {
  // payload per aggiornare lo stato della richiesta
message AggiornamentoStato {
  int32 id_richiesta = 1;
  RichiestaAccreditamentoState.StatoRichiestaAccreditamento nuovo_stato = 2;
  string note = 3;
  string data_accreditamento = 4;
}
// payload per caricare nuove hash di file o aggiornare quelli esistenti
message AggiornamentoFile {
  int32 id_richiesta = 1;
  repeated HashedFile file_aggiornati = 2;
}

enum PayloadType {
  PAYLOAD_TYPE_UNSET = 0;
  CREAZIONE_RICHIESTA = 1;
  AGGIORNAMENTO_STATO = 2;
  AGGIORNAMENTO_DOCUMENTI = 3;
}

PayloadType type = 1;
bytes data = 2;
}

```

RichiestaAccreditamentoPayload descrive invece la struttura del payload di una transazione di questa famiglia. Per gestire l'eterogeneità dei dati utilizzati nelle varie operazioni di modifica il payload è costituito da due campi: type, un'enumerazione che definisce tutte le possibili operazioni della richiesta di accreditamento e data, un campo di tipo bytes che contiene la versione serializzata dei dati utilizzati nell'operazione.

utente

Questa transaction family permette di memorizzare le informazioni essenziali per autenticare un utente della blockchain e stabilire le sue autorizzazioni circa l'esecuzione di transazioni delle altre transaction family.

Indirizzi

Gli indirizzi della transaction family utente sono costruiti in questo modo: un prefisso costituito dai caratteri "c7e44f" (i primi 6 caratteri hex dell'hash sha512 della stringa "utente") seguito dai primi 64 caratteri hex dell'hash sha512 della chiave pubblica (in formato esadecimale) dell'utente i dati del quale sono memorizzati (o da memorizzare) a quel determinato indirizzo.

Strutture dati

La transaction family utente utilizza la stessa struttura dati per lo stato e per il payload definita attraverso il message protobuf Utente.

```

message Utente {
  enum Ruolo {
    CEDENTE = 0;
    ACQUIRENTE = 1;
    VALIDATORE = 2;
    REVISORE_FISCALE = 3;
  }
}

```

```
string public_key = 1;  
int32 id = 2;  
Ruolo ruolo = 3;  
int32 id_gruppo_acquirente = 4;  
}
```

Utente descrive l'insieme delle informazioni che identificano un utente autorizzato a eseguire transazioni sulla blockchain. L'informazione più importante è quella contenuta nel campo `public_key` che contiene la chiave pubblica dell'utente e permette di associare l'utente che firma le transazioni delle altre transaction family (di cui si conosce la chiave pubblica) con la sua "identità" nella blockchain.

Sicurezza

Questo capitolo approfondisce uno degli aspetti più importanti nella realizzazione dell'applicazione: la sicurezza. Verranno descritte pertanto le soluzioni di sicurezza implementate su diversi fronti, discutendo anche sulle varie opzioni prese in considerazione.

Sicurezza della rete Blockchain

Come si è già anticipato nel capitolo dedicato alla blockchain, uno dei requisiti che ha portato all'adozione di questa tecnologia è stato proprio la **sicurezza dei dati memorizzati**. Tuttavia non si è mai specificato, per questa particolare applicazione, come la blockchain dovesse contribuire ad ottemperare questo requisito. La sicurezza dei dati di cui stiamo parlando si riferisce fondamentalmente a due obiettivi:

- Assicurarsi che i dati sensibili degli utenti e delle pratiche memorizzati siano accessibili solo dagli utenti che ne hanno il diritto.
- Garantire l'integrità dei dati memorizzati, evitando la loro possibile corruzione, perdita o alterazione da parte di persone non autorizzate.

Riguardo al primo punto, inizialmente si è pensato di affidare questa responsabilità interamente al framework scelto (Hyperledger Sawtooth), andando a memorizzare tutti i dati relativi al nuovo servizio nello stato globale della blockchain. Come già accennato, Sawtooth, in nome della sua notevole flessibilità, non vincola all'utilizzo di un'unica struttura dati permettendo allo sviluppatore di scegliere il modo in cui serializzare i dati nello stato globale. Questo, se da una parte implica indirettamente che il framework non fornisca nativamente uno strumento di autenticazione e autorizzazione per gestire l'accesso (in lettura) ai dati, dall'altra non vieta nemmeno allo sviluppatore di **cifrare i dati** prima di serializzarli e memorizzarli nello stato globale, implementando la cifratura e decifratura dei dati nell'applicazione client o direttamente nei transaction processor interessati se si volesse cifrare anche il contenuto delle transazioni. Inoltre per la cifratura dei dati si sarebbero potute riutilizzare le chiavi private dei wallet degli utenti della blockchain o comunque derivare da esse delle chiavi per cifrare/decifrare questi dati (crittografia simmetrica). Altra opzione valutata è stata quella di intervenire sulla **sicurezza della rete dall'esterno**, anziché direttamente sui dati, isolando i vari nodi della rete e permettendo unicamente la comunicazione con gli altri nodi e con l'applicazione client. In questo modo il requisito di sicurezza si sarebbe concentrato sulla costruzione di questo sistema di autenticazione e autorizzazione esterno. Entrambe le idee sono però state quasi subito scartate a causa di problemi che emergevano circa l'efficienza e la sicurezza/trasparenza dell'applicazione:

- Uno dei requisiti dell'applicazione era che questa fosse integrata come servizio all'interno di una piattaforma più ampia. Questa necessità deriva dal fatto che l'applicazione si basa su dati provenienti da altri servizi della piattaforma: nello specifico i dati e i documenti delle pratiche a partire dalle quali creano le cessioni del credito e delle informazioni personali degli utenti registrati alla piattaforma. Utilizzare esclusivamente la blockchain per memorizzare le informazioni avrebbe comportato un delicato processo di migrazione di questi dati dal database aziendale allo stato della blockchain o alternativamente una duplicazione in modo da rendere il servizio "standalone".

- La blockchain non può essere paragonata a un DBMS. Infatti la maggior parte dei framework blockchain che supportano smart contract consentono semplici e limitate operazioni per la memorizzazione e non dispongono di un linguaggio per interrogazioni complesse. Hyperledger Sawtooth nello specifico consente unicamente la lettura e la scrittura di un blocco di dati da e su un determinato indirizzo dello stato globale. Questo come si può immaginare renderebbe quindi piuttosto inefficiente l'applicazione. Sebbene esistano dei framework per lo sviluppo di "**Blockchain-based database**", molti non supportano gli smart contract e in ogni caso sono soluzioni in cui un DBMS "classico" è affiancato ad un registro decentralizzato (di fatto non molto dissimili alla soluzione che è stata utilizzata in questo progetto)[22].
- La soluzione che prevede la cifratura dei dati porta a delle problematiche relative alla trasparenza dei dati memorizzati. Poniamoci ad esempio in uno scenario in cui si è aperta una disputa tra cedente e acquirente riguardante la vendita di una cessione: se uno dei ha l'accesso esclusivo a una parte dei dati (poiché cifrati con la propria chiave privata), potrebbe rifiutarsi di consentire l'accesso all'altro o ad una terza parte per dirimere la disputa. In questo modo si perderebbero quindi i benefici nell'utilizzo della blockchain circa la responsabilità degli utenti e la non ripudiabilità delle loro azioni.
- La soluzione che prevede invece l'isolamento dei singoli nodi della rete porta a delle complicazioni dal punto di vista della sicurezza. Infatti la distribuzione dei dati e la loro replicazione su molteplici nodi se da una parte diminuisce la probabilità di perdite e corruzione dei dati, dall'altra aumenta i rischi per la sicurezza, considerando che all'aumentare del numero di nodi aumenta la superficie di attacco che un utente malevolo può sfruttare.

Per questi motivi le due opzioni iniziali sono state scartate in favore di una **soluzione ibrida** che consiste nell'affiancare il DBMS aziendale preesistente alla nuova rete blockchain. Nel DBMS continuano ad essere salvati tutti i dati della piattaforma (compresi i dati del nuovo servizio), semplificando notevolmente l'integrazione del nuovo servizio con la piattaforma e riutilizzando i protocolli per la sicurezza del database già esistenti. Nello stato globale della blockchain vengono invece parte delle informazioni già presenti del db. Queste informazioni sintetizzando in maniera essenziale informazioni "non sensibili" (riguardanti le proposte di cessione, le richieste di accreditamento e le identità degli utenti della blockchain), ovvero senza che esse possano essere associate agli utenti veri e propri della piattaforma a meno di non avere accesso alle informazioni complete. In questo modo questi dati possono essere resi "pubblici" garantendo un certo livello di trasparenza in caso di dispute (un analogo discorso vale per le transazioni nel registro della blockchain) e non necessitano, almeno in lettura, di particolari accorgimenti.

Per quanto riguarda invece il secondo obiettivo, considerando quanto detto nei capitoli precedenti, appare chiaro come garantire l'integrità e l'incorruttibilità dei dati sia uno dei punti di forza della tecnologia blockchain grazie fondamentalmente a tre caratteristiche:

- La ridondanza dei dati memorizzati su molteplici nodi della rete.
- Il tracciamento delle modifiche allo stato attraverso la memorizzazione delle transazioni eseguite nei blocchi del registro distribuito.
- L'algoritmo di consenso che permette di mantenere la coerenza dei dati nello stato sui vari nodi della rete, impedendo, entro determinati limiti, alterazioni e corruzioni dello stato accidentali e non.

Di conseguenza non sono state necessarie particolari accorgimenti in questo senso in quanto queste proprietà sono garantite dalla scelta della tecnologia blockchain stessa.

Sicurezza della firma digitale

Un altro aspetto fondamentale per la sicurezza dell'applicazione è sicuramente la firma digitale. La **firma digitale (digital signature)** è uno schema matematico per verificare l'autenticità di documenti o messaggi digitali utilizzando tecniche di crittografia asimmetrica (necessitano quindi che il firmatario sia in possesso di una coppia di chiavi crittografiche di cui una pubblica e una privata). Sotto molti aspetti la firma digitale può considerarsi l'equivalente della classica firma autografa (quella fatta a mano), sebbene una firma elettronica opportunamente implementata sia molto più difficile da falsificare di quest'ultima. Di fatto la firma digitale nasce inizialmente con l'obiettivo di fornire le stesse caratteristiche e possibilità di applicazione della firma autografa:

- **Autenticazione:** Nonostante un messaggio possa includere informazioni circa il mittente, queste informazioni potrebbero non essere accurate. La firma digitale può essere utilizzata per autenticare l'identità del mittente. Infatti se si associa il possesso di una chiave privata per la firma digitale ad un determinato utente, la verifica della validità della firma digitale, conoscendo la relativa chiave pubblica, consente di stabilire che il messaggio è stato inviato proprio da quell'utente.
- **Integrità:** In alcuni casi il mittente e il ricevente di un messaggio o i contraenti di un contratto devono accertarsi che il messaggio o il documento del contratto non vengano alterato durante la trasmissione. Se un messaggio è firmato digitalmente una qualsiasi modifica al messaggio invalida la firma. La maggior parte degli schemi di firma digitale infatti si basano su funzioni di hash che rendono computazionalmente impossibile modificare un messaggio e la sua firma digitale in modo da crearne uno nuovo con una firma valida.
- **Non Ripudio:** Probabilmente il non ripudio è la proprietà più importante per una firma digitale e non. Consiste nel garantire che un utente che ha firmato un documento o un messaggio non possa negare negare in futuro di averli fatti. Nel caso specifico della firma digitale questo implica anche che il possesso della chiave pubblica non dia la possibilità ad un utente "disonesto" di falsificare una firma valida.

Ogni schema di firma digitale si compone di tre algoritmi:

- Un **algoritmo di generazione chiavi** che genera randomicamente una chiave privata e calcola la corrispondente chiave pubblica attraverso una funzione non invertibile (one-way function).
- Un **algoritmo di firma** vero e proprio che data un messaggio e una chiave privata genera una firma.
- Un **algoritmo di verifica** che dato un messaggio, una chiave privata e una firma ne stabilisce l'autenticità [23].

La blockchain fa ampio utilizzo della firma digitale al punto che essa si può considerare come uno dei pilastri fondamentali su cui si basa la sicurezza di questa tecnologia. Viene utilizzata principalmente per **verificare l'autenticità delle transazioni**. Ogni utente utilizza la propria chiave

privata per firmare la transazione prima di inviarla al nodo validatore che prima di eseguire la transazione va a controllare la validità della firma. Nei registri distribuiti per lo scambio di criptovaluta come Bitcoin o Ethereum, in cui ogni transazione è letteralmente un trasferimento di criptovaluta da un indirizzo a un'altro, la firma digitale è in particolar modo utilizzata per dimostrare sai che un utente è in possesso dei fondi che sta trasferendo sia evitare che altri utenti possano spenderli. In Hyperledger Sawtooth invece la firma digitale viene più semplicemente utilizzata per garantire che la chiave pubblica nell'header della transazione firmata (che identifica un utente in Sawtooth) appartiene all'utente che ha generato la transazione (la stessa cosa vale per i batch). Questo consente al validator e al transaction processor di essere certi sull'identità di chi sta eseguendo la transazione e, a partire da questa informazione (la chiave pubblica), gestire l'autorizzazione a modificare lo stato e tracciare l'attività di un determinato utente.

Per quanto riguarda lo schema specifico, Hyperledger al momento supporta solo ed unicamente **l'Elliptic Curve Digital Signature Algorithm (ECDSA)**, una variante le Digital Signature Algorithm (DSA) che utilizza la crittografia ellittica [24]. Se paragonato ad altri schemi di firma più classici si può notare dalla tabella in figura 5.1 come l'ECDSA permetta di utilizzare chiavi più corte (quindi più facilmente gestibili) e richieda requisiti computazionali mediamente inferiori, pur mantenendo una buona sicurezza. Questo perché l'ECDSA basa la propria sicurezza sul **problema del logaritmo discreto su curve ellittiche** a differenza degli schemi più "classici" (quali DSA e RSA) che si basano sul problema della fattorizzazione di numeri primi grandi [25]. La curva ellittica utilizzata da Sawtooth prende il nome di **secp256k1** (la stessa utilizzata in Bitcoin) è standardizzata dal NIST e prevede l'utilizzo di chiavi private di 256 bit. L'implementazione dell'ECDSA è stata fornita dalla suite crittografica **phpseclib**, basata su openssl e integrata nel back end dell'applicazione.

| Bits of Security | Symmetric Algorithm | RSA | ECC |
|------------------|---------------------|-------------|-----------------|
| 80 | 2TDEA | $k = 1024$ | $f = 160 - 223$ |
| 112 | 3TDEA | $k = 2048$ | $f = 224 - 255$ |
| 128 | AES-128 | $k = 3072$ | $f = 256 - 383$ |
| 192 | AES-192 | $k = 7680$ | $f = 384 - 511$ |
| 256 | AES-256 | $k = 15360$ | $f = 512+$ |

Figura 5.1 - Tabella di confronto sicurezza/lunghezza chiave tra AES, RSA e ECC ([25])

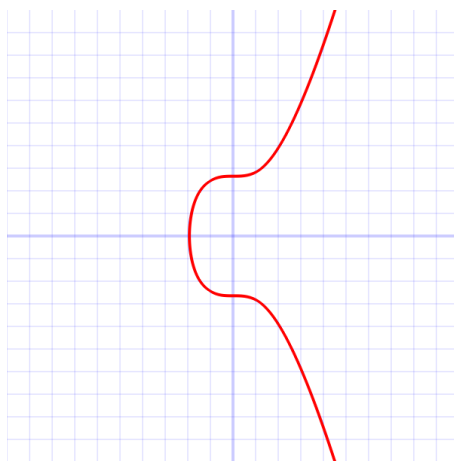


Figura 5.2 - Grafico della curva ellittica secp256k1's ($y^2 = x^3 + 7$) nel dominio reale (<https://en.bitcoin.it/wiki/File:Secp256k1.png>)

Sicurezza delle chiavi

Nella sezione precedente si è visto come la firma digitale richieda che ogni "firmatario" sia in possesso di una coppia di chiavi pubblica-privata. Grazie alla crittografia ellittica e in particolare al

problema del logaritmo discreto su curve ellittiche ogni utente può rendere pubblica la propria chiave pubblica e dimostrare l'autenticità della propria firma senza correre il rischio che da essa si possa risalire alla relativa chiave privata (in quanto appunto computazionalmente impossibile). L'attenzione sulla sicurezza si sposta a questo punto sulla gestione delle chiavi stesse. Per **garantire la sicurezza nella gestione delle chiavi** in questa applicazione ci si è concentrati sul raggiungimento di due obiettivi fondamentali:

- Mantenere la **segretezza delle chiave private** permettendone l'utilizzo per la firma al solo utente che ne detiene il possesso.
- Mantenere **l'associazione di ogni utente con il proprio wallet**, garantendo quindi la corrispondenza utente-chiave pubblica, in modo che ogni utente possa essere identificato sulla blockchain.

Nel processo di progettazione del sistema di gestione delle chiavi è apparso chiaro fin da subito che, dato il basso livello di consapevolezza degli utenti della piattaforma in merito a chiavi private-pubbliche, crittografia asimmetrica, sicurezza delle password, ecc. la segretezza della chiave privata non potesse essere affidata esclusivamente all'utente dell'applicazione. Questo in particolare avrebbe comportato diversi problemi in caso di furti o smarrimenti della chiave privata stessa. Tuttavia nemmeno l'approccio diametralmente opposto, ovvero affidare la gestione del wallet dell'utente interamente all'applicazione, in maniera totalmente trasparente per l'utente, era una soluzione desiderabile. L'obiettivo su cui ci si è concentrati è stato dunque un compromesso tra questi due estremi: l'applicazione doveva gestire il wallet "per conto" dell'utente che, responsabilizzato, ne manteneva comunque il controllo e doveva garantire anche meccanismi di recupero e ripristino delle chiavi in caso di problemi.

Inizialmente è stata valutata una possibile soluzione basata sull'**Attribute Based Encryption** (Crittografia su base attributi) per la protezione della chiave privata. L'Attribute Based Encryption (ABE) è un tipo di crittografia asimmetrica in cui la chiave di cifratura e il messaggio cifrato dipendono da una serie di attributi dell'utente. In un tale sistema decifrare un messaggio cifrato è possibile solo se l'insieme di attributi che possiede l'utente "corrispondono" agli attributi del testo cifrato. L'ABE si basa su due componenti fondamentali:

- **Insieme di attributi**: una lista di proprietà/attributi di una utente (es. Età=30, Ruolo=Acquirente).
- **Albero di accesso (access tree o policy tree)**: un'espressione che coinvolge alcuni degli attributi definendo una politica di accesso (es. Ruolo == Admin || Ruolo == Cedente && Età > 40)

*Esistono essenzialmente due tipi di schemi di crittografia basati su attributi: la crittografia basata su attributi con politica di chiave (**key-policy o KP-ABE**) e la crittografia basata su attributi con politica di crittostesto (**ciphertext-policy o CP-ABE**). I due schemi sono speculari nella gestione delle due operazioni fondamentali: la cifratura dei dati e la generazione delle chiavi segrete; infatti, nello schema KP-ABE le chiavi segrete degli utenti vengono generate in base a un albero di accesso che definisce l'ambito dei privilegi dell'utente interessato, mentre i dati vengono cifrati su un insieme di attributi. Di contro, CP-ABE utilizza gli alberi di accesso per crittografare i dati e le chiavi segrete degli utenti vengono generate su un insieme di attributi [26][27].*

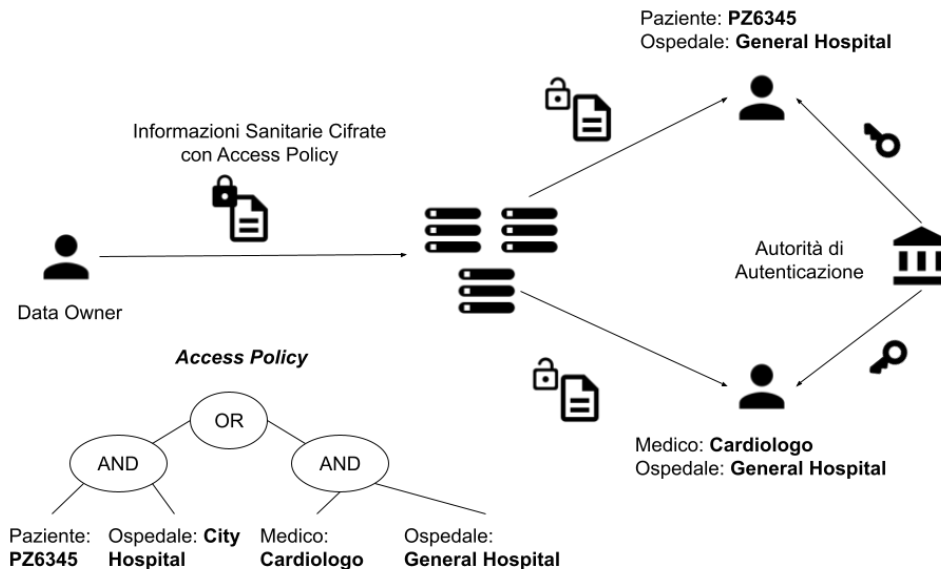


Figura 5.3 - Condivisione di informazioni con CP-ABE

L' ABE era stata considerata principalmente poichè in linea teorica avrebbe permesso di utilizzare le informazioni degli utenti come attributi (eventualmente in parte memorizzate nel database e in parte fornite appositamente dall'utente) per generare la password senza doverla richiedere esplicitamente all'utente. Basandosi sugli attributi inoltre l'ABE consentiva di generare un unico testo cifrato decifrabile con diverse combinazioni di attributi (e quindi diverse chiavi) che soddisfano la policy, aprendo interessanti possibilità per il meccanismo di recupero/ripristino della chiave privata (cifrata con ABE). Tuttavia analizzando questo approccio da un punto di vista pratico sono emerse alcune problematiche che rischiavano di complicare ulteriormente la gestione della sicurezza anzichè semplificarla:

- Un sistema di ABR necessita di un'autorità centrale che gestisca le chiavi private per la blockchain tramite ABE (cifratura, decifratura e generazione delle chiavi attribute based degli utenti)
- Bisognerebbe gestire con autenticazione e autorizzazione l'accesso alle funzionalità che fornisce questa autorità (altrimenti chiunque potrebbe crearsi la propria chiave con attributi o policy tree che in realtà non ha)
- La chiave di cifratura di un utente è generata a partire dall'insieme dei suoi attributi o da un albero e questa deve essere conservata segretamente. Inoltre per generarla l'autorità centrale utilizza una sua chiave privata (un po' come avviene per le Public Key Infrastructure) che deve essere tenuta assolutamente segreta (in letteratura si consiglia addirittura l'utilizzo di un Hardware Security Module). Questa chiave costituisce un single point of failure in quanto la sua compromissione metterebbe a rischio tutte le chiavi con essa generate.
- Considerando che l'accesso all'autorità centrale deve essere in qualche modo autorizzato, il recupero della chiave privata ABE smarrita da un utente potrebbe essere semplicemente una nuova richiesta di generazione della chiave a partire dai suoi attributi una volta che l'utente è stato autenticato dall'applicazione. Quindi cifrare una chiave privata della blockchain in modo che possa essere decifrata o dal proprietario o da un qualche amministratore grazie all'ABE "perde un po' di significato".

A causa dei problemi evidenziati questa soluzione è stata scartate in favore una seconda opzione di gestione che è risultata quella essere definitiva. Questa soluzione prevede di memorizzare la chiave pubblica e quella privata in file separati (con estensione .pem) nel file system del back end dell'applicazione in modo tale da essere accessibili solo tramite essa e utilizzando il **Public-Key Cryptography Standards #8⁸ (PKCS8)** per la serializzazione delle chiavi. Per proteggere la chiave privata il PKCS8 supporta la cifratura con passphrase; tra i molteplici cifrari supportati, in questo caso si è scelto di utilizzare **aes128-CBC-PAD** come schema crittografico unito all'**hmacWithSHA256** per l'autenticazione del messaggio, in quanto consigliato dallo standard per questo tipo di applicazioni [28].

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIHsMfcGCSqGSIb3DQEFDTBKMCKGCSqGSIb3DQEFDDAcBAj0W8EG4Wh3pQICCAAw
DAYIKoZIhvcNAgkFADAdBgIghkgBZQMEAQIEEMYT0fo16aWA3k1FZto6DhoEgZCX
KXog0k0jX4NBEcm4fj3zmvYC8DA3DCDR5egK4Sqp6Xgt+0zITroboq14I3G+cjQH
RnSh6hJw65QdWezB3wXut0/+54TvEIUXlRBKi6nYvONN/4UHPKSSyoCcJaApgCOj
gBQv5HM0wPSNZwuWxglA7kLu0siCxHLIh1i1AGM8vs6dw6XKGrBjsX0+TW1B7so=
-----END ENCRYPTED PRIVATE KEY-----
```

Esempio di chiave privata cifrata e serializzata nel formato PKCS #8

Al momento della registrazione dell'account blockchain all'utente viene richiesto di inserire una nuova password e a rispondere a delle domande di sicurezza. Per garantire un certo livello di sicurezza la password inserita deve rispettare alcuni criteri: almeno 8 caratteri di cui una lettera maiuscola, una minuscola un numero e un carattere speciale (non alfanumerico). La password inserita viene utilizzata come **passphrase** per la chiave privata ottenendo una versione cifrata della chiave. Le risposte alle domande di sicurezza vengono invece concatenate con altre informazioni "immutabili" personali dell'utente memorizzate nel database quali ad esempio l'id dell'utente, il suo username e il suo codice fiscale, in modo da ottenere una **password di recupero** con cui si va a cifrare la stessa chiave privata ottenendo una seconda versione cifrata. In sostanza quindi per ogni utente registrato alla blockchain si avranno tre file .pem: uno con la chiave privata cifrata con la password "principale", uno con la chiave privata cifrata con la password di recupero e uno con la chiave pubblica in chiaro. Si noti come né la password di recupero né le risposte alle domande di sicurezza necessitano di essere memorizzate. La password principale viene richiesta all'utente ogni volta che deve essere eseguita una transazione sulla blockchain, in quanto necessaria per decifrare la chiave privata che la firma. In questo senso questa password costituisce il secondo fattore (il primo è il login alla piattaforma preesistente) di autenticazione esclusivo per il nuovo servizio di cessione del credito. La password di recupero invece viene generata quando l'utente richiede il **reset della password**, dovendo fornire le risposte alle domande di sicurezza e la nuova password principale. In questo modo l'utente mantiene il controllo sul proprio wallet attraverso una password non dovendo gestire in prima persona la sicurezza della chiave privata e disponendo di un meccanismo di ripristino in caso di smarrimento della password.

⁸ Il Public-Key Cryptography Standards #8 è uno standard per la memorizzazioni di informazioni crittografiche di chiavi private creata dagli RSA Laboratories. Lo standard permette di salvare le chiavi cifrandole con una passphrase e supporta diversi cifrari. Le chiavi vengono generalmente serializzate utilizzando il formato PEM con codifica base64

Sicurezza dell'identità su Blockchain

L'ultimo aspetto di sicurezza dell'applicazione che si vuole approfondire è quello che riguarda l'**identità sulla blockchain** su cui si basa il meccanismo di permessi. Più volte è stato sottolineato che in Hyperledger Sawtooth (ma in generale nella maggior parte dei framework blockchain) di base, l'unico modo per distinguere gli utenti è la chiave pubblica indicata nelle transazioni che essi firmano. In assenza di un meccanismo di autorizzazione "esplicito" chiunque può generare una chiave privata secp256k1 valida per firmare le transazioni sulla blockchain. Per questo motivo è stato necessario considerare un modo per autenticare gli utenti in modo da poter loro assegnare dei permessi relativi alle operazioni associate alle varie transaction family, rendendo a tutti gli effetti la blockchain "permissioned". Inizialmente si è pensato di sfruttare **sawtooth_identity**: una transaction family fornita dagli stessi sviluppatori del framework Sawtooth. Questa transaction family fornisce ai vari componenti di Sawtooth un modo per definire i permessi attraverso un sistema basato su ruoli e policy:

- **Policy:** descrive un insieme di identità che hanno il permesso o meno di eseguire determinate azioni. senza specificare l'azione stessa. Concretamente è una lista di coppie chiave pubblica-tipo di permesso (ovvero PERMIT_KEY o DENY_KEY).
- **Ruolo:** è semplicemente un riferimento ad una Policy, introdotto per motivi di flessibilità. Ad esempio, una policy può essere referenziata da più ruoli e in questo modo se la policy è aggiornata, tutti i ruoli che la referenziano solo a loro volta aggiornati.

Le "istanze" di queste due entità vengono memorizzate all'interno del namespace della transaction family nello stato globale della blockchain divenendo in questo modo delle impostazioni on-chain che sono distribuite su tutta la rete per una maggiore sicurezza.

Si potrebbe in questo senso definire ad esempio il ruolo "transactor" associandolo alla policy che controlla chi è autorizzato a inviare transazioni alla rete [29]. L'idea dell'utilizzo della transaction family sawtooth_identity è stato tuttavia scartata a causa di problemi in merito alla granularità dei permessi definibili. Infatti questa transaction family, pur nella sua discreta flessibilità, non permette di definire i permessi al livello delle singole operazioni di una specifica transaction family, capacità che invece era richiesta nei requisiti di questa applicazione. Inoltre sawtooth_identity non permette di gestire informazioni aggiuntive oltre a ruolo di un utente. Per questo motivo sull'esempio di sawtooth_identity, ma considerando un approccio leggermente diverso, è stata creata la transaction family utente. La struttura dello stato e del payload di questa transaction family è stata già descritto nel capitolo precedente: per ogni utente vengono memorizzati almeno id, ruolo, e chiave pubblica ad un indirizzo nel sottospazio di generato dalla chiave pubblica stessa. Ciò permette mantenere la corrispondenza tra una identità di un utente e relativa chiave pubblica. A questo punto le altre transaction family possono definire le regole/permessi per l'esecuzione delle loro operazioni sulla base degli attributi presenti tra le informazioni delle identità memorizzate dalla transaction family utente. Ad esempio il transaction processor della transaction family proposta_cessione deve garantire che una nuova proposta sia creata da un utente il cui suo ruolo sia quello di cedente e che l'id del cedente che sta creando la nuova transazione sia quello riportato nel payload della transazione. Per fare ciò nel momento in cui riceve nell' handler la transazione che genera la nuova proposta:

- Estrae dall'header la chiave pubblica dell'utente che ha firmato la transazione.
- Dalla chiave pubblica deriva l'indirizzo dello stato nel sottospazio della transaction family utente in cui si trovano le informazioni dell'utente associato alla quella chiave
- Accede alle informazioni dell'utente presenti all'indirizzo derivato.

- Controlla che il ruolo dell'utente sia "Cedente" e che l'id dell'utente corrisponda a quello presente nel campo id_cedente del payload della transazione che riporta le informazioni della nuova proposta di cessione
- Se i controlli vanno a buon fine esegue la creazione della proposta aggiornando lo stato, altrimenti dichiara la transazione non valida.

Sviluppi Futuri e Conclusioni

Sviluppi Futuri

Rilascio in produzione della rete blockchain

Allo stato attuale si può affermare che l'applicazione si trova in una fase di sviluppo definibile come "**parziale beta**": l'applicazione client (in cui includiamo front end back end e database) è stata correttamente integrata come servizio nella piattaforma preesistente e viene utilizzata da clienti della piattaforma reali nel server di produzione; la parte relativa alla blockchain è ancora in una fase di test in un singolo nodo locale e non è stata ancora ufficialmente rilasciata. Il principale sviluppo futuro è quindi cercare di raggiungere questo obiettivo: il rilascio in produzione della rete blockchain. Per fare ciò è necessario risolvere però alcune problematiche:

- **Scelta delle macchine che ospitano i nodi:** A differenza di altre tecnologie, per via della sua natura distribuita e decentralizzata, la blockchain deve essere rilasciata su una rete di nodi. Come si è già visto nei capitoli precedente Hyperledger consente, in modo molto flessibile, di gestire i vari componenti dei nodi attraverso container docker lasciando spazio a diverse soluzioni per la "topologia fisica" della rete. In ogni caso i container devono comunque essere eseguiti su macchine fisiche o virtuali. Ora, sebbene grazie a docker sia possibile ospitare tutti i nodi sulla stessa macchina, questa opzione è altamente sconsigliata in quanto si andrebbero a perdere tutti i vantaggi nell'utilizzo della blockchain dal momento che la macchina costituirebbe un single point of failure in caso di guasti o malfunzionamenti. Appare evidente come un requisito nella costruzione della rete sia quello di mantenere i nodi fisicamente, o meglio ancora, geograficamente distanti. Nella maggior parte dei casi (aziende che non possiedono diverse sedi/macchine distribuite in un spazio geografico ampio) questo implica affidarsi a un provider di **servizi cloud** come ad esempio Google Cloud, Amazon AWS o Microsoft Azure che in alcuni casi forniscono già delle soluzioni preconfigurate per le blockchain (e per Sawtooth in particolare)[30]. Chiaramente questi servizi hanno un costo di cui deve essere tenuto conto nel budget del progetto.
- **Scelta del numero di nodi:** Altra questione da considerare nella scelta della rete blockchain è il numero di nodi che costituiscono la rete. Questo numero è da stabilire cercando di ottenere un compromesso tra l'esigenza di integrità e incorruttibilità dei dati memorizzati che si ottiene tramite la ridondanza (quindi con un numero di nodi più elevato possibile) e il costo complessivo della rete a cui si accenna nel punto precedente e che naturalmente diminuisce al diminuire del numero dei nodi della rete. Infine, riguardo al numero di nodi, c'è da considerare un ultimo aspetto influenzato dall'algoritmo di consenso scelto, il Practical Byzantine Fault Tolerance. Il PBFT garantisce la byzantine fault tolerance a patto che no più di $\frac{1}{3}$ dei nodi della rete siano "corrotti", questo implica che per il PBFT il numero minimo di nodi per avere Byzantine Fault Tolerance è 4.
- **Sicurezza dei nodi:** Nel capitolo relativo alla sicurezza sia stato spiegato come per motivi di trasparenza le informazioni memorizzate non contengono dati sensibili ma solo le informazioni tali da consentire delle verifiche a partire dai dati completi memorizzati nel database principale. Nonostante ciò è comunque necessario proteggere i nodi da attacchi che mirano alla "distruzione" di queste informazioni o a causare disservizi alla rete. In particolar modo bisogna accertarsi che le comunicazioni con l'esterno siano solo verso gli

altri nodi e sicure. Le soluzioni in cloud, a cui si è accennato precedentemente, in questo senso possono aiutare fornendo dei layer di sicurezza inclusi nei piani tariffari.

Miglioramento sicurezza recupero chiave

Nel capitolo relativo alla sicurezza è stato mostrato il sistema che permette di recuperare la chiave privata quando l'utente ha smarrito la passphrase con cui viene cifrata. La password viene costruita a partire dalle risposte alle domande di sicurezza standard e da altri dati personali dell'utente presenti nel database. Sebbene la procedura sia accessibile solo all'utente che è già autenticato, le risposte alle domande di sicurezza, scelte per la loro relativa facilità di implementazione, rimangono un anello debole nella sicurezza delle chiavi. Un miglioramento che può essere apportato è quello di aumentare la variabilità delle domande di sicurezza o di far decidere direttamente all'utente le domande di cui fornire le risposte. Lo step successivo sia per la password di recupero che per quella principale potrebbe poi essere sostituire completamente questo secondo fattore di autenticazione con un meccanismo più raffinato e ampiamente utilizzato quale ad esempio le **One Time Password**. Le OTP sono password "usa e getta", in quanto valide per un'unica transazione, che possono essere generate quando necessario attraverso **app di autenticazione**, appositi token fisici o inviate via SMS al numero di telefono fornito dall'utente (anche se questi ultimi due metodi sono ormai sconsigliati per motivi di sicurezza)[31].

Maggiore automatizzazione dei controlli

Un altro miglioramento che si potrebbe introdurre come sviluppo futuro riguarda i controlli circa la validità delle proposte di cessione e delle richieste di accreditamento. Allo stato attuale prima che una proposta o una richiesta vengano accettate/confermate è necessario che l'acquirente stesso si accerti che sia tutto in regola. Questo, come si può immaginare, rallenta il processo di cessione del credito. L'idea a questo sarebbe quella di automatizzare il più possibile alcune verifiche in modo da sollevare l'acquirente da questo onere e al tempo stesso garantirgli che tutti i requisiti per la cessione o l'accredito sono soddisfatti.

Gestione degli altri servizi attraverso la blockchain

Attualmente gli smart contract sviluppati permettono di gestire unicamente proposte di cessione e richieste di accreditamento, ovvero le due entità fondamentali del servizio di cessione del credito. Il servizio di cessione del credito è tuttavia uno dei tanti presenti sulla piattaforma in cui è stato integrato. Una volta dimostrata l'efficacia della blockchain relativamente a questo servizio, l'idea sarebbe di sfruttarla anche per alcuni dei dati degli altri servizi della piattaforma, in particolare quelli direttamente collegati alle cessioni, come ad esempio quello per la creazioni e gestione pratiche (a partire dalle quali si creano le proposte di cessione). Inoltre questa integrazione della blockchain è facilitata dalla riusabilità della transaction family utente da parte di nuove transaction family nell'autenticazione e autorizzazione degli utenti della piattaforma. Un possibile sviluppo futuro consisterebbe quindi nell'affiancare la blockchain all'intera piattaforma e non semplicemente al solo servizio di cessione del credito.

Gestione di più tipologie di bonus

Il Superbonus 110 è solo uno dei diversi bonus statali che prevede detrazioni fiscali con la possibilità di cessione del credito di imposta. Visto il successo di questo bonus la piattaforma si sta preparando per poter gestire anche altri tipi di bonus. Grazie alla struttura modulare dei servizi e in particolar modo di cessione del credito, è possibile espandere le funzionalità fornite proposta semplicemente associando la proposta ad una nuova tipologia di pratica (creata nel servizio di gestione delle pratiche bonus). In considerazione di questa possibilità il workflow della proposta di

cessione (che va dalla creazione fino all'acquisto) è stato pensato per essere generico e flessibile, in modo da potersi adattare non solo alle specifiche esigenze (in termini di documentazione) dei singoli gruppi acquirente ma anche alla possibile introduzione appunto di nuove tipologie di pratiche.

Supporto alla "ricessione" del credito da parte dell'acquirente

Sebbene non sia una possibilità particolarmente sfruttata, il possessore di credito d'imposta ceduto precedentemente da un altro soggetto può a sua volta ricedere il credito a terzi. Nell'ottica dei ruoli gestiti dall'applicazione questo implica che un acquirente potrebbe a sua volta cedere il credito di una proposta che ha acquistato. Attualmente ciò non è possibile e pertanto una idea per un possibile sviluppo futuro è implementare questo meccanismo di **"ricessione"** del credito utilizzando in particolar modo la blockchain come mezzo per tracciare tutti i "passaggi di proprietà" del credito.

Conclusioni

Volendo fare delle considerazioni finali si può affermare che l'applicazione sia riuscita a raggiungere, anche se in maniera parziale, i due principali obiettivi di cui si è parlato nell'introduzione. Attualmente la componente client (front end, back end e database) dell'applicazione è integrata nella piattaforma e utilizzata dagli utenti finali. Contando su una base di utenti cedenti consolidata grazie agli altri servizi, con l'introduzione di cessione del alcuni utenti acquirenti e gruppi acquirenti hanno iniziato a registrarsi alla piattaforma e ad acquistare proposte di cessione avanzate dai cedenti. Riguardo alle aspettative in merito alla blockchain, sebbene la componente della rete non si ancora entrata nella fase di produzione, bisogna sottolineare come il processo di sviluppo della rete, e in particolare il framework Hyperledger Sawtooth, abbiano rivelato due cose molto importanti:

- I potenziali campi di applicazione che possono trarre beneficio da questa tecnologia sono innumerevoli anche se spesso vengono messi in secondo piano dalle criptovalute e dagli aspetti prettamente finanziari legati alla blockchain.
- La costruzione di una rete di una rete prevede accorgimenti e soprattutto investimenti particolari rispetto a un semplice DBMS di cui si deve tenere conto.

Bibliografia - Sitografia

[1] *Superbonus 110%, cos'è?* - [governo.it](https://www.governo.it)

<https://www.governo.it/it/articolo/superbonus-110-case-pi-efficienti-e-sicure-costo-zero/15948>

[2] *Cessione del Credito Superbonus: come si procede (passo per passo)* - [edilizia.com](https://www.edilizia.com)

<https://www.edilizia.com/guide/cessione-del-credito-superbonus-come-si-procede-passo-per-passo/>

[3] *Blockchain* - [wikipedia.org](https://it.wikipedia.org)

<https://it.wikipedia.org/wiki/Blockchain>

[4] *101 Smart Contracts and Decentralized Apps in Ethereum* - Pablo Cibraro - auth0.com

<https://auth0.com/blog/101-smart-contracts-and-decentralized-apps-in-ethereum/>

[5] *Public vs Private Blockchain: What's the difference?* - [e-zigurat.com](https://www.e-zigurat.com)

<https://www.e-zigurat.com/innovation-school/blog/public-vs-private-blockchain-whats-the-difference/>

[6] *Understanding Blockchain Consensus Models* - Dr. Arati Baliga

<https://www.persistent.com/wp-content/uploads/2017/04/WP-Understanding-Blockchain-Consensus-Models.pdf>

[7] *Proof of Work* - [wikipedia.org](https://en.wikipedia.org)

https://en.wikipedia.org/wiki/Proof_of_work

[8] *Proof of Work (PoW) vs Proof of Stake (PoS): la guida* - Stefano Cavalli - cryptonomist.ch

<https://cryptonomist.ch/2019/10/05/proof-of-work-pow-vs-proof-of-stake-pos-la-guida/>

[9] *Byzantine Fault* - [wikipedia.org](https://en.wikipedia.org)

https://en.wikipedia.org/wiki/Byzantine_fault

[10] *What is Practical Byzantine Fault Tolerance?* - Brian Curran - blockonomi.com

<https://blockonomi.com/practical-byzantine-fault-tolerance/>

[11] *Sawtooth Healthcare* - github.com

<https://github.com/hyperledger-labs/sawtooth-healthcare>

[12] *Non-fungible Token* - [wikipedia.org](https://it.wikipedia.org)

https://it.wikipedia.org/wiki/Non-fungible_token

[13] *Hyperledger Fabric – Top Use Cases* - Toshendra Kumar Sharma - [blockchain-council.org](https://www.blockchain-council.org)

<https://www.blockchain-council.org/blockchain/hyperledger-fabric-top-use-cases>

[14] *Perché il real estate ha molto da guadagnare dalla blockchain* - [01building.it](https://www.01building.it)

<https://www.01building.it/strumenti/real-estate-molto-guadagnare-blockchain/>

- [15] *Classification and importance of nodes in a blockchain network* - Yves Longchamp, Saurabh Deshpande, Ujjwal Mehra - seba.swiss
<https://www.seba.swiss/research/Classification-and-importance-of-nodes-in-a-blockchain-network>
- [16] *Smart Contract* - wikipedia.org
https://it.wikipedia.org/wiki/Smart_contract
- [17] *Sawtooth Architecture Guide* - sawtooth.hyperledger.org
<https://sawtooth.hyperledger.org/docs/core/releases/latest/architecture.html>
- [18] *Distinctive Features of Sawtooth* - sawtooth.hyperledger.org
<https://sawtooth.hyperledger.org/docs/core/releases/latest/introduction.html#distinctive-features-of-sawtooth>
- [19] *Gossip Protocol* - wikipedia.org
https://en.wikipedia.org/wiki/Gossip_protocol
- [20] *Transaction Family Overview* - sawtooth.hyperledger.org
https://sawtooth.hyperledger.org/docs/core/releases/latest/app_developers_guide/overview.html
- [21] *Albero di Merkle* - wikipedia.org
https://it.wikipedia.org/wiki/Albero_di_Merkle
- [22] *Blockchain-based Database* - wikipedia.org
https://en.wikipedia.org/wiki/Blockchain-based_database
- [23] *Digital Signature* - wikipedia.org
https://en.wikipedia.org/wiki/Digital_signature
- [24] *Elliptic-curve Cryptography* - wikipedia.org
https://en.wikipedia.org/wiki/Elliptic-curve_cryptography
- [25] *An introduction to elliptic curve cryptography* - Johan Dams - embedded.com
<https://www.embedded.com/an-introduction-to-elliptic-curve-cryptography/>
- [26] *Attribute-based Encryption* - wikipedia.org
https://en.wikipedia.org/wiki/Attribute-based_encryption
- [27] *The OpenABI Library C++ API Guide* - Zeutro LLC
<https://github.com/zeutro/openabe/blob/master/docs/libopenabe-v1.0.0-api-doc.pdf>
- [28] *PKCS #8* - wikipedia.org
https://en.wikipedia.org/wiki/PKCS_8
- [29] *Identity Transaction Family* - sawtooth.hyperledger.org
https://sawtooth.hyperledger.org/docs/core/releases/latest/transaction_family_specifications/identity_transaction_family.html
- [30] *Blockchain-as-a-Service (BaaS)* - Jake Frankenfield
<https://www.investopedia.com/terms/b/blockchainasaservice-baas.asp>

[31] *One-time Password* - *wikipedia.org*
https://en.wikipedia.org/wiki/One-time_password