POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Blockchain e Smart-Contract per i servizi finanziari

Relatore

Prof.ssa Valentina GATTESCHI

Tutor aziendali

Ing. Cyril VIGNET

Ing. Jose LUU

Candidato

Sergio GIARDINA

Aprile 2022

Ringraziamenti

Vorrei dedicare qualche riga a coloro che hanno contribuito a farmi arrivare fin qui.

In primis, un ringraziamento speciale alla mia relatrice, la Professoressa Gatteschi Valentina, per la sua infinita disponibilità, la sua tempestività ad ogni mia richiesta dalla lontana Francia, l'immensa pazienza ed, infine,i suoi indispensabili consigli.

Al "Poli" con i suoi professori, studenti ed aule studio sempre piene che con tutti i suoi meravigliosi difetti si è rivelata una vera scuola di vita.

Alla mia famiglia, i miei zii, i miei amici, i miei compagni di università a Torino ed a Grenoble...

Finiti i ringraziamenti scontati, voglio dedicare un pensiero a tutti quelli che hanno incrociato la loro vita con la mia in questi anni, restando con me o andando via, ma lasciandomi qualcosa. Non sempre qualcosa di buono ma in ogni caso qualcosa che oramai fa parte di me, contribuendo a ciò che sono adesso.
Quindi vi ringrazio. Grazie per essere stati miei complici, ognuno a suo modo, in questa piccola grande parte della mia vita. E' stato intenso e travolgente, nel bene e nel male. Adesso sono così tanti i ricordi, i momenti, gli sguardi che mi passano per la testa, che è davvero difficile trovare le parole giuste per descriverli. Ma a farlo saranno le mie emozioni, i miei sorrisi e le mie lacrime mescolati in un bagaglio di nostalgia e gratitudine per tutti voi.
Grazie per aver condiviso anche solo un pezzettino della vostra strada insieme a me.

Sergio

Sommario

Da un paio d'anni a questa parte, la Blockchain ha lasciato il suo ruolo di tecnologia per pochi, aprendosi ad un pubblico sempre più ampio. Le nuove esigenze dell' "Industry 4.0" chiedono processi innovativi, prestazioni innovative e infrastrutture innovative. In quest'ottica l'elemento dirompente costituito dalla tecnologia blockchain ha dettato legge, inondando il mercato di nuovi modelli di produzione che si basano o sfruttano le caratteristiche dei più rinomati sistemi blockchain.

Nello specifico, l'utilizzo di questa tecnologia risulta essere particolarmente proficuo nel settore Fintech [1]. Infatti, a fronte di un aumento della complessità del rapporto tra Finanza e Tecnologia che la società moderna richiede, la trasparenza e la linearità del settore sono state terribilmente compromesse. A questo proposito, l'applicazione di sistemi basati su Blockchain si è rivelata fondamentale al fine di fornire una risposta valida ed efficente: mantenere la totale trasparenza delle operazioni finanziarie permettendo un uso piu chiaro degli strumenti finanziari da parte del consumatore, garantendo al contempo l'immutabilità dei registri e di conseguenza la totale sicurezza delle singole operazioni finanziarie.

Nel presente lavoro, si è voluto approfondire il ruolo e l'uso degli strumenti della Blockchain all'interno del progetto QAXH.IO (BPCE SA), che permette ad un utente/cliente di verificare la propria identità e di conseguenza eseguire alcune operazioni bancarie in pochi minuti. Tale progetto punta a rendere ad esempio l'apertura di un conto in banca, la sottoscrizione di un particolare contratto con eventuali obblighi o qualsiasi altra attività, che richieda un processo amministrativo, molto più agile e snella. La piattaforma Qaxh.io funge quindi da collegamento tra un fornitore di servizi (banca, azienda, istituzione..) e una persona fisica o una società. Per rispettare questo principale obiettivo progettuale, in particolare è stata studiata e analizzata una soluzione per dar vita ad un sistema di gestione dei poteri bancari che raccolga gli ordini finanziari emessi da un'entità fisica (persona o azienda) verso un'istituzione (banca ecc). Sono state studiate le varie relazioni tra i diversi attori coinvolti e il rispettivo flusso dati che si genera per poter progettare una soluzione scalabile lungo tutto il work-flow. In dettaglio, sono stati implementati sia degli smart contract per offrire un'interfaccia di permessi di creazione/firma per il cliente da un lato e un fornitore di ordini per l'istituzione

dall'altro, garantendo all'istituzione la firma e l'autorizzazione all'operazione senza tuttavia rivelare obbligatoriamente l'identità del firmatario. Al contempo, sono state sviluppate delle API che permettono ad un'istituzione di implementare in maniera customizzata uno smart contract di gestione dei permessi e diritti, e di verificare e tracciare agevolmente le interazioni di firma/creazione degli ordini del contratto da parte del singolo utente.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

Il presente lavoro è stato concepito con l'idea di approfondire il ruolo e l'uso degli strumenti della Blockchain all'interno di progetti di innovazione bancaria, cosi da permettere ad un utenti bancari di eseguire alcune operazioni in tal senso in maniera veloce ed intuitiva. Le attività descritte nei prossimi capitoli sono state svolte presso il team di Innovazione 89C3 del gruppo BPCE , seconda banca di Francia, in collaborazione con Natixis, fornitore di servizi in ambito fintech.

In questo capitolo introduttivo verrà dunque presentato il Gruppo BPCE (che controlla Natixis), verranno poi descritti brevemente gli obiettivi della piattaforma QAXH.IO, in terzo luogo verranno elencati brevemente i punti salienti del lavoro svolto in QAXH.IO ed infine verrà presentata una struttura generale dei capitoli seguenti. [2] [3]

1.1 Presentazione del Gruppo BPCE

Il gruppo *BPCE* nasce dalla fusione della *Caisse d'Économie* e della *Banque Populaire* nel 2008. Il gruppo BPCE non si concentra solamente sull'ambito prettamente bancario, ma investe in società finanziarie e tecnologiche come *Natixis*, acquistata nel 2009. Il gruppo BPCE offre una panorama completo di servizi finanziari e tecnologici, tra cui:

- Un'offerta bancaria su larga scala: con Natixis, che offre servizi fintech, di mercato e consulenza strategica.
- Un'offerta bancaria locale, soluzioni e competenze finanziarie nonché assicurazioni. Questi servizi sono garantiti in particolare attraverso la rete di casse di risparmio e cooperative, grazie alle banche Palatine e Oney. Quest'ultime forniscono servizi bancari e finanziari, consulenza, assicurazioni finanziarie e personali specializzate.

- Un'offerta di asset e wealth management attraverso 2 divisioni Natixis: Natixis Wealth Management e Natixis Investment Managers, che gestiscono oltre 800 miliardi di euro.

- Una gamma di soluzioni di pagamento con Natixis Payments. Natixis Payments è l'emittente leader carte Visa in Europa e gestisce più di 7 miliardi di transazioni all'anno.

1.2 Piattaforma *QAXH.IO*

Il progetto Qaxh.io è un progetto sviluppato congiuntamente da BPCE e Natixis attraverso Cyril VIGNET (BPCE) e José LUU (Natixis). Qaxh.io è un framework per l'ambiente bancario per gli utenti blockchain al dettaglio. Questo progetto si base sullo sviluppo di "un'applicazione che consenta l'utilizzo della blockchain da parte dei clienti bancari di ogni genere". Il progetto si focalizza in particolare sulla certificazione d'identità su un sistema blockchain e sull'implementazione di soluzioni che consentano ai clienti bancari di interagire con essa.

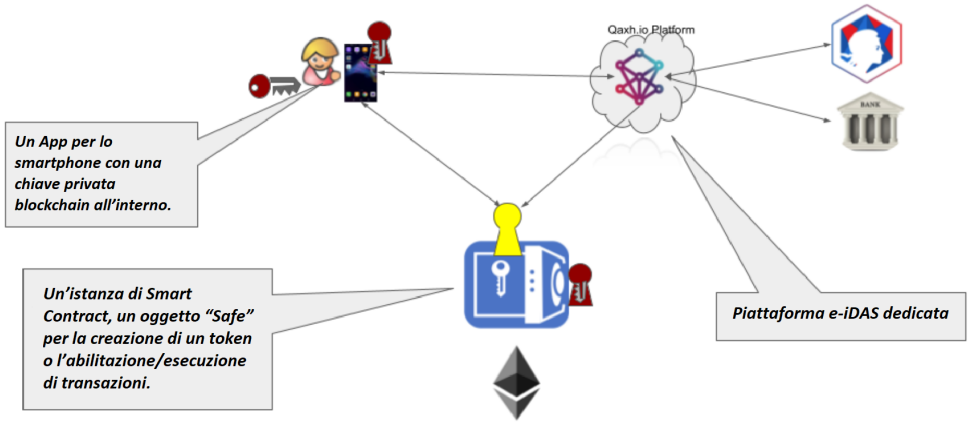

Figura 1.1: Piattaforma QAXH.IO

Il progetto Qaxh.io si articola in più fasi, proponendo diversi obiettivi, alcuni raggiunti ed altri ancora in fase di sviluppo:

1. Progettare un'applicazione (Android/Ios) che permetta la creazione di account di identità (autodichiarati) in grado di interagire con la blockchain di Ethereum. L'applicazione QAXH.IO, in questo caso, contiene la chiave privata di un

indirizzo Ethereum esterno. Il cliente può quindi consultare l'elenco delle transazioni che lo vedono coinvolto personalmente dall'app.

2. Certificare l'identità degli account con una terza parte fidata tramite la piattaforma France Connect. Si occupa quindi di stabilire un protocollo che permetta a tutti gli utenti di verificare che un indirizzo esterno sia effettivamente certificato dall'indirizzo Qaxh.io su Ethereum. Ciò garantisce il livello eIDAS mantenendo al contempo le identità personali segrete (regolamento GDPR).

3. Consentire agli utenti di ottenere una nuova chiave in caso di smarrimento della chiave privata senza rischi per la propria identità o i propri bene. Tutto questo è stato possibile grazie all'implementazione di uno speciale smart contract in grado di memorizzare le chiavi dei suoi possessori e aggiornarne la lista.

4. Aprire il pieno potenziale della tecnologia blockchain agli utenti introducendo una versione interoperabile moneta elettronica, ai sensi della DME2 (Direttiva del Parlamento Europeo sull'accesso alle attività degli istituti di moneta elettronica).

5. Consentire da un lato ad aziende e FinTech di offrire i propri servizi e dall'altro ai consumatori di accedervi con tutta la sicurezza che hanno diritto di aspettarsi da un servizio bancario: tutela del consumatore, rispetto delle normative Europee, sicurezza nell'operatività quotidiana.

6. Collegare i conti Qaxh.io con i conti bancari. A tal fine, devono essere osservati alcuni standard bancari, in particolare per quanto riguarda il riciclaggio di denaro.

In conclusione, il quadro è offerto sia in un ambiente "*business to business*" sia in un ambiente "*business to customer*". [4]

1.3 Gestione dei poteri bancari

Il lavoro svolto, descritto nei seguenti capitoli, si concentra sul miglioramento dell'offerta di servizi QAXH.IO attraverso la creazione di un sistema di gestione dei poteri bancari (*Habilitation Folder*) che raccolga gli ordini finanziari emessi da un'entità fisica (persona o azienda) ad uno stabilimento (banca, ecc.). Nel trading bancario, per ordine si intende l'istruzione, inviata al broker o immessa in piattaforma, di acquistare o vendere un determinato strumento finanziario. Questo sistema si basa sulla possibilità di creare da parte di un "Issuer" (utente che emette ordini) in collaborazione con un "Acquirer" (istituto che riceve ed elabora ordini, una banca ecc) un registro immutabile e condiviso utile a memorizzare gli

ordini finanziari e tutte le caratteristiche di quest'ultimi. Ogni sistema di gestione manterrà al suo interno:

- Un Issuer;
- Un Acquirer ;
- Un insieme di informazioni che descrivono l'Acquirer.
- Un elenco di utenti con determinati poteri (firma/creazione/distruzione).
- Un elenco di autorizzazioni alla transazione per ogni singolo utente (issuer).
- Un insieme di autorizzazioni che consentirà all'issuer e all'acquirer di disattivare l'intero sistema di abilitazione in qualsiasi momento.

L'elenco degli ordini verrà conservato in questo contesto bancario, ogni ordine avrà determinate caratteristiche:

- Manterrà al suo interno un elenco delle transazioni (indirizzo utente Ethereum, dati crittografati, Iban cifrato) che lo compongono.
- Avrà un "Merkle Header" per consentire la verifica delle autorizzazioni e dei dati quando dovrà essere firmato.

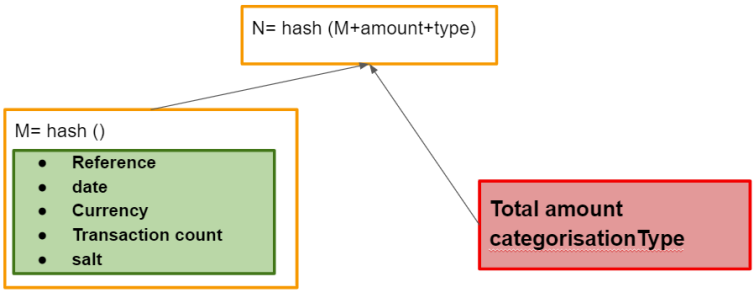

Figura 1.2: Merkle Header

- Memorizzerà una cronologia dei vari stati legati all'Issuer (Creazione / Firma / Registrazione).
- Manterrà una storia dei vari stati relativi all'Acquirer. (Creazione/Ricezione/-Ripresa carico/esecuzione).

Il sistema offrirà svariate possibilità di gestione al ruolo dell'issuer e al ruolo dell'acquirer :

Issuer: Gestione dei permessi del sistema e del lato "client" degli ordini.

Acquirer: Creazione del sistema di gestione dei poteri, con relativi dati all'interno, in maniera personalizzata e gestione di esso attraverso un sistema di controllo e di notifica dei cambiamenti.

Ultima ma non meno importante è la gestione della riservatezza che contraddistingue l'*HabilitationFolder*. Quest'ultima è garantita attraverso l'implementazione di moderni algoritmi crittografici simmetrici e non.

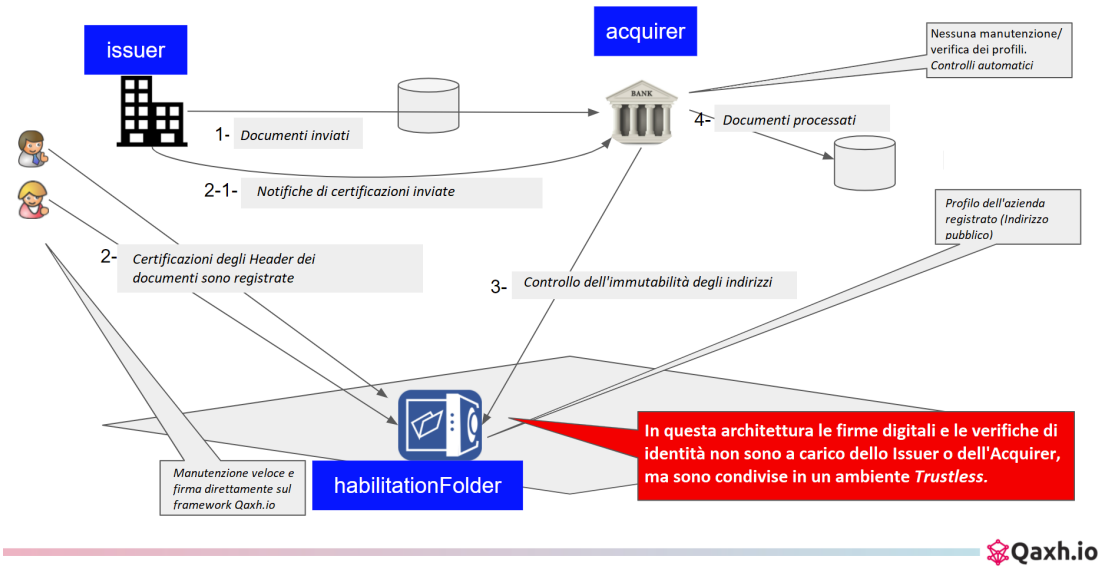

Figura 1.3: *Habilitation Folder*

L'efficienza e la validità di questo sistema da implementare in QAXH.io si basano su 3 concetti fondamentali nel campo del fintech:

1. *Riservatezza*: L'anonimato delle firme (spesso sinonimo di maggiore sicurezza) che, ad esempio, consente all'istituto bancario di sapere che un ordine è stato firmato, senza sapere realmente chi lo ha firmato e senza verificare se la firma sia valida o meno. (compito che in questo caso è riservato a smart contract e non alla struttura bancaria).

2. *Automatismo*: Lo Smart Contract incorpora clausole contrattuali standard e univoche, in modo che quando si verifichino determinate condizioni, quest'ultimo venga eseguito automaticamente. Stiamo parlando di una vera funzione "if / then" . Questo automatismo è ovviamente un punto di forza del settore bancario.

3. *Immutabilità* di Ethereum: Il modello blockchain di Ethreum impedisce qualsiasi forma di censura, nessuno è in grado di impedire che una transazione avvenga e venga aggiunta al libro mastro una volta ottenuto il consenso necessario tra tutti i nodi (partecipanti) della blockchain . I registri non autorizzati possono essere utilizzati come database globale per tutti i documenti che devono essere assolutamente immutabili nel tempo, a meno che gli aggiornamenti non richiedano la massima sicurezza in termini di consenso, caratteristica fondamentale dei servizi bancari.

In conclusione seguono le principali implementazioni svolte come oggetto di tesi:

1. Progettazione e ideazione delle regole dell'intero sistema di attori.
2. *Smart Contract* per la gestione del sistema di ordini direttamente su Blockchain Ethereum.
3. *Backend* per l'interazione con gli *Smart Contract*
4. *Frontend* per un utente mobile.
5. *APIs* per automatizzare alcune interazione con gli *Smart Contract* da parte di alcune istituzioni (Banca etc).

1.4 Struttura della tesi

Il resto della tesi è strutturato come segue:

- **Capitolo 2 - Stato dell'arte:** descriverà nella prima parte più nel dettaglio la tecnologia Blockchain, in particolare i suoi elementi base, il protocollo e le sue proprietà principali. Nella seconda parte verranno approfonditi i maggiori sistemi Blockchain attuali elecandone eventuali vantaggi e svantaggi di ognuna. Nella terza parte si entrerà nel dettaglio di alcune tecnologie come Ethereum e gli Smart contract. L'ultima parte del secondo capitolo racconterà le aree di studio e alcuni casi d'uso dei sistemi blockchain nell'ambito FinTech, cercando quindi di dare una panoramica complessiva dello stato dell'arte.
- **Capitolo 3 - Sviluppo dell'applicazione:** Il terzo capitolo verterà sullo sviluppo in dettaglio dell'intero sistema di gestione dei poteri bancari, approfondendo da svariati punti di vista, tecnici, finanziari e d'esperienza-utente, l'intera soluzione.
- **Capitolo 4 - Risultati:** Il quarto capitolo è destinato alla descrizione di alcuni test della soluzione e all'analisi dei conseguenti risultati.

- **Capitolo 5: Conclusioni:** Il capitolo conclusivo trarrà le conclusioni riguardanti la tesi svolta. Da ultimo, traccerà le linee dei possibili sviluppi futuri.

Capitolo 2

Stato dell'arte

2.1 Blockchain

Nell'ottobre del 2008 Satoshi Nakamoto pubblicò il paper scientifico "Bitcoin: A Peer-to-Peer Electronic Cash System" col quale spiegò i principali punti del protocollo Bitcoin e della corrispondente criptovaluta. Nel nucleo protocollare di questo sistema di pagamento elettronico, basato sulla criptografia e priva di terze parti fidate, vi è la tecnologia di un sistema Blockchain. [5]

2.1.1 Tecnologie

La Blockchain [6] è un insieme di registri diverso da ciò a cui potremmo essere abituati. Con il termine "Blockchain" si fa riferimento ad un insieme di oggetti e concetti in via di sviluppo con cui è necessario prendere confidenza per comprendere i complessi meccanismi di un tale ecosistema. Per questo motivo, questo capitolo si sofferma in questa prima sezione sugli elementi di questo sistema e in seguito su alcuni sviluppi particolari.

Lo struttura della Blockchain è composto dai seguenti elementi:

- **Ledger - Decentralizzazione**
  I sistemi blockchain funzionano secondo il principio della decentralizzazione. Questo database distribuito (ledger) è costituito da blocchi (verranno approfonditi più avanti) ed è visto dagli utenti come un registro globale e pubblico, nel quale vengono segnate tutte le transazioni in maniera immutabile, sequenziale e trasparente. Quest'ultime vengono verificate e archiviate stabilmente. La verifica è detta "Mining",ed è possibile grazie alla potenza di calcolo di più macchine in parallelo.

- **La figura del "Miner"**:
  Il "Miner" è colui che ha verificato le transazioni all'interno di un blocco, e per

tale ragione guadagna una certa quantità di criptovaluta (Ether su Ethereum, Bitcoin per l'omonima blockchain). L'estrazione viene quindi accettata dalle altre parti, fino a quando non arriva un altro blocco da verificare (un nuovo blocco viene estratto ogni 15 secondi su Ethereum).

- **I blocchi**
  Il blocco è una parte essenziale della catena, è il luogo in cui vengono archiviate tutte le informazioni, proprio come le pagine di un libro. Il blocco contiene informazioni relative al contesto: la persona che ha estratto questo blocco, la ricompensa ricevuta per l'estrazione, l'ora in cui è stata eseguita, ecc. Tuttavia, ciò che rende un sistema blockchain uno strumento così fruibile è la capacità di inserire qualsiasi tipo di informazione in un blocco. Come detto in precedenza, le informazioni e gli scambi di criptovaluta su Ethereum sono archiviati in questi blocchi, ma solamente in maniera indiretta. Questo perché il blocco memorizza le cosiddette transazioni e quindi non memorizza mai dati utente univoci (ad es. "L'utente X ha 5 ether il 23 marzo alle 20:00").

- **Hash**:
  La funzione di hash produce una sequenza di bit (o stringa), detta "digest", strettamente correlata con i dati in ingresso. Viene utilizzata, nel caso specifico delle transazioni della blockchain, ad esempio per identificare univocamente i blocchi di transazioni o per controllare un'eventuale modifica degli stessi. Aspetto fondamentale è l'impossibilità di invertire la funzione, cioè l'impossibilità di recuperare l'input iniziale dato l'output della funzione.

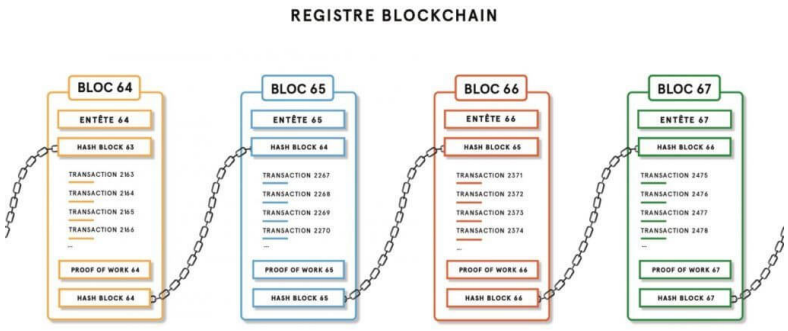

Figura 2.1: Registri blockchain

- **Le transazioni**:
  All'interno della Blockchain, alla base dell'archiviazione di informazioni, troviamo le transazioni in quanto trasportano questi dati che si sono accumulati

creando informazione sugli utenti e sulle loro operazioni. Quando avviene uno scambio sulla blockchain, viene creata una transazione. In quest'ultimo ci sono informazioni sul mittente, sul destinatario della transazione, i dati che sono stati scambiati e, se applicabile, il valore dello scambio (nella criptovaluta ether).

- **Meccanismo del "Consenso"**:
  Tutte le persone coinvolte nella blockchain devono essere d'accordo su tre punti cardine, al fine di generare un consenso distribuito. Devono essere d'accordo sulla storia delle transazioni, sulle regole intrinseche del protocollo e infine sul concetto di valore della moneta. L'ostacolo del consenso è descritto attraverso un grattacapo molto famoso in ambito informatico, detto il Problema dei Generali Bizantini. Quest'ultimo si basa sull'idea che quando la posta in gioco (beneficio personale) è troppo alta, l'incentivo a tradire l'intero esercito cresce, rendendo praticamente impossibile raggiungere un consenso univoco sul piano di battaglia. Allo stesso modo, quando parliamo di sistemi distribuiti informatici, raggiungere il consenso tra i vari nodi del network è molto complesso. A tal proposito nei maggiori sistemi blockchain vengono applicati alcuni dei seguenti sistemi di consenso [7]:

  - **Byzantine fault tolerance - BFT:**
    Il concetto di tolleranza al problema bizantino in Blockchain è la caratteristica che permette di raggiungere un accordo o un consenso su blocchi basati sulla prova di lavoro, anche quando alcuni nodi stanno fornendo valori fallaci per fuorviare la rete. L'obiettivo principale di BFT è quindi quello di salvaguardare il sistema e assicurarne il buon funzionamento anche quando alcuni nodi sono difettosi. [8]
  - **Proof of work - PoW:**
    Una prova di lavoro è un dato difficile da produrre ma facile da verificare per gli altri nodi e che soddisfa determinati requisiti. In altre parole, non si tratta che di risolvere un problema matematico (una ricerca del HASH adatto) spendendo molta potenza di calcolo al fine di convalidare le transazioni e creare nuovi blocchi. [9]
  - **Proof of stake - PoS:**
    Utilizza un processo di elezione pseudo-casuale per selezionare un nodo come validatore del blocco successivo, in base a una combinazione di fattori che potrebbero includere l'età dello staking, la randomizzazione e la ricchezza del nodo. La ricchezza del nodo determina le possibilità che un nodo venga selezionato come prossimo validatore per creare il blocco successivo (il sistema Proof-of-Stake di solito utilizza le commissioni di transazione come ricompensa). [10]

– **Proof of stake delegated - PoSd:** Utilizza un sistema di voto in cui le parti interessate delegano il tutto a una terza parte. In altre parole, possono votare dei delegati che proteggano la rete al posto loro. I delegati vengono chiamati anche testimoni e hanno il compito di raggiungere il consenso durante la generazione e la validazione di nuovi blocchi. Il potere di voto è proporzionale al numero di monete in possesso a ciascun utente.

Figura 2.2: PoW - PoS

2.1.2 Work-flow Blockchain

La blockchain permette la creazione e gestione di un grande database distribuito per la gestione di transazioni condivisibili tra più peer di una rete. Si tratta di un database strutturato in Blocchi (contenenti più transazioni) che sono tra loro collegati (catena di Hash) in rete in modo che ogni transazione avviata sulla rete debba essere validata dalla rete stessa nell'"analisi" di ciascun singolo blocco. La blockchain risulta così costituita da una catena di blocchi che contengono più

transazioni ciascuno. La validazione per tutte le transazioni è affidata ai nodi che sono chiamati a vedere, controllare e approvare (Consenso) tutte le transazioni creando una rete che condivide su ciascun nodo l'archivio di tutta la blockchain e dunque di tutti i blocchi con tutte le transazioni. Ciascun blocco è per l'appunto anche un archivio "abbreviato" di tutte le transazioni e di tutto lo storico di ciascuna transazione che, possono essere modificate solo con l'approvazione dei nodi della rete. Le transazioni possono essere considerate immodificabili (se non attraverso la riproposizione e la "ri"-autorizzazione delle stesse da parte di tutta la rete). Da qui il concetto di *immutabilità*.

2.1.3 Attacco del 51%

La garanzia della sicurezza è il fondamento di qualsiasi sistema blockchain. Questo perché la manomissione di un elemento della catena (cioè un blocco) comporterebbe la modifica di tutti i blocchi che potrebbero essere estratti da esso, poiché ogni blocco è collegato al successivo. L'attaccante (è facile pensare a qualcuno che vuole far accettare al sistema una transazione falsa , ad esempio un' auto-attribuzione di criptomoneta) avrebbe quindi bisogno di una potenza di calcolo maggiore di almeno la metà dei miner affidabili. Questo limite è chiamato soglia del 51 %, da questo limite deriva il possibile attacco del 51 %. Con quest'ultimo attacco, gli aggressori sarebbero in grado di impedire nuove transazioni e ottenere conferme, che permetterebbero loro di bloccare i pagamenti tra alcuni o tutti gli utenti. Sarebbero anche in grado di invertire le transazioni effettuate mentre controllavano la rete, il che significa che potrebbero spendere il doppio dello stesso importo. Un attacco del

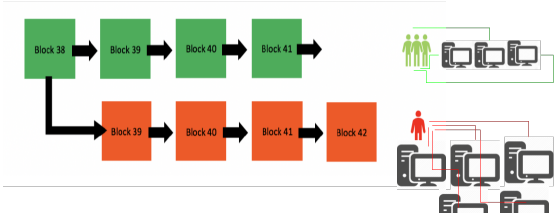

Figura 2.3: Attacco del 51%: Attaccante in rosso

51% probabilmente non distruggerebbe ether/bitcoin o qualche altra valuta basata su blockchain, anche se si dimostrerebbe molto dannoso. In teoria questo noto attacco è realizzabile ma il numero di "Miner" affidabili è molto alto (soprattutto per le blockchain più conosciute come Ethereum), quindi è improbabile che una sola persona possa raggiungere veramente la potenza di calcolo necessaria.

2.1.4 Trilemma

Il trilemma della blockchain è una condizione che riguarda i tre principi fondamentali della tecnologia ossia sicurezza, scalabilità e decentralizzazione. Il trilemma sostiene che tutte le blockchain possono risolvere solo due dei problemi appena citati. L'esempio dell'evoluzione di Ethereum è emblematico: il prossimo passaggio dal PoW al PoS tende a rendere il sistema meno sicuro al fine di una maggiore scalabilità e decentralizzazione nel mining.

	PoW	**PoS**
Energia consumata	Alta	Bassa
Strumenti necessari	Attrezzature costose	Nessuna attrezzatura
Sicurezza	Alta	Dipende dall'attacco (vulnerabile all'attacco del 51%)
Decentralizzazione vs Centralizzazione	Tende alla centralizzazione	Ogni utente mantiene i propri token
Transazioni per secondo	7-10	30-40

Tabella 2.1: Tabella tra PoW e PoS

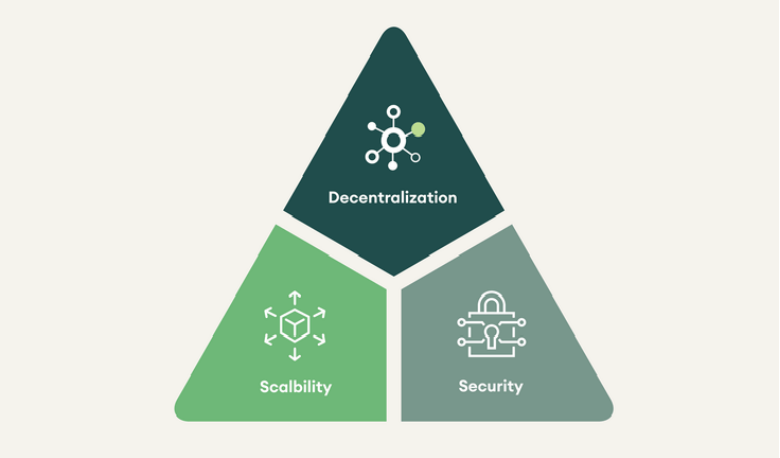

Figura 2.4: Trilemma Blockchain

2.2 La governance della Blockchain

Nonostante le numerose differenze, tutte i sistemi Blockchain possiedono alcune caratteristiche comuni:

- Sono reti peer-to-peer decentralizzate, nelle quali tutti i nodi della rete mantengono una copia del registro principale sul proprio apparecchio.
- Tutto è tenuto constantemente aggiornato (ogni copia per ogni nodo è aggiornata) grazie al protocollo del consenso.

L'obiettivo della classificazione che segue è identificare le differenze non solo ad alto livello ma anche a un livello di applicazione. Le distinzioni che si possono fare sono legate alle caratteristiche della rete e della presenza o assenza di permessi di accesso ad essa. Le principali differenze fra Blockchain, allontanandosi dall'aspetto puramente tecnico sono rilevabili a livello del ruolo dei vari nodi :

1. Chi può leggere tutti i record presenti sulla Blockchain?
2. Chi può scrivere su di essa?
3. Chi si occupa di mantenere la stabilità e l'integrità della rete, in altre parole chi ha il ruolo di "Miner"?

2.2.1 Classificazione delle Blockchain

Una prima attuale classificazione delle Blockchain è la seguente: pubbliche, permissioned e private. Tuttavia è opportuno sottolineare come elementi caratterizzanti di queste tre tipologie possono essere combinati facilmente in un'ampia varietà di modalità, al fine di creare registri personalizzati per tutte le tipologie di applicazione. [11]

- **Pubblica** o **Permissionless** : Questa tipologia di blockchain è definita in tal modo in quanto i dati sono condivisi con tutta la rete ed è possibile accedere e disporre di una copia di ogni singola transazione avvenuta nella rete grazie al sistema di consenso totalmente distribuito. L'impossibilità di vietare l'inserimento di una transazione permette ai suoi partecipanti di non dover sottostare al controllo di un ente centrale.
- **Permissioned** : E' definita così per il concetto di permesso che caratterizza il funzionamento. Se si vuole inserire un dato all'interno del libro mastro è sufficiente che i nodi definiti "trusted" approvino l'operazione. E' possibile fornire un esempio immaginando un consorzio di 20 istituzioni, ognuna delle quali gestisce un nodo della rete e 15 di esse devono firmare ogni blocco affinché il blocco sia valido. Il diritto di leggere la blockchain può essere pubblico o limitato ai partecipanti. Da questi modelli nascono anche sistemi ibridi che permettono delle API riservate agli utenti pubblici, per le verifiche dei blocchi, e delle API riservate agli utenti del consorzio. Queste tipologie sono note anche con il nome di *Blockchain del consorzio*.

- **Privata**: Le Blockchain private condividono molte caratteristiche con quelle del consorzio. Si tratta di reti private e non visibili, che sacrificano decentralizzazione, sicurezza e immutabilità in cambio di spazio di archiviazione, velocità di esecuzione e riduzione dei costi. Questo sistema viene solitamente controllato da un'organizzazione centrale, ritenuta altamente attendibile dagli utenti, che determina i permessi di accesso e scrittura nella blockchain. La governance di quest'ultima tipologia limita il concetto stesso di blockchain demandando un controllo oligarchico.

	Pubblica	**Privata/Consorzio**
Nuovi nodi	Chiunque può far parte della rete	Solo alcuni elementi pre-approvati
Creazione di una Transazione	Chiunque può creare una transazione	Solo i membri
Velocità di una Transazione	Bassa	Alta
Costo di una Transazione	Alto	Basso
Consenso	PoW,PoS,PoSd	PBFT, Proof of Authority
Fiducia	Tende alla centralizzazione	Ogni utente mantiene i propri token
Identità	Anonima Pseudoanonima	Conosciuta
Asset[1]	Nativo	Non presente
USP[2]	Decentralizzazione	Taglio dei costi

Tabella 2.2: Tabella di confronto tra Blockchain pubblica e privata

Da vari studi è emerso, inoltre, che ben l'80% dei progetti in via di realizzazione si distinguono per scopi privati, il che significa che cresce l'esigenza di migliorare le Blockchain del Consorzio in quanto permettono un meccanismo di adesione adatto per casi d'uso aziendali, garantiscono la confidenzialità delle transazioni e di conseguenza la privacy del processo e dei dati aziendali. D'altra parte, in alcuni processi è utile implementare soluzioni blockchain pubbliche, quindi è evidente

[1]Rappresenta un tipo di risorsa digitale o criptovaluta. Alcuni rappresentano partecipazioni in una particolare azienda. Altri sono pensati per essere valute, come Ether ad esempio.

[2]Caratteristica vincente di un prodotto.

una crescente necessità di migliorare l'interoperabilità tra Blockchain consorzio e pubbliche al fine di dar vita a nuove soluzioni cross-chain.

2.2.2 Limiti & Potenzialità

Chiarificata la natura del sistema Blockchain è possibile evidenziarne allo stesso tempo vantaggi e svantaggi [12]:

Pro	**Contro**
Decentralizzazione	Riduzione delle performance
Dati di alta qualità	Verifica della firma molto complessa
Persistenza e Sicurezza	Rischio di perdita delle chiavi private
Alto livello di Integrita	Problemi di integrazione
Longevità e Affidabilità	Consumo di energia non indifferente
Immutabilità e Transparenza	Regolamentazione non ancora definita
Sistema lineare	Poche possibilità di controllo per le aziende
Transazioni veloci	Problemi di confidenzialità e Privacy
Bassi costi per le transazioni	Mancanza di esperti nel campo
Tracciabilità affidabile	alti costi strutturali
Nuovo Business Model	Distruzione culturale dei sistemi classici

Tabella 2.3: Tabella Pro/Contro sistemi Blockchain

2.3 Blockchain 2.0

Tra i sistemi Blockchain più conosciuti e sviluppati negli ultimi anni si posiziona Ethereum. Quest'ultima, nata nel 2015, è la prima Blockchain programmabile al mondo (ha anche una propria criptovaluta nativa chiamata ether (ETH)). In pratica, Ethereum è solo una grande macchina virtuale decentralizzata:

- "Virtuale" perché è un computer che risiede in rete, composto da tutti gli altri computer presenti nella rete Ethereum.
- "Decentralizzato" perché nessuno può attaccarlo, censurarlo o controllarlo.

Oltre alle classiche possibilità offerte da qualsiasi tipo di sistema Blockchain, Ethereum permette di sviluppare e gestire degli "Smart Contracts" dando vita facilmente ad un particolare tipo di applicazioni detto "DApp". Queste applicazioni decentralizzate sfruttando i vantaggi delle criptovalute e della tecnologia blockchain, si propongono come delle vere e proprie applicazioni customizzabili per l'occasione. Punto di forza evidente di quest'ultime è, senza dubbio, l'affidabilità in quanto una volta "scritti" su Ethereum, funzioneranno in maniera inequivocabile come un classico codice programmato. Hanno la capacità inoltre di controllare le risorse digitali per creare autonomamente nuovi tipi di app finanziarie.

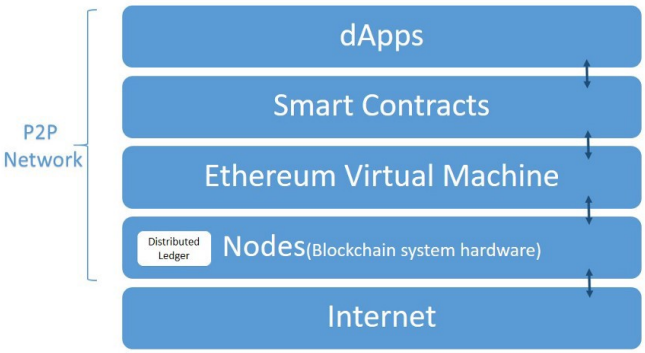

Figura 2.5: Blockchain 2.0

2.3.1 Ethereum Virtual Machine

Ethereum rappresenta un primo sistema Blockchain basato interamente su una EVM (Ethereum Virtual Machine). L'EVM è il primo sottostrato del sistema Ethereum. Si tratta di uno stack virtuale sandbox integrato in ogni nodo Ethereum completo, responsabile dell'esecuzione del bytecode del contratto. I contratti sono in genere scritti in linguaggi di livello superiore, come Solidity, quindi compilati in bytecode EVM. Ciò significa che il codice macchina è completamente isolato dalla rete, dal filesystem o da qualsiasi processo del computer host. Ogni nodo di Ethereum esegue un'istanza EVM che consente loro di concordare l'esecuzione delle stesse istruzioni. L'EVM è Turing equivalente, che si riferisce a un sistema in grado di eseguire qualsiasi passaggio logico di una funzione computazionale. [13]

Le macchine virtuali Ethereum sono state implementate con successo in vari linguaggi di programmazione tra cui C++, Java, JavaScript, Python ecc. L'EVM è essenziale per il protocollo Ethereum ed è utilizzato all'interno del motore di consenso del sistema Ethereum. Consente quindi a qualsiasi utente di eseguire

codice in un ecosistema trustless in cui l'esito di un'esecuzione può essere garantito ed è completamente deterministico.

Per ogni istruzione implementata sull'EVM, il sistema tiene traccia del costo di esecuzione e assegna all'istruzione un costo associato in unità di Gas. Quando un utente vuole avviare un'esecuzione, riserva dell'Ether, che è disposto a pagare per questo costo del gas.Il meccanismo del gas permette di assicurare due punti fondamentali per il corretto funzionamento del EVM : ci sarà sempre un validatore a garantire la transazione anche se fallisce. L'esecuzione dei contratti sarà sempre limitata dal gas impegnato (ciò eviterà qualsiasi tentativo di loop senza fine). Quando una transazione viene inviata alla rete, i validatori si occupano della transazione, eseguendo il codice associato e facendo i dovuti controlli sulle informazioni degli utenti, sui fondi impiegati e sulla validità del codice.

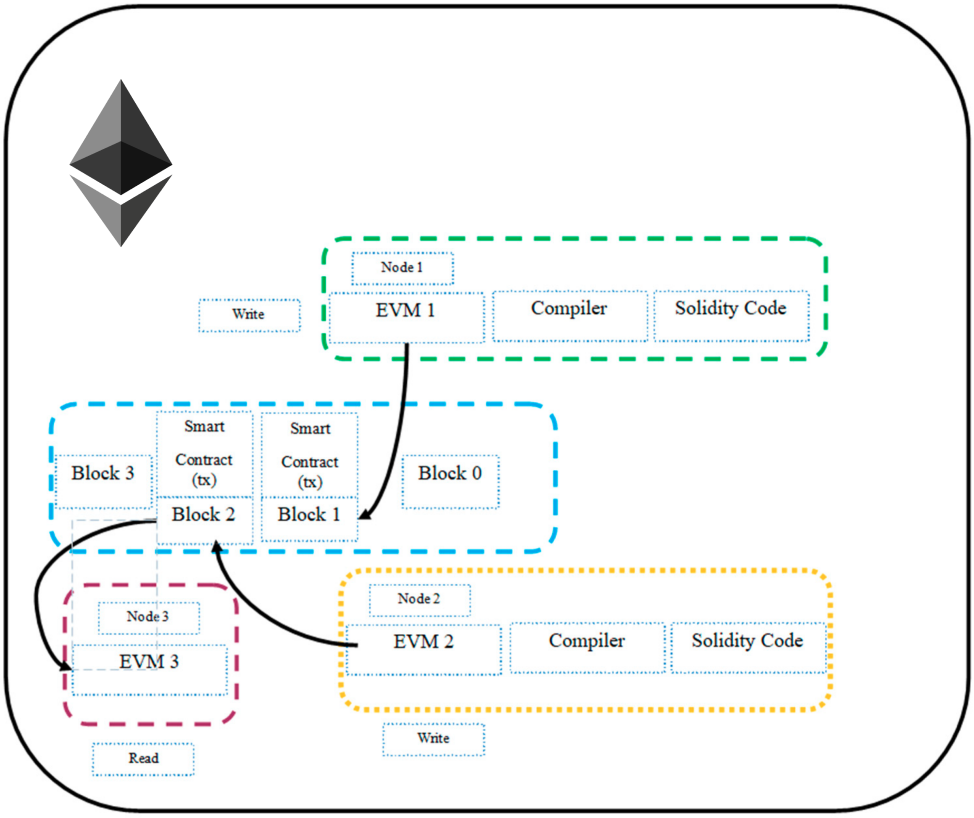

Figura 2.6: Ethereum Blockchain

Questo stack di Ethereum crea un'economia che si addebita per istruzione macchina (software) eseguita anziché per transazione finanziaria eseguita come fanno altre Blockchain. Si sposta l'idea della commissione dalla transazione all'esecuzione del codice. Il fatto che questo strato sia Turing completo rende Ethereum tecnicamente un super-computer peer-to-peer su scala mondiale, rendendolo persino in grado di assumere le funzioni di Internet come lo conosciamo. Ethereum potrebbe consentire quindi di creare economie di condivisione di file, eventi di crowdfunding peer-to-peer, contratti intelligenti, mercati per affittare lo spazio inutilizzato del disco rigido sul tuo laptop, un Uber o Facebook senza un'organizzazione alle spalle, ecc.

2.3.2 Smart-Contract

I *Smart Contract*, o contratti intelligenti, sono programmi informatici irrevocabili, il più delle volte distribuiti su una blockchain, che eseguono una serie di istruzioni predefinite. Il nucleo concettuale alla base dei smart contract è quello di garantire la forza vincolante dei contratti non più per legge, ma direttamente per codice informatico: "code is law", per citare la famosa frase di Lawrence Lessig. Gli Smart Contract sono veri e propri algoritmi posti su Ethereum. Questi contratti inoltre possono essere richiamati dall'esterno della blockchain permettendo una reale interazione con essi. Tuttavia, come tutto ciò che sta sulla blockchain, anche il contenuto di questi contratti è immutabile. Questi algoritmi si distinguono per la loro natura facilmente accessibile e fruibile da parte di qualunque utente della rete. In breve, lo smart contract offre un vero equivalente informatico (IT) del contratto cartaceo, eseguendo il codice-contratto, registrando e validando tutti i passaggi nella blockchain utilizzata. Questo processo quindi protegge tutti i dati del contratto impedendone la modifica o la cancellazione a posteriori. [14] [15]

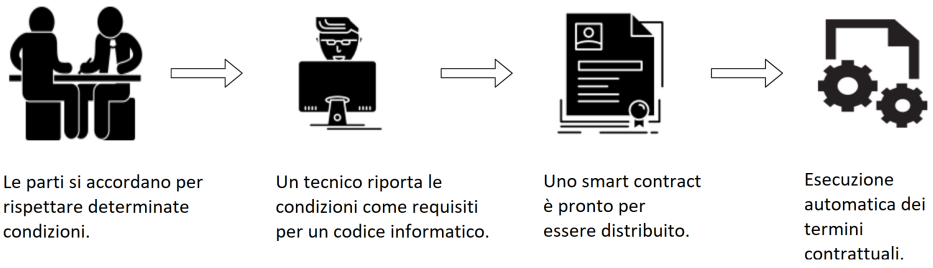

Figura 2.7: Work-Flow Smart Contract

Gli *Smart Contract*, redatti in *Solidity* (*Javascript Oriented*), sono per lo più standardizzati, e allo stesso modo dei contratti cartacei, si ha la possibilità di utilizzare dei modelli pre-esistenti. Lo standard di Smart Contract più diffuso è

quello dell'ERC-20 ed è utilizzato principalmente per la creazione e gestione di token (asset digitali, criptovalute) sulla rete Ethereum.

Tra i benefici più noti degli Smart Contract si ha:

1. *Flessibilità*: Questa può essere alla base di un accordo più ampio: implica che le parti che concludono accordi off-chain, che si trovano al di fuori di una blockchain, abbiano la possibilità di formalizzare in tutto o in parte le fasi successive utilizzando uno smart contract.

2. *Concettualmente inequivocabile* È adatto per costruire l'intera struttura degli accordi tra le parti.

3. *Completezza* È in grado di imporre una doppia condizione nei confronti di un accordo tra soggetti in quanto da un lato, si occuperà della formalizzazione dell'accordo e, dall'altro, ne permetterà l'esecuzione degli accordi, imponendola ai soggetti coinvolti.

2.3.3 DApp

Si tratta di una tipologia di applicazioni "Decentralized Applications", [16] il cui funzionamento non dipende da centri di controllo o da server centrali, bensì, si basa su una rete decentralizzata. Una rete nella quale gli utenti hanno il pieno controllo della rete stessa. Le DApp non sono una novità. Quest'ultime sono state già viste nei protocolli di condivisione file come BitTorrent o DC ++. Entrambe le applicazioni sono sistemi di condivisione di file peer-to-peer con elevata resistenza alla censura. Tuttavia è solo grazie ad Ethereum, il suo linguaggio Solidity e la possibilità di implementare dei smart contract se le DApp, in esecuzione su blockchain, hanno iniziato ad acquisire molta popolarità. Quest'ultima può portare presto all'adozione massiva della tecnologia blockchain, consentendo nuove forme d'interazione tra gli utenti e tra il mondo reale e quello virtuale. In DApp:

- Il **backend** (la logica interna) è correlato a uno smart contract che viene eseguito su una blockchain, ad esempio Ethereum. In questo modo, uno smart contract ha una programmazione che garantisce il funzionamento inequivocabile della DApp. Poiché i contratti intelligenti sono visibili e pubblici, è garantito un elevato livello di trasparenza e sicurezza. Le Dapp, quindi, garantiscono che non verrà eseguita alcuna operazione che non sia stata specificata prima da un contratto, scritto e caricato su una blockchain.

- L'**archiviazione dei dati** è completamente decentralizzata. Ogni utente DApp memorizza una cronologia completa delle azioni eseguite sulla rete DApp; inoltre, le interazioni vengono memorizzate nella blockchain all'interno dei blocchi. Tutto questo avviene in modo crittograficamente sicuro, impedendo l'accesso non autorizzato da parte di terzi.

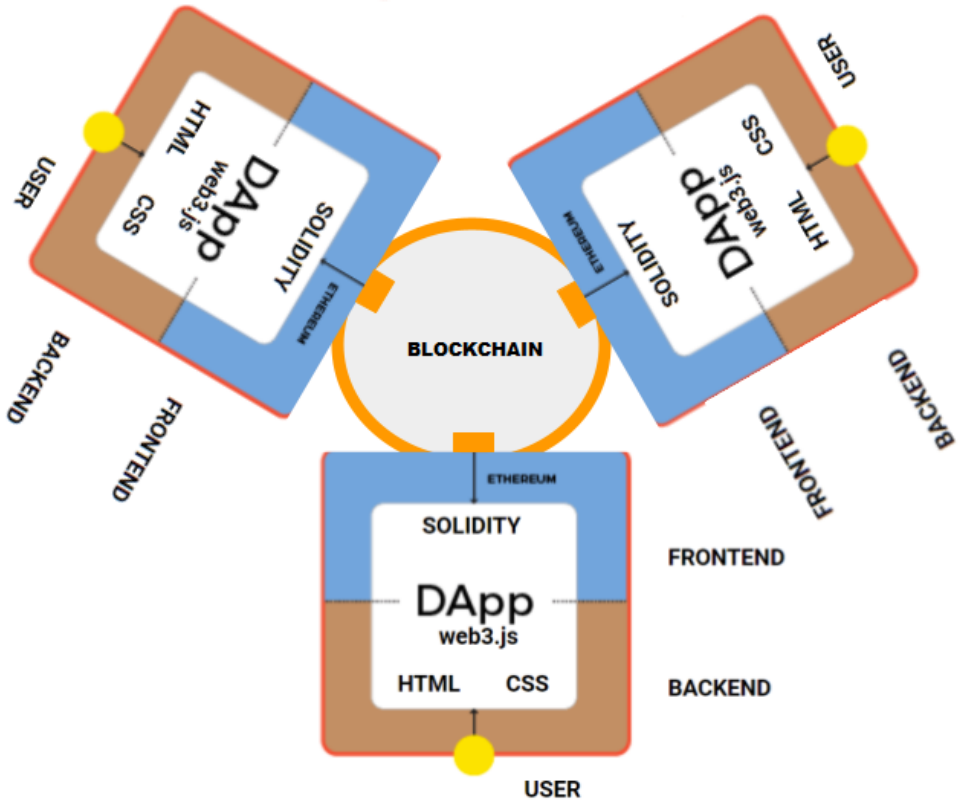

Figura 2.8: Struttura DApp

Una DApp funziona in modo simile ad una rete blockchain. In questo caso, ogni utente è un nodo all'interno della rete. Ciascun utente vigila sul corretto funzionamento e sulle operazioni svolte su tale rete. Lo *Smart Contract*, in questo caso, è un punto intermedio che è responsabile di corroborare la validità di ogni interazione. Ad ogni nuova operazione nella DApp, le informazioni sulla piattaforma vengono aggiornate in ogni nodo. Ciò garantisce che l'intero sistema di informazioni sia memorizzato in ciascuno di essi. In questo modo, ogni utente contribuisce al mantenimento dell'applicazione con le risorse del proprio computer. Questa struttura garantisce, inoltre, una piattaforma sempre in servizio, totalmente resistente ad un classico attacco Dos o DDos, in quanto è tecnologicamente impossibile rimuovere tutti i nodi dalla rete contemporaneamente.

Queste applicazioni basandosi completamente su un sistema blockchain godono degli stessi pregi quali sicurezza, privacy e persino anonimato. Con tali caratteristiche l'utente di una Dapp mantiene il controllo assoluto sui propri dati in ogni momento.

2.3.4 Testnet Ethereum

Una testnet (Test Network) è una rete sperimentale in cui gli sviluppatori possono testare, creare o modificare funzionalità e monitorare le prestazioni della rete blockchain. Una Testnet non che è una blockchain di Test che utilizza tecnologia e software identici alla blockchain principale, detta in questo caso "Mainnet". Tuttavia, al contrario della rete Mainnet che è utilizzata per transazioni "reali" con "valore", i testnet vengono utilizzati per testare Smart Contract e applicazioni decentralizzate ("DApp"). Questo ambiente sandbox consente agli sviluppatori di correre dei rischi, sperimentare e scoprire il miglior modello possibile, una versione stabile, da implementare nella Mainnet. Tutto ciò avviene su larga scala in modo controllato. I Testnet inoltre assicurano che le distribuzioni Mainnet avvengano più velocemente.

Differenze tra la Testnet e la Mainnet

Presupponendo un'architettura pressochè identica, l'unica differenza tra le Testnets e la Mainnet è identificabile nella governance; cioè un gruppo di utenti/nodi ha accettato di lavorare insieme e formare una comunità di test (rete Testnet), mentre un altro gruppo di nodi ha accettato di lavorare insieme per fungere da rete Mainnet [17]. Affinché una rete di computer accetti di "lavorare tra loro", è fondamentale che alcuni parametri della rete vengano settati in modo univoco:

- **Un ID di rete**: un parametro numerico che funge da identificatore per una rete. Ad esempio, l'ID di rete della Mainnet è 1, mentre le altre Testnet (Ethereum) più comunemente utilizzate hanno ID di rete di 3, 4 e 42 rispettivamente per Ropsten, Rinkeby e Kovan. Quindi, al fine di connettersi a una delle blockchain, bisogna eseguire il software Ethereum, specificare un ID di rete 1 per connetterlo alla Mainnet, oppure specificare un ID di rete di 3 per Ropsten.

- **Blocco genesi**: tutti i nodi su una rete devono accettare tutti i dati memorizzati su quella blockchain, che ovviamente dovrà iniziare da qualche parte : il blocco genesi. Il blocco di genesi è il blocco n.1 della blockchain di una rete: sono solo dati arbitrari che sono stati designati come l'inizio di quella catena-blockchain.

Alcune differenze principali [18]:

- **Nodi**: una rete di test ha meno nodi di una rete principale.

- **Costo delle operazioni**: nella testnet, i token non hanno alcun valore (*Test Ether*). Il costo delle operazioni nella mainnet è più alto. Ogni operazione

eseguita sulla blockchain richiede una commissione sotto forma di token che detiene un certo valore.

- **Frequenza delle transazioni**: la frequenza delle transazioni è bassa per una testnet.

Test Ether in una Ethereum Testnet può anche essere facilmente estratto, tuttavia esiste un altro modo piu semplice per acquisirne. Quest'ultimo è basato su un servizio Web noto come *Faucet*. Questi sono rubinetti impostati per acquisire e donare facilmente i token Testnet. Come accennato, il software utilizzato dai nodi su Testnet e Mainnet è identico. Quindi, se il software è esattamente lo stesso, perché un Test Ether non ha valore mentre Mainnet Ether ha valore? Il motivo per cui "Mainnet Ether" ha valore è proprio perché la community assegna valore a Ether che esiste sulla rete Ethereum con un ID di rete pari a 1. Questo è simile al concetto della banconota di 5 euro, in quanto è solo un pezzo di carta; l'unico motivo per cui ha valore è perché la BCE gli conferisce valore e il popolo in generale accetta tutto cio, usando la banconota da 5 euro per negoziare il valore.

L'implementazione sulla testnet non che è una simulazione di come funzionerebbe una qualsiasi piattaforma come QAXH.IO sulla Mainnet stessa, con token reali ed Ether reale. Poiché la testnet funziona allo stesso modo, è utile testare smart contract, transazioni e mining, simulando un ambiente reale. Esistono diverse Testnet disponibili per gli sviluppatori:

- **Ropsten**: La Ropsten Testnet (Network id: 3) è la rete più simile alla Mainnet. Utilizza lo stesso algoritmo di consenso ("prova di lavoro"), che ottiene il consenso attraverso il mining. Pertanto, qualsiasi nodo che si connette alla rete Ropsten può eseguire il mining per testare Ether. (Chaindata size 15 GB - Apr 2018)
- **Rinkeby**: La Rinkeby Testnet (Network id: 4) utilizza la PoA ed è immune dagli attacchi SPAM in quanto il rilascio di Ether è totalmente controllato. E' utilizzata dal progetto *QAXH.IO*. (Chaindata size 6 GB - Apr 2018)
- **Kovan**: La Kovan Testnet (Network id: 42) utilizza la PoA , tuttavia non supporta il tool Geth. (Chaindata size 13 GB - Apr 2018)
- **Goerli**: Avviata nel novembre 2018. È una testnet (Network id: 5) che ha l'obiettivo di essere ampiamente utilizzabile in tutte le implementazioni client che supportano il motore Clique PoA (EIP-225).

2.4 FinTech & Blockchain

L'industria finanziaria è stata una delle prime a notare le opportunità derivanti dalle tecnologie per la Blockchain, non soltanto per la necessità di partecipare ad un

mercato della criptomoneta, ormai in forte espansione. Il mondo Fintech è infatti per diversi aspetti fortemente vincolato ad architetture legacy, molte delle quali frutto di implementazioni sviluppate nell'arco degli ultimi decenni ma difficilmente sostituibili, per varie questioni, dall'efficienza all'entità dell'investimento necessario per una migrazione, dalla mancanza di alternative tecnologiche al non ultimo aspetto della sicurezza dei sistemi. [19] [20] [21]

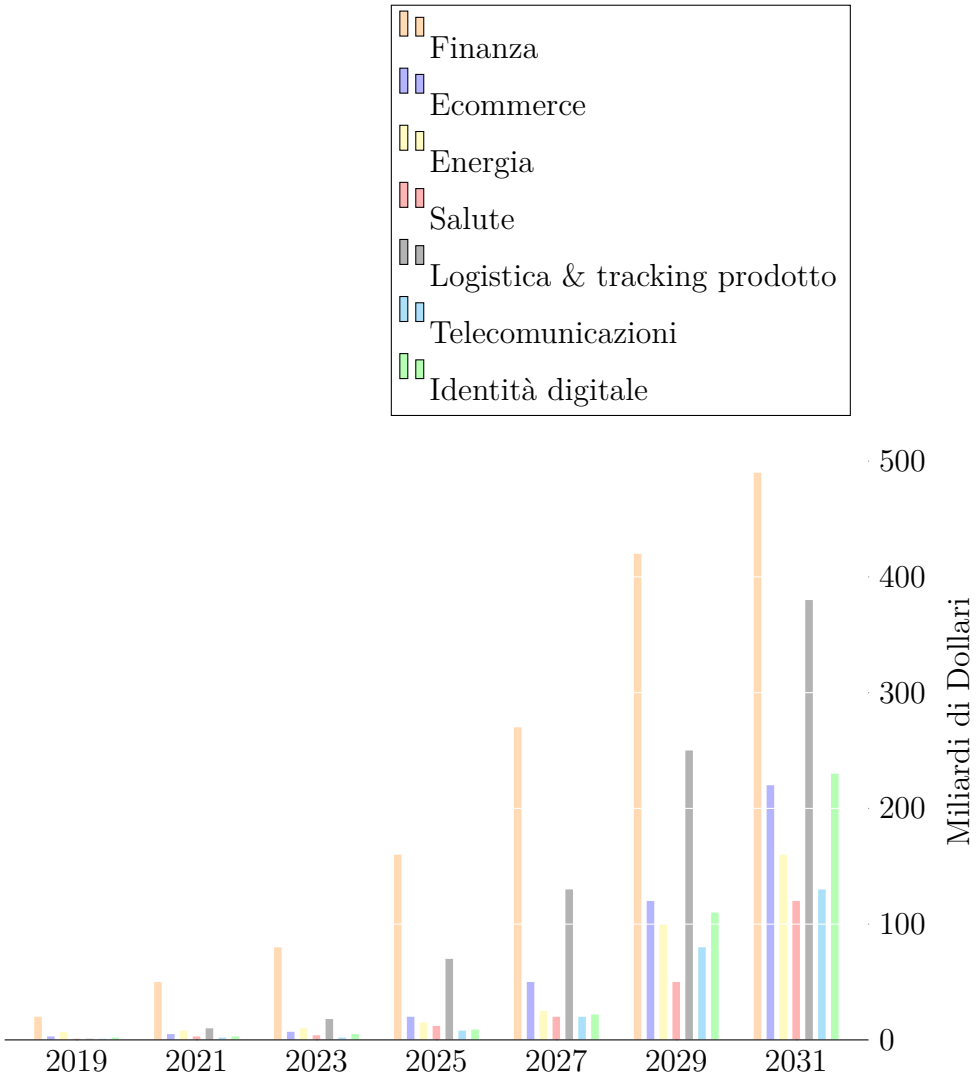

Figura 2.9: IHS Markit sugli investimenti in Blockchain

L'espandersi della Blockchain come soluzione *disruptive* avrebbe quindi portato un numero sempre più consistente di stakeholder del settore ad investire su questo sistema *trustless*.

Applicando il tutto su larga scala, le tecnologie distribuite rendono le procedure di trading più immediate e meno costose, garantendo un vincolo di fiducia tra i contraenti senza l'intervento di una terza parte.

E' quindi utile approfondire alcuni degli aspetti del Fintech che potrebbero essere maggiormente coinvolti, e migliorati, o peggiorati da implementazioni *Blockchain-based*:

- **Riduzione dei costi**
  Un utilizzo sistemico e strutturale delle DLT (distributed ledger technology) per tutti gli operatori del settore richiederebbe un impegno economico inferiore rispetto ad un sistema in cui ciascun istituto utilizzasse la propria piattaforma (piattaforme legacy) per la gestione delle transazioni.

- **Sicurezza-Trasparenza-Immediatezza**
  La velocità rappresenta infatti un fattore fondamentale per la *riduzione del rischio* in quanto viene notevolmente contenuta l'eventualità che uno dei contraenti venga meno ai propri obblighi. Inoltre, sulla questione della sicurezza occorre rammentare che le informazioni registrate tramite DLT sono durabili, virtualmente immutabili, tracciabili in tempo reale e totalmente trasparenti. Al monitoraggio possono partecipare anche gli enti regolatori fornendo garanzie dal punto di vista della compliance normativa, rendendo il tutto un'opportunità interessante per le implementazioni degli smart contracts.

- **Costo del Consensus**
  La validazione delle transazioni si basa su un sistema di Consensus fortemente dispendioso dal punto di vista energetico e computazionale. Ciò dipende da un sistema basato sulla interazione tra i nodi che opera in maniera bidirezionale: una singola transazione determina infatti un processo di validazione che influisce su tutta la catena dei dati, precedenti e successivi.

- **Ridondanza** I processi computazionali in un'infrastruttura centralizzata presentano una ridondanza inferiore rispetto all'impiego delle DLT. La ridondanza nasce quando un registro viene modificato e l'aggiornamento deve obbligatoriamente coinvolgere tutti i nodi ad esso associati. Nonostante possa risultare molto dispendioso non si tratta che del nucleo centrale per il meccanismo di certificazione di un'architettura distribuita.

Diversi studi affermano come l'impatto dei DLT per il settore finanziario possa essere paragonabile a quello di Internet negli anni '90 nel settore delle telecomunicazioni. A tal fine è utile citare un recente studio di *Santander* secondo il quale le blockchain

contribuiranno a ridurre i costi delle infrastrutture finanziarie di piu di 15 miliardi di dollari all'anno entro il prossimo biennio (2021-2023). [19] [22]

2.4.1 Casi d'uso per il mondo assicurativo

- **Ania**: Il primo caso d'uso è quello di Ania che nell'ambito di un progetto Insurance Blockchain SandBox, insieme ad alcuni team di Reply, ha sperimentato la blockchain in una sandbox, un ambiente protetto, nel quale testare ad esempio gli smart contract anche con la presenza del regolatore.

  "Nell'ambito assicurativo, ad esempio, la Blockchain assume un ruolo importante. Soprattutto cambia il ruolo dell'assicuratore che si affianca all'assicurato in una logica di servizio". Afferma Fabio Maniori, titolare dello Studio Maniori che assiste Ania nello sviluppo del progetto.

2.4.2 Le applicazioni per il mondo bancario

- **Mediolanum**: Banca Mediolanum ha reso noto di aver certificato l'immodificabilità della Dichiarazione Non Finanziaria (DNF) attraverso l'utilizzo di blockchain Ethereum e la conseguente pubblicazione dell'hash del documento sul sito istituzionale della Banca. [23]

- **Unicredit**: La piattaforma We.trade, targato Unicredit, basata su blockchain sviluppata per volontà di nove istituti bancari e "*di fatto ha l'obiettivo di promuovere la fiducia nelle transazioni internazionali*", superando anche quelle remore che avevano agli albori della sua storia, e che tenevano ben lontani gli istituti di credito dalla blockchain. Questo progetto copre infatti 11 Paesi e offre servizi di DLT e smart contracts per clienti con esigenze transazionali legate alle attività di import/export.

2.4.3 QAXH.IO in Fintech-Blockchain

La piattaforma Qaxh.io in questo scenario non è che un framework destinato all'ambiente bancario per gli utenti blockchain al dettaglio, funge quindi da collegamento tra un fornitore di servizi (banca, ente, istituzione..) e una persona fisica o una società. Per rispettare questo principale obiettivo progettuale, ll framework si focalizza in particolare sulla certificazione d'identità su un sistema blockchain e sull'implementazione di soluzioni attraverso lo strumento degli Smart Contract che consentano ai clienti bancari di interagire con essa.
In particolare il framework si propone di fornire:

- Un'applicazione mobile che permetta la creazione di account di identità (autodichiarati) che possano interagire su blockchain.

- Un'identità certificata degli account con una terza parte fidata tramite piattaforme amministrative di identità digitale.(Regolamento eIDAS per la privacy).
- L'uso customizzabile dei servizi offerti da un'azienda fintech al consumatore in totale sicurezza.
- Aprire il pieno potenziale della tecnologia blockchain agli utenti introducendo una versione interoperabile moneta elettronica, ai sensi della DME2 (Direttiva del Parlamento Europeo sull'accesso alle attività degli istituti di moneta elettronica).

Capitolo 3

Sviluppo dell'applicazione

3.1 Panoramica

L'obiettivo della tesi è sviluppare un'applicazione software, basata su blockchain, presso il Team di Innovazione 89C3 di BPCE SA. L'ente bancario ricerca una soluzione blockchain per controllare i processi interni legati ai meccanismi di gestione dei poteri bancari. Il sistema è caratterizzato principalmente dalla presenza di due parti (Banca & Utente,o azienda), quindi il software in questione deve implementare un meccanismo di permessi per gestire gli ordini finanziari emessi da un'entità fisica (persona o azienda) verso un'istituzione (banca ecc). In aggiunta, sul tema della sicurezza, l'azienda vuole sfruttare il sistema blockchain per garantire l'immutabilità e la trasparenza che contraddistingue questa tecnologia. Di seguito una panoramica del lavoro di tesi. Partendo dalle tecnologie utilizzate per la soluzione, è descritto il lavoro prodotto, inserito nel contesto di un progetto già esistente, ed infine i principali processi della tesi.

3.1.1 Casi d'uso

All'interno del contesto del progetto QAXH.IO, approfondito in seguito, l'elaborato riporta lo sviluppo di una soluzione che si pone come interfaccia nel rapporto banca-azienda, superando i classici ostacoli delle soluzioni legacy, figura 3.1. Al fine di focalizzare i punti principali della soluzione è possibile trasmutare il concetto delle *AAA*, abbastanza noto in ambito di telecomunicazioni, nel rapporto specifico Azienda-Banca e in generale in qualsiasi rapporto Issuer-Acquirer.

Un protocollo AAA, nella telematica, e in particolare nelle reti di telecomunicazione, indica un protocollo che realizza le tre funzioni di *Autenticazione*, *Autorizzazione* e *Accounting*. [24] [25]

La soluzione riportata si fonda sulla garanzia di questi 3 principi per entrambe le parti, affiancadone allo stesso tempo una quarta A, l'*Automatizzazione*, che si

occupa di garantire un ulteriore livello di sicurezza all'intero sistema. Quest'ultimo concentra le prime 2 A direttamente in un'unica soluzione tecnologica, la *Blockchain*, totalmente automatizzata e trasparente; allo stesso modo si occupa della terza automatizzandola ma lasciandole un grado di libertà al fine di poterla customizzare. L'intera *HabilitationFolder* punta all'idea di ridurre sensibilmente gli interventi umani (possibili errori o manomissioni) senza, al contempo, limitare le possibilità di gestione da parte dei manager delle parti coinvolte.

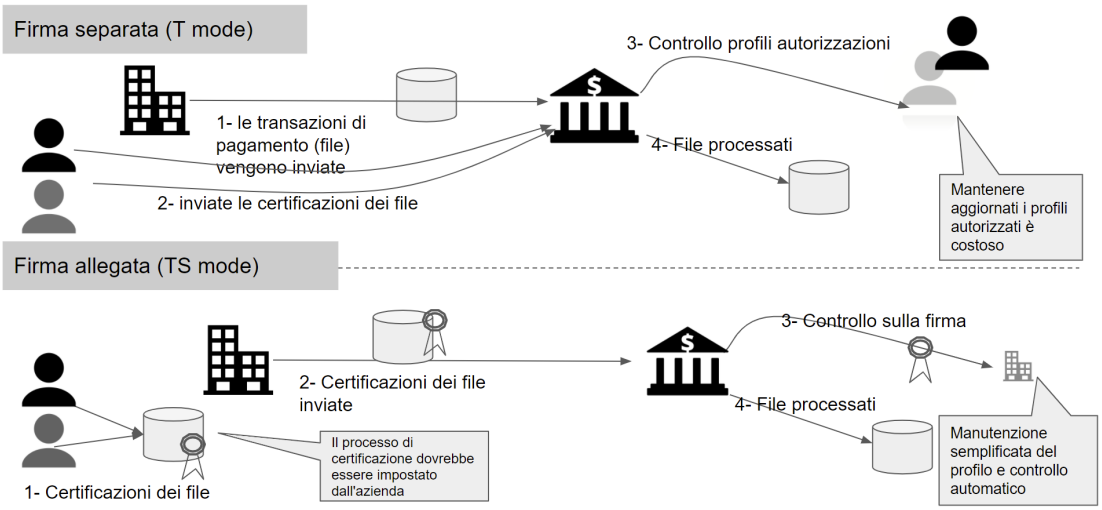

Figura 3.1: Sistema di ordini tra azienda e banca (*Classico*)

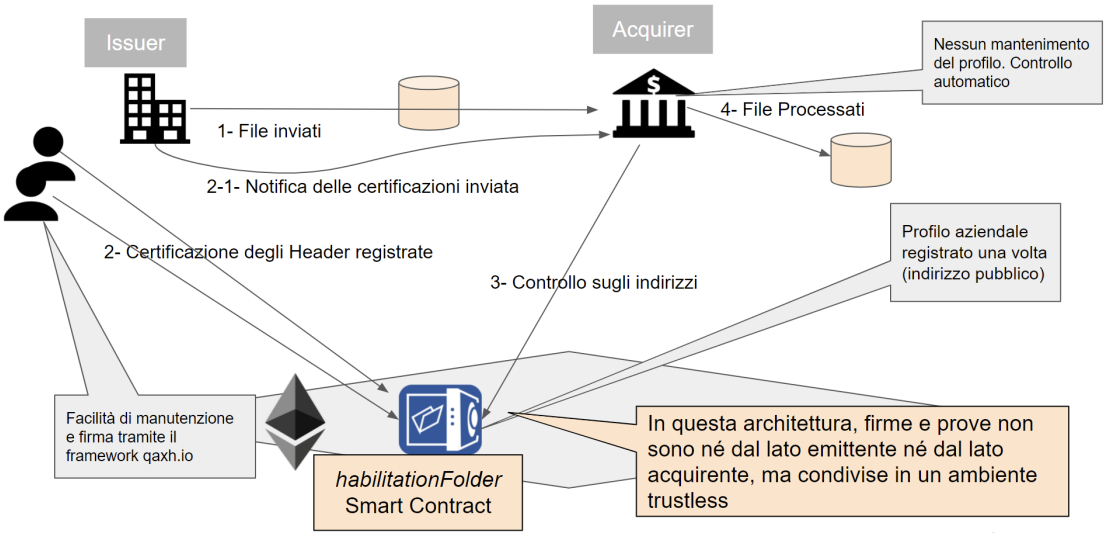

Figura 3.2: Soluzione per la gestione dei poteri bancari

Attori principali

- **CEO**: L'amministratore che deciderà di utilizzare il servizio fornito dal *habilitationFolder* e avrà i pieni poteri all'interno dello stesso.
- **Contabile**: Il rappresentante d'azienda che potrà "creare" gli ordini nel sistema.
- **CFO**: L'ufficiale che potrà "firmare" gli ordini all'interno del sistema.
- **Acquirer**: Il manager di banca che gestirà e offrirà il servizio.
- **Server Bank**: Il server della banca che gestirà automaticamente file e notifiche.

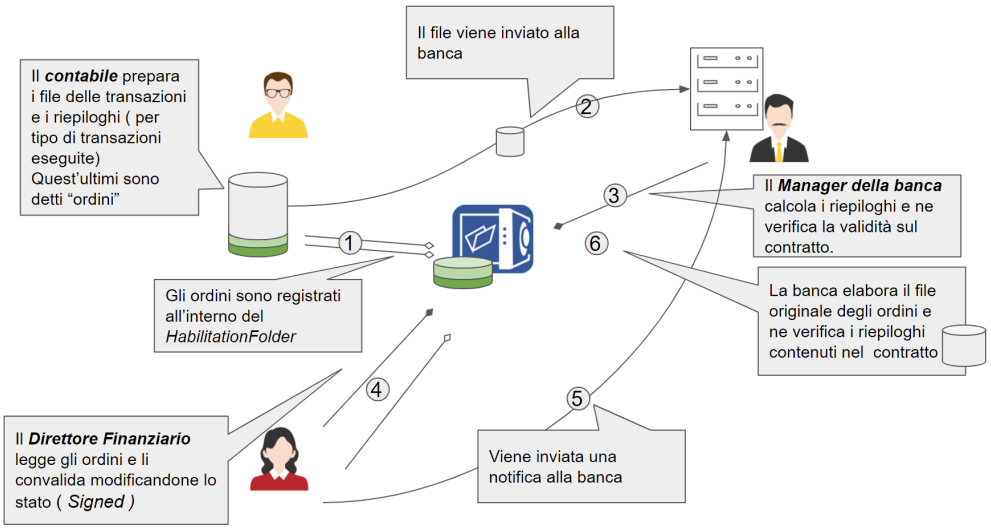

Figura 3.3: Attori

Regole del Business

1. La banca non dovrebbe conoscere le persone che hanno firmato ma solamente la notifica che è stata consegnata una firma della società per un conto a un servizio.
2. Durante le operazioni la banca dovrebbe interagire solo con *HabilitationFolder*
3. Alla base dell'implementazione si pone il concetto di un UNICA parametrizzazione del sistema in questione, occupandosi principalmente di registrare un *CorporateSafeAddress* all'interno.

4. L'intestazione del file (importo totale, numero di record, riferimento, valuta, data, categoria) deve essere presentata in chiaro all'utente che firma.

5. L'azienda è responsabile della gestione delle deleghe (firma e/o creazione) ai propri dipendenti utilizzando direttamente le opportunità concesse dalla *HabilitationFolder*.

3.1.2 Tecnologie

- **Ethereum**: Ethereum è una piattaforma software decentralizzata open source, basata su blockchain. Consente di creare ed eseguire contratti intelligenti e applicazioni distribuite (DApp). [26]

- **Metamask**: viene utilizzato come portafoglio Ethereum per eseguire e firmare le transazioni avviate dalle dapp. Sfruttando l'API Metamask, viene garantito un elevato livello di sicurezza per eseguire transazioni sulla rete Ethereum. Per il corretto utilizzo dell'intera applicazione è obbligatorio che l'utente sia loggato in Metamask tramite il wallet specificato in fase di registrazione. [27]

- **Web3**: è la libreria software utilizzata per interagire con gli smart contract. L'API Web3.py / Web3j soddisfa le esigenze degli sviluppatori per l'integrazione tra client/server e blockchain di Ethereum. È una raccolta di librerie che permette agli sviluppatori di eseguire azioni come inviare Ether da un account a un altro, leggere e scrivere dati da contratti intelligenti, creare contratti intelligenti e molto altro. [28] [29]

- **Truffle**: è un ambiente di sviluppo, un framework di test e una pipeline di risorse per blockchain che utilizzano Ethereum Virtual Machine (EVM). Utilizzabile anche direttamente da linea di comando [30]. Caratteristiche principali:

  1. Compilazione, collegamento, distribuzione e gestione binaria di smart contract integrati.
  2. Test automatizzato del contratto per uno sviluppo rapido.
  3. Gestione della rete per l'implementazione su qualsiasi numero di reti pubbliche e private.
  4. Console interattiva per la comunicazione diretta del contratto.
  5. Runner di script esterno che esegue script all'interno di un ambiente Truffle.

- **Remix**: è un editor online che consente di sviluppare Smart Contract ben strutturati in Solidity. Grazie ai plugin, installabili sull'editor, è possibile

compilare il codice scritto degli smart contract. Una volta concluso con successo il processo di compilazione, sviluppa i rispettivi bytecode e ABI dello Smart Contract. Sia il bytecode che l'ABI vengono utilizzati per definire i comportamenti degli smart contract. Questi parametri sono utilizzati durante il processo di distribuzione. [31]

- **Flask**: è un micro-framework web utilizzato per sviluppare app web e API. È un framework leggero, facile e veloce che integra diversi metodi utili per lo sviluppo di API HTTP e middleware.
- **Infura**: consente di eseguire un nodo Ethereum, per impostare un endpoint utilizzato per interagire con il contratto. Consente in modo semplice di configurare un endpoint pubblico per l'indirizzo del contratto distribuito. Fornisce API e chiave personali per l'accesso all'endpoint. Inoltre, fornisce una dashboard ben definita e dettagliata per analizzare tutte le chiamate degli smart contract, fornendo analisi più approfondite anche per il metodo chiamato. [32]
- **Etherscan**: è una piattaforma di esplorazione e analisi dei blocchi da un lato, dall'altro una piattaforma di contratti intelligenti decentralizzata. [33]

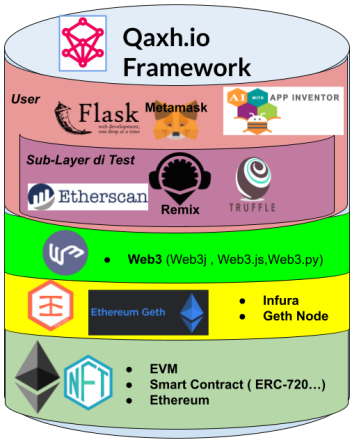

Figura 3.4: Strati tecnologici di QAXH.IO

3.2 Progetto QAXH.IO

Al fine di focalizzare il lavoro svolto è necessario riportare una visione d'insieme dell'intero sistema con i suoi obiettivi e attori principali. Da un lato quest'ultimo propone un nuovo "livello" tecnologico da sovrappore alla tecnologia attuale (Web, API, Mail) al fine di securizzare e migliorare i rapporti tra i principali attori in ambito bancario.

E' possibile delineare le proprietà esistenti dai quali parte il processo di efficientamento:

- Autenticazione, consensi, prove gestite bilateralmente
- Stoccaggio su un singolo attore (sistema master-slave)
- Sistemi interprofessionali centralizzati

Il *Tier* in questione propone nuove proprietà, quali:

- Attori identificabili e/o identificati
- Consensi certificati
- Transazioni verificate
- Sistema Blockchain , un "bene comune" condiviso tra tutti gli *stakeholders*.

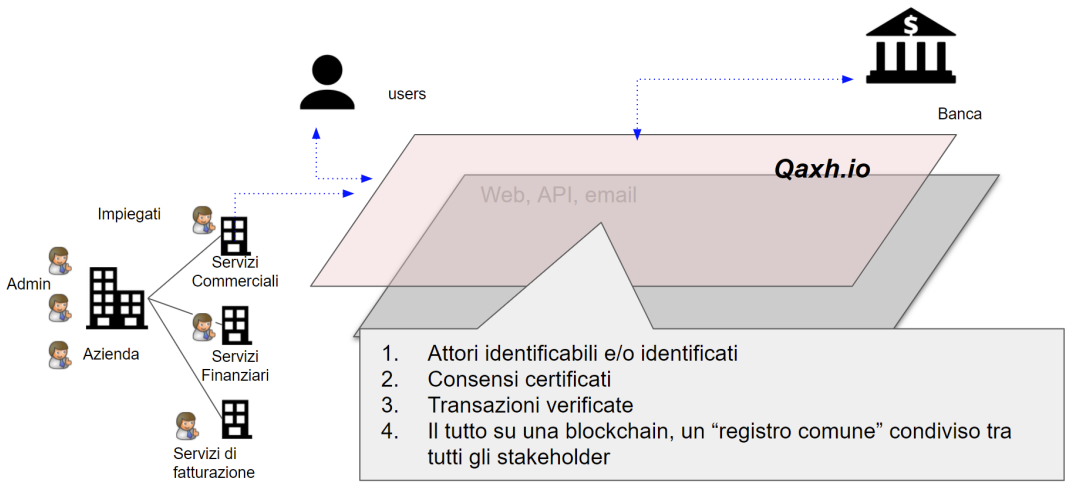

Figura 3.5: Securizzazione di QAXH.IO

Dall'altro l'ideazione di un livello QAXH.IO è dovuta all'esigenza di conciliare i concetti di cliente bancario e utente blockchain:

Cliente bancario	**Utente Blockchain**
Milioni di persone, fisiche e giuridiche, e per ogni persona...	x0961E3..4b145c798b5, ovvero una chiave crittografica
Un'identità (obbligo legale di KYC)	E se questa chiave viene persa, tutte le risorse vengono perse
Un conto bancario	Tutto ciò che è scritto sulla blockchain può essere letto da chiunque
Il diritto di commettere errori (ad es. perdere la chiave)	Tutte le transazioni sono verificate crittograficamente
Protetto e vigilato dalla legge (GDPR, E-IDAS, DSP2, CMF)	L'esecuzione di programmi per computer deterministici è sicura
Per il quale le banche creano e/o distribuiscono servizi	
In un ambiente prevalentemente euro (CMF, DME2)	

Tabella 3.1: Cliente bancario - Utente blockchain

Dati tali presupposti è possibile identificare la natura del progetto attraverso alcune sue proprietà fondamentali :

1. La capacità di sperimentare e sfruttare le interazioni tra Smart Contract e utenti.

2. L'accesso esterno alla piattaforma totalmente tramite API.

3. La produzione di un'estensione per un linguaggio Android (AI2) che consenta una perfetta interazione tra piattaforma e blockchain.

4. Agilità e la flessibilità del sistema.

5. Garanzia dell'identità (KYC) e protezione delle criptovalute contro lo smarrimento delle chiavi.

6. Trasparenza e immutabilità garantite dalla blockchain pubblica.

7. Pratiche di sviluppo (API, Low-code) adattate al contesto esplorativo.
8. Tutela della proprietà intellettuale sotto forma di brevetti.

3.2.1 Casi d'uso generali

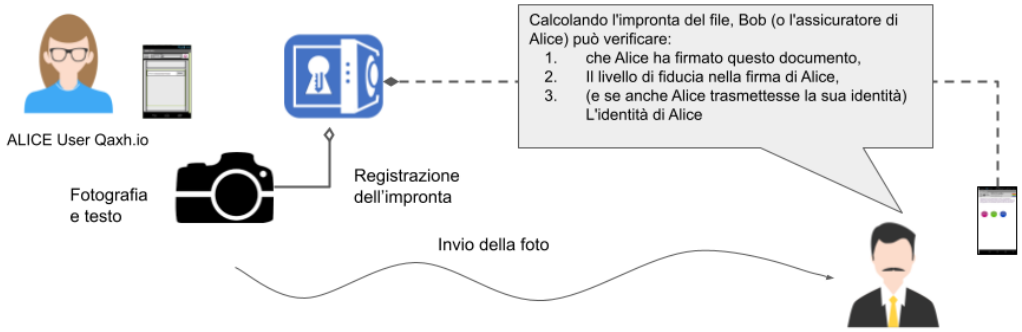

Figura 3.6: Firma elettronica con qualsiasi smartphone

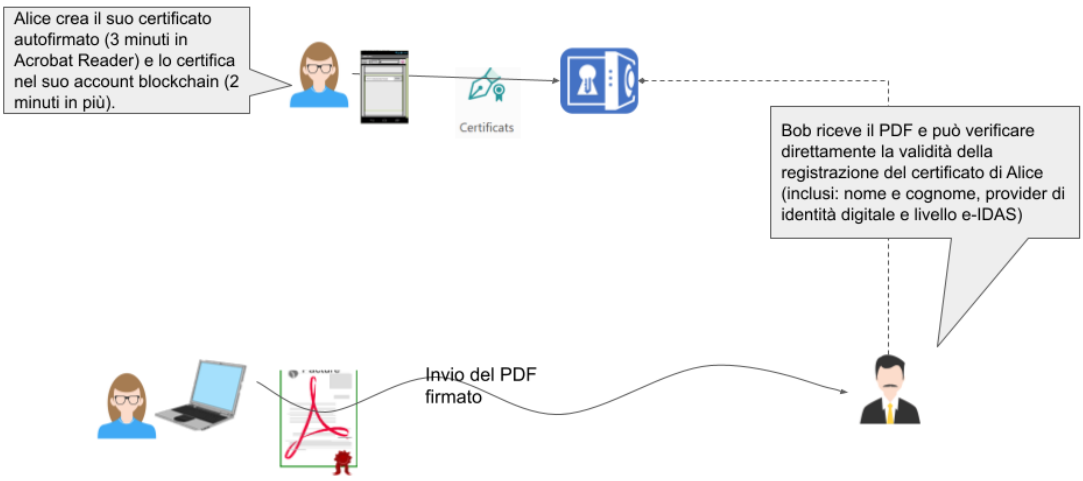

Figura 3.7: Firma identificata del documento PDF (Acrobat Reader)

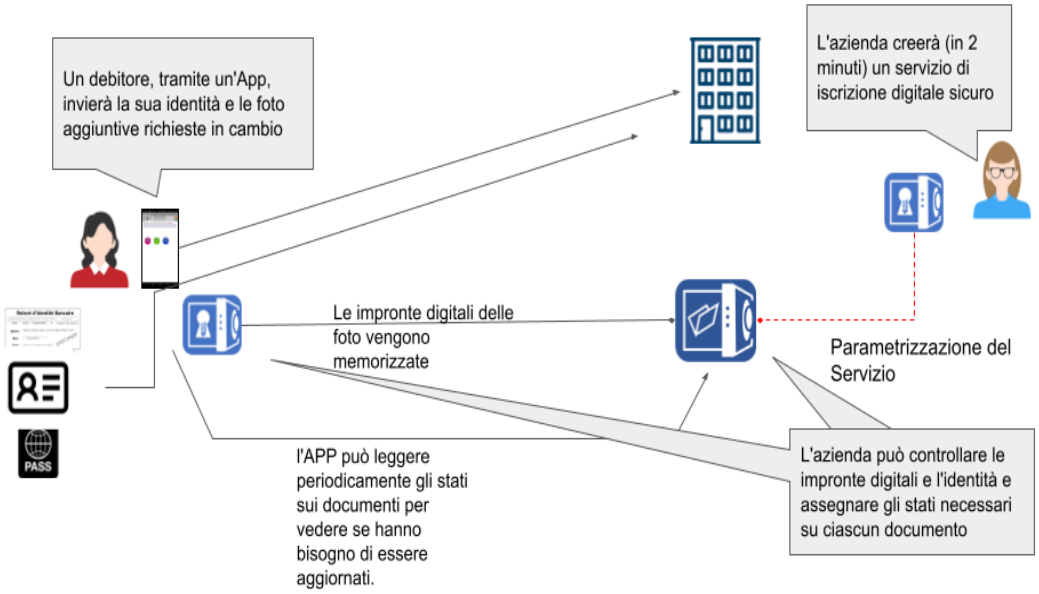

Figura 3.8: Iscrizione digitale securizzata (*OnBoarding Folder*)

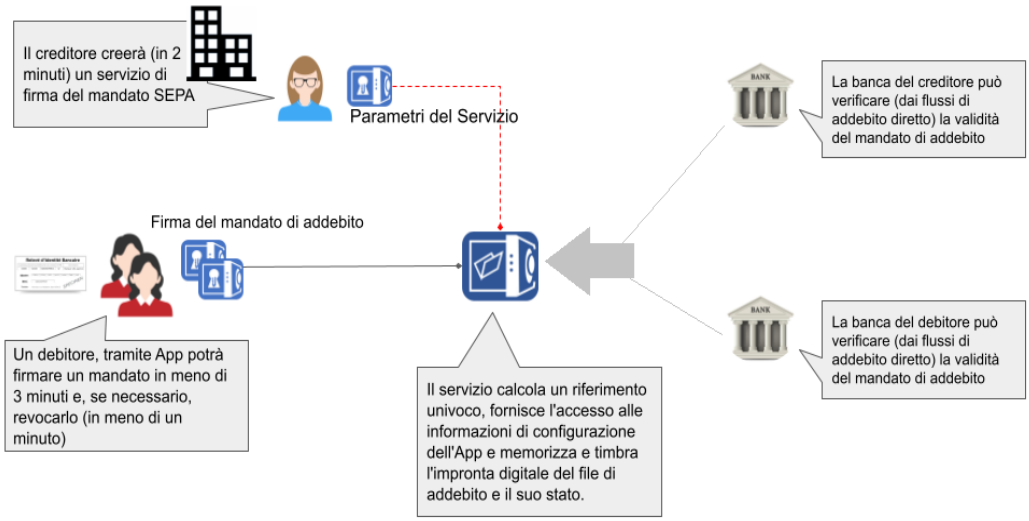

Figura 3.9: Mandato di addebito (*MandateFolder*)

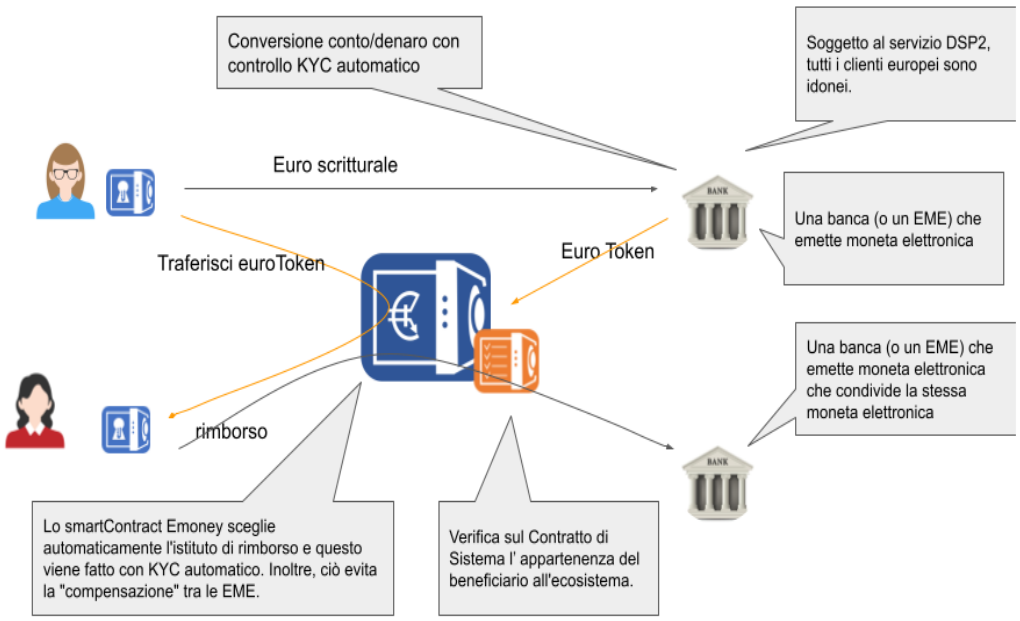

Figura 3.10: Moneta elettronica (DME2) in formato Blockchain (*Emoney Contract*)

3.2.2 Architettura generale

Come mostra la figura 3.13, il framework *QAXH.IO* si impone come una piattaforma di mezzo in un triangolo che coinvolge un End-Corporate user, un sistema blockchain, e altri servizi esterni di identifiazione personale.

All'interno della struttura è possibile distinguere, partendo da sinistra nella figura 3.13, alcuni elementi principali con i quali Qaxh.io interagisce:

- **User-Smartphone**: Un utente, *End user*, con un moderno smartphone con 2 semplici caratteristiche:

  1. Connessione alla rete.
  2. Utilizzo della fotocamera.

  Lo Smartphone permetterà all'utente sia di interagire direttamente con i contratti distribuiti su Blockchain, e sia di distribuire un nuovo contratto con dei parametri customizzabili. Inoltre grazie allo smartphone si potrà inquadrare dei QRcode per recuperare agilmente nuove informazioni e produrne dei nuovi, a partire dalle proprie informazioni personali.

- **Company**: La piattaforma QAXH.IO offre svariate possibilità al gruppo-azienda. Quest'ultimo, oltre ad essere un insieme di individui, con le possiblità

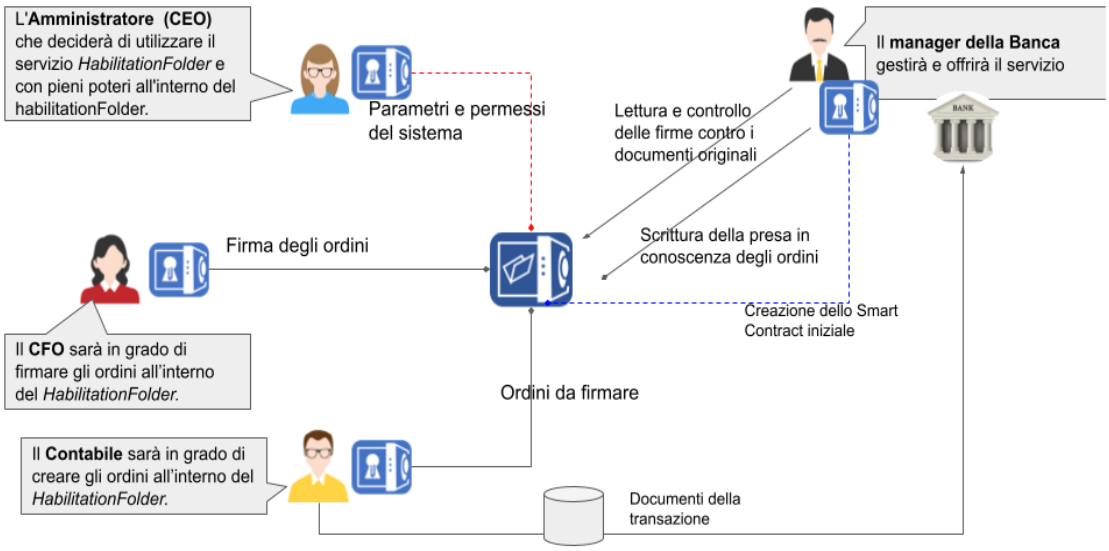

Figura 3.11: Gestione dei poteri bancari (*HabilitationFolder*)

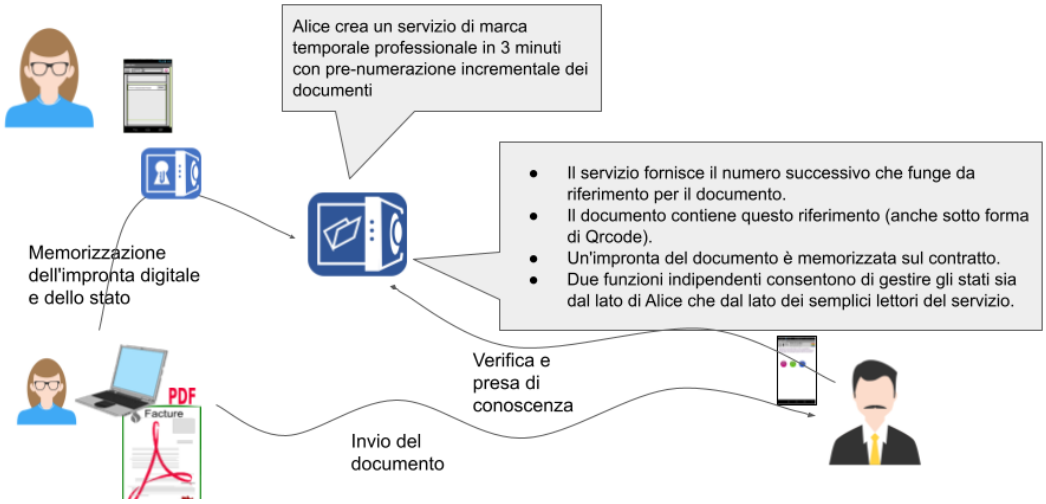

Figura 3.12: Marcatura temporale professionale dei documenti inviati

descritte al punto precedente, è pensato anche all'interno della piattaforma in quanto sono stati implementati degli Smart Contract per azienda con all'interno vari ruoli operativi (*CEO*, *CFO*, *Manager*..). I ruoli operativi manteranno diritti e doveri diversi, e di conseguenza non saranno mai statici

ma godranno di una certa flessibilità ed intercambiabilità.

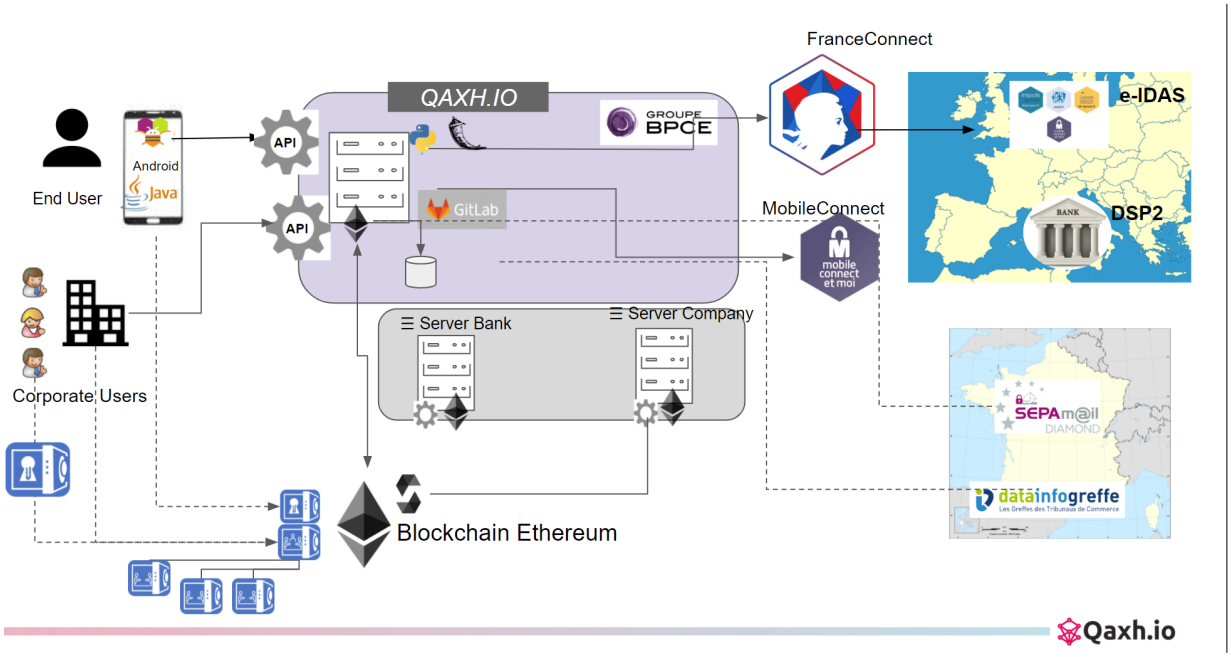

Figura 3.13: Panoramica QAXH.IO

- **Blockchain**: Il sistema Blockchain è il core dell'intera architettura, permettendo in maniera abbastanza flessibile la conservazione delle informazioni, la privacy che richiedono delle informazioni bancarie ed infine l'utilizzo di contratti informatici irrevocabili. Come verrà approfondito in seguito in una sezione dedicata, il sistema è pensato per funzionare su *Ethereum*, tuttavia la versione di QAXH.IO che è in fase di sviluppo, è testata su una rete blockchain di test, *Rinkeby*.
- **Sistema di identità digitale**: Qaxh.io si poggia totalmente su un sistema amministrativo di natura digitale nel quale esistono solo account con un'identità verificata. Questo sistema è solitamente pubblico ed a disposizione di tutte le autorità amministrative. In Francia, è possibile utilizzare il sistema *FranceConnect*. Quest'ultimo è un analogo dello *SPID* su suolo italiano.
- **Provider di identità digitale**: Al fine di mettere QAXH.IO sullo stesso piano di un servizio amministrativo, all'interno di *FranceConnect*, è fondamentale l'uso di un provider che verifichi tutte le identità del caso. A tal proposito all'interno del progetto è utilizzato *Mobile Connect et moi* (*Orange*).

Entrando nel dettaglio dell'architettura interna 3.14:

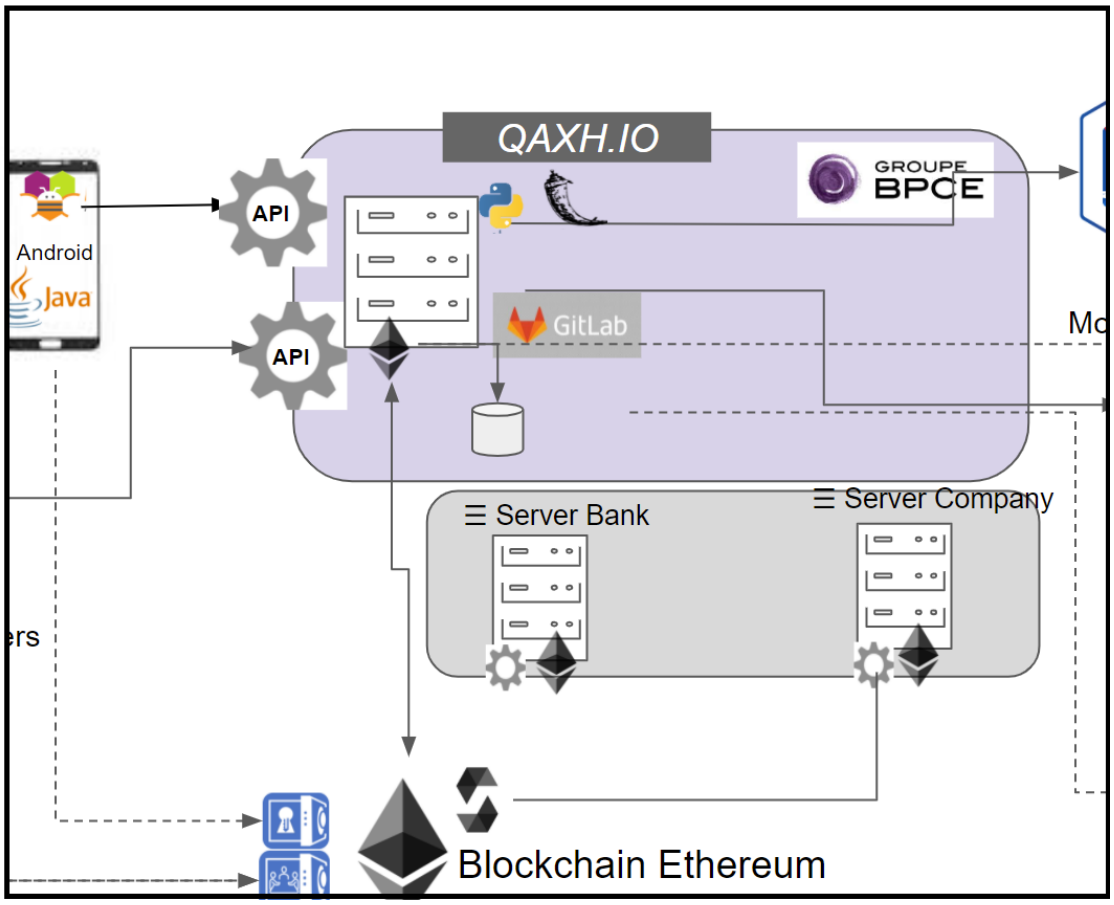

Figura 3.14: Panoramica QAXH.IO (2)

- **App for User**: Un'applicazione prototipo che utilizza le funzionalità offerte tramite funzioni Java ed API. L'app di test attualmente sviluppata è stata implementata grazie al tool *APP Inventor*, sviluppato da Google in collaborazione con il MIT per la programmazione *a blocchi*. Questo ambiente di sviluppo, utilizzabile totalmente sul Web, è stato ideato soprattutto per individui che volessero programmare semplici applicazioni per android, ad uso personale, in maniera rapida e funzionale.

- **Logica implementativa**: Come anticipato nel punto precedente, la logica di App Inventor si poggia totalmente su alcune funzioni Java indipendenti (come dei *blocchi* singoli 3.15), che interagiscono con la blockchain, attraverso la libreria Web3j, sviluppata e distribuita direttamente dalla fondazione *Ethereum*.

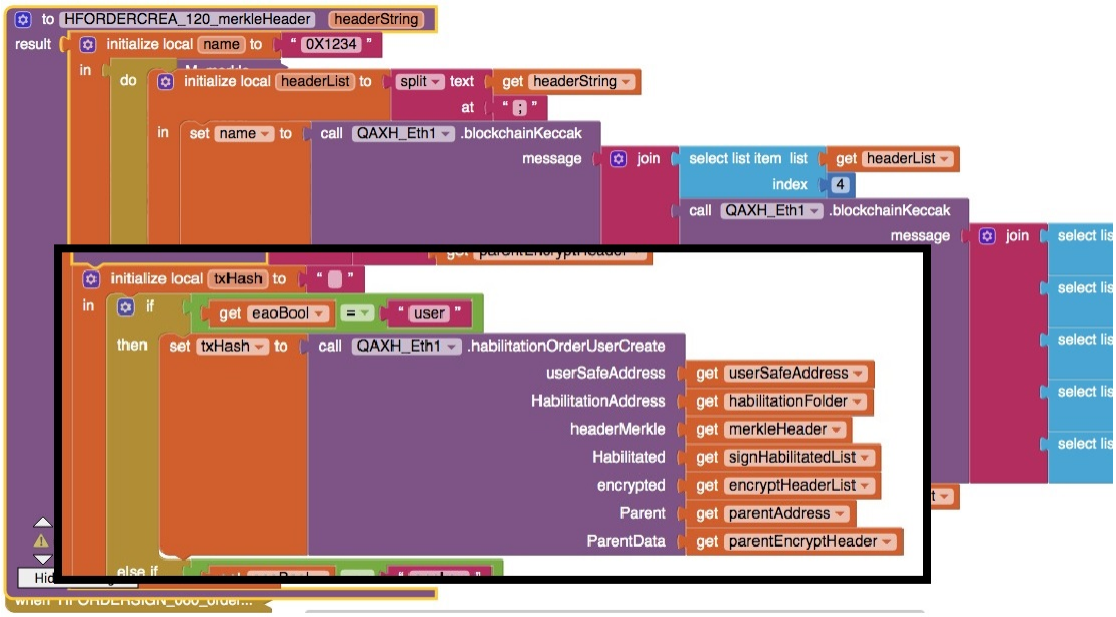

Figura 3.15: Blocchi Java in App Inventor

- **External API**: All'interno della piattaforma alcune operazioni complesse sono svolte da alcune *Application programming interface*, implementate sui server d'azienda. Queste permettono sia l'interazione sia la distribuzione di nuovi contratti sulla Blockchain.
- **Server Bank**: Sui server del progetto sono state implementate le API che interagiscono con la blockchain (server Acquirer). La possibilità di scrivere ed interagire sulla blockchain è fornita da un nodo *Geth* con rispettivo account Metamask (indirizzo & chiave privata) configurato sul server.
- **Server Company**: Tecnicamente è configurato come il Server Bank, tuttavia è utilizzato come server esterno, d'azienda, al fine di implementare tutte quelle API destinate ad essere utilizzate dalle aziende per interagire con Qaxh.io.
- **Database**: Qaxh.io dispone di una piccola base di dati, all'interno, che mantiene le informazioni fondamentali per il corretto funzionamento (Indirizzi, Chiave private, dati sensibili).
- **Strumenti di Versionamento**: Qaxh.io in quanto progetto, sviluppato da individui diversi, e con tecnologie molto diverse, si equipaggia di una piattaforma di versionamento. In dettaglio, è stato utilizzato *GitLab*.

3.3 Smart Contract

Partendo concettualmente da una soluzione blockchain, è possibile porre alla base dell'intera struttura del progetto il concetto tecnico di Smart Contract. Quest'ultimo permette di creare del codice sulla blockchain che sia accettato da tutti i componenti, e che inoltre regoli perfettamente e chiaramente ogni tipo di commercio, bancario o civile. Secondo la nota regola del "Code is law", il codice che compone lo smart contract è propriamente legge, diventando immutabile e inviolabile. All'interno di Qaxh.io gli utenti e le istituzioni affidano determinate informazioni allo Smart Contract e usufruiscono dei servizi che offre in piena sicurezza (Creazione identità, firma degli ordini, verifica delle autorizzazioni alle transazioni).

3.3.1 Smart Contract *QAXH.IO*

Le fondamenta concettuali e funzionali di QAXH.IO si basano sulla tecnologia Blockchain e i suoi smart contract. Per tale motivo è possibile facilmente elencare un insieme di funzionalità offerte dalla piattaforma che corrispondano ad un rispettivo contratto intelligente, distribuito su Ethereum [34].

Main Actor-Contracts

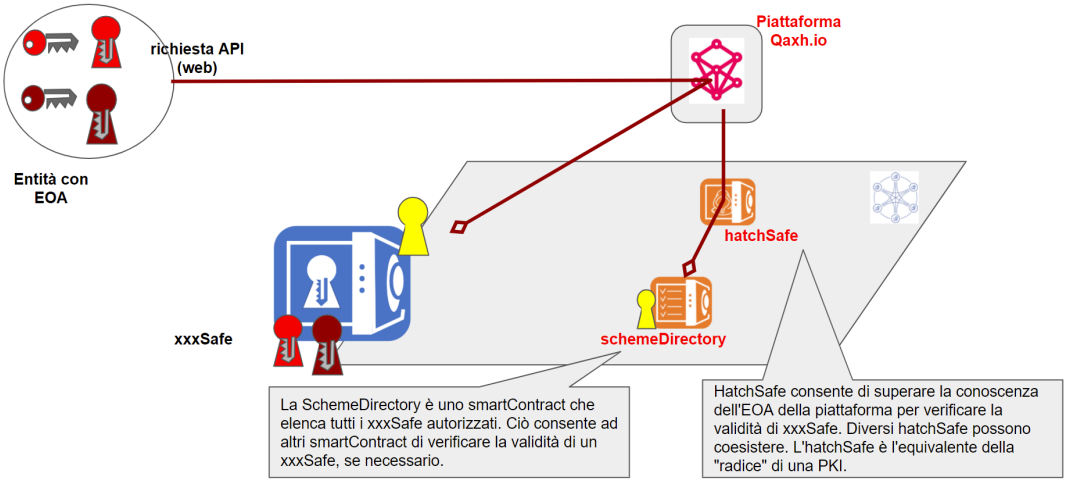

Figura 3.16: Contratti *User* in QAXH.IO

- **UserSafe - Proxy Diamond Pattern EIP-2535**: rappresenta il primo contratto implementato (insieme a librerie ed interfaccie) in QAXH.IO E' utilizzato per definire un utente sicuro all'interno del contesto della piattaforma.

- **CorporateSafe**: è l'analogo di un usersafe per entità legali, come le aziende. Permette inoltre di far da scheletro per nuovi contratti per user-delegati e dipendenti da un contratto CEO.
- **HatchSafe**: rappresenta un server di QAXH.IO che, ad esempio, permette la creazione/distribuzioni di contratti terzi. L' "Hatch" mantiene riferimenti verso diversi "Scheme" per Customer_ID.
- **SchemeDirectory**: rappresenta un database-QAXH.IO (contenente l'insieme dei contratti) sulla blockchain che mantiene le strutture principali che ne garantiscono il funzionamento. Lo "Scheme" può mantenere riferimenti verso più "HatchServer".

Services-Libraries Contracts

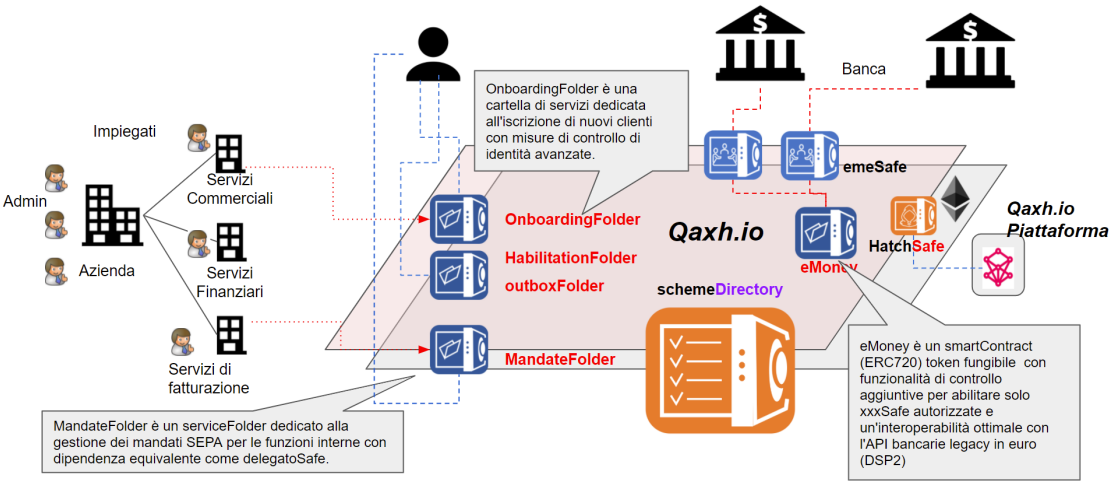

Figura 3.17: Contratti di servizio in QAXH.IO

- **OnBoarding**: contratto che offre un servizio di gestione dei documenti "sicuri" e "da verificare".
- **EmeSafe**: contratto che offre un servizio di moneta elettronica interoperabile.
- **Mandate**: contratto che offre un servizio di addebito/ mandato di prelievo.
- **Habilitation**: contratto che offre un servizio di gestione dei poteri bancari.
- **IotSafe**: contratto che offre la securizzazione di Actor-Iot.

SchemeDirectory

Il contratto *SchemeDirectory* permette di verificare se un "Safe" o un qualsiasi altro Smart Contract è stato creato da una piattaforma autorizzata *Qaxh.io*. Poiché tutti i *User-contracts* qaxh.io sono registrati all'interno, lo *SchemeDirectory* non è che elenco di tutti i contratti "Safe" esistenti (Utenti, aziende, childSafe, EmeSafe, ecc...) e una pseudo-lista di ogni contratto elettronico creato.
Principali caratteristiche:

- Lo *SchemeDirectory* è uno Smart-Contract multi-firma che può essere gestito da diversi hatchSafe.
- Lo *SchemeDirectory* è distribuito insieme ai HatchSafe da entità chiamate "Keepers".
- L'ideale è uno *SchemeDirectory* per ID cliente.
- Tutti i contratti di moneta elettronica devono essere registrati nello *SchemeDirectory* da un HatchSafe.
- Tutti i contratti sono registrati da un HatchSafe. Sono necessari 4 parametri:
  1. SafeType (un valore compreso tra 1 e X).
  2. L'ID cliente.
  3. Un valore numero (1 per vero/ 0 per falso) che indica se un contratto implica uno scambio di Ether.
  4. L'indirizzo del HatchSafe che aggiunge il "Safe-Contract".

	Adresse Bi					
Owner		Mode A				
0x...EOAb		2				
ParentAddress						
0x...EOAr						
		liste des (Ni, Aj) (liste S2 certified Safes)				
liste des A autorisés (liste S1, authorised hatch Safes)		Ni certifiés	Aj de certification	customerI D	eMoney	safeType
0x...A1		0x....N1	0x...A1	7	TRUE	1
0x...A2		0x....N2	0x...A1	7	TRUE	1
0x...A3		0x....N3	0x...A1	5	FALSE	2

Figura 3.18: Struttura *DirectoryScheme*

- Qualsiasi utente su Ethereum può sfruttare una funzione dello *SchemeDirectory* per sapere se un contratto è stato registrato o meno.

```
struct safe {
      uint256 safeType;
      uint256 customerId;
      uint256 money;     // 0 false   -   1 true
      address linkedPlateform;
  }

  mapping (address => safe) internal registeredSafes;
```

Listing 3.1: Contratti Safe registrati

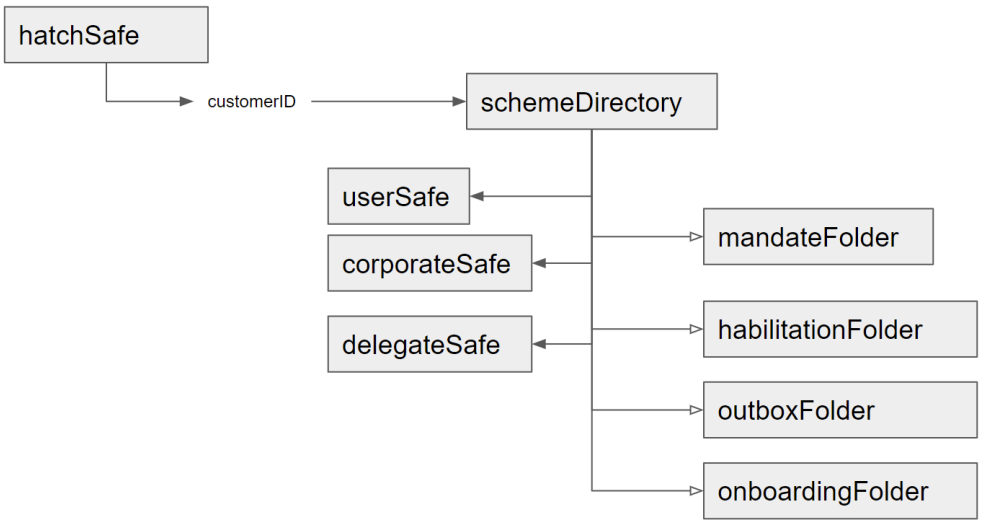

Figura 3.19: Hatch - Scheme - Folder

HatchSafe

Il contratto HatchSafe nasce dall'esigenza di creare un protocollo per semplificare i vincoli di distribuzione, vale a dire:

- Consentire al distributore di terze parti di modificare facilmente i propri EOA.
- Ridurre i vincoli per l'utente che controlla la distribuzione.

Principali attori:

1. Il deployer di terze parti, denominato *hatchServer*.
2. *HatchSafe* che rappresenterà il server sulla blockchain.
3. *SchemeDirectory* contratto che elencherà i Smart Contract distribuiti.

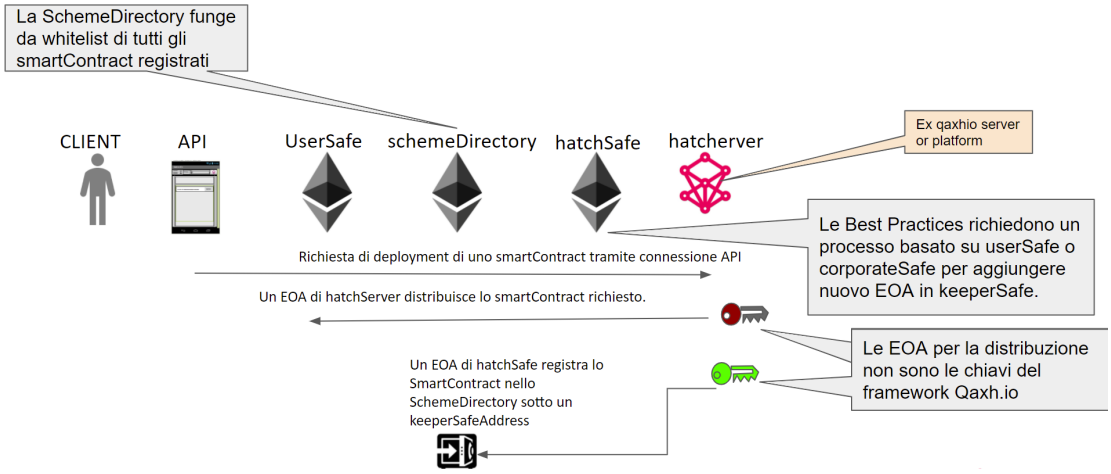

Figura 3.20: Soluzione Hatch

La soluzione implementa uno *Hatchsafe*, ideato per la scrittura nello *SchemeDirectory*, con le seguenti caratteristiche :

- Può essere gestito da uno o più EOA (5 per la chiave privata e 4 per la chiave pubblica), oppure uno o più indirizzi di Smart Contract.
- Può registrare lo smart contract distribuito nello *SchemeDirectory*. Quest'ultimo indicherà l'indirizzo del *Hatch* accanto a quello del contratto registrato, come mostrato nella 3.18 .

```
1  /// @dev    This function is called by the server to add a given
     hatch in a scheme.
2      ///         The EOA of the server must be in the certifiedKey
     map.
3      /// @param  _safeAddr the address of the safe.
4      /// @param  _safeType the Type of safe.
5      /// @param  _customerId the customerId of the safe, used to
     know in which Scheme to deploy.
6      /// @param  _money value used to know if the safe is allowed
     to use money or not.
7      function registerSafe(address _safeAddr, uint256 _safeType,
8          uint256 _customerId, uint256 _money) public
     onlyCertifyingKeys()
```

```
9     {
10        if (schemeDirectoryIsActive(_customerId) == uint256(1)) {
11            address _schemeAddr = schemeDirectoryAddressGet(
    _customerId);
12            if (_schemeAddr != nullAddress) {
13                SchemeQAXH scheme = SchemeQAXH(_schemeAddr);
14                scheme.addSafeFromHatch(_safeAddr, _safeType,
    _customerId, _money);
15            }
16        }
17    }
```

Listing 3.2: Registrazione nello Scheme

UserSafe: Diamond standard

Il framework QAXH.IO garantisce un livello astratto tra la chiave privata all'interno dello smartphone e l'Indirizzo di quell'identità che possiede token o ether. La piattaforma crea quindi un contratto "Safe" per utente con una SafeKey dedicata e autorizza l'AppKey presentata dall'utente a interagire con questo contratto. È possibile inserire più di un App-Address nella cassaforte. Inoltre tali AppKey potrebbero essere "congelate" (temporaneamente impossibili da utilizzare) o cancellate. Solo la piattaforma Qaxh.io è in grado di concedere l'accesso a un nuovo AppAddress all'interno di un *UserSafe*.

Secondo questo principio, se un AppAddress viene perso, è possibile associare un nuovo appAddress allo UserSafe, ma le risorse rimangono con quest'ultimo. L'associazione di un App-Address ad uno UserSafe avviene durante il processo di identificazione, supervisionato dalla piattaforma Qaxh.io. Al fine di tutelare l'utente, la piattaforma Qaxh.io ha solo i diritti di gestione delle chiavi ma non del patrimonio del UserSafe. Dopo la prima creazione, è possibile modificare i parametri dello UserSafe con un'autenticazione *FranceConnect*.

Lo UserSafe facendo parte di quella cerchia di attori che compongono le interazioni permesse da QAXH.IO, ha il diritto ad usufruire dei vari servizi *Folder*. Al fine di evitare un nuova creazione di UserSafe, per ogni servizio che viene integrato in maniera "sicura", l'implementazione si è evoluta seguendo il pattern abbastanza noto del "Diamond".

```
1
2  diamondcut_array = []
3     diamondcut_array.append(utils.set_contract_cut_array('Add',
    etagfacet_instance))
4     """
5     Facets
6     """
7     diamondcut_array.append(utils.set_contract_cut_array('Add',
    keymanagerfacet_instance))
```

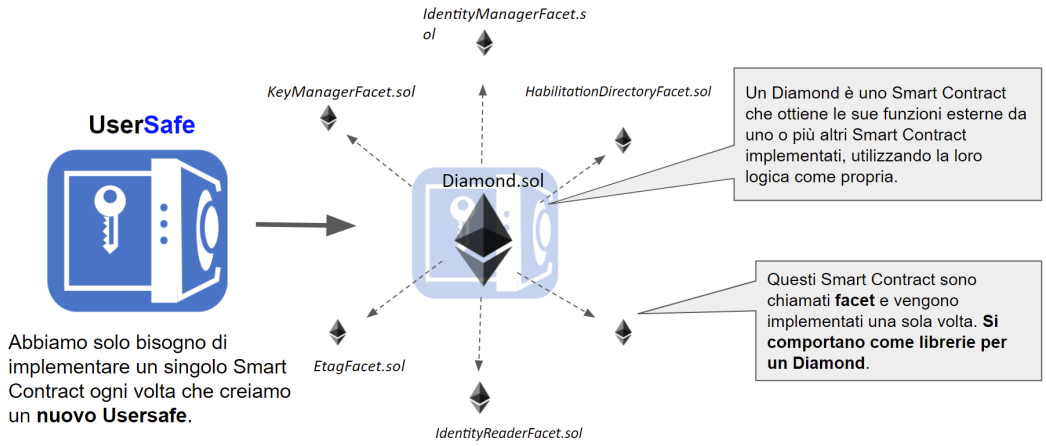

Figura 3.21: Usersafe Diamond

```
8
9       tx_hash = contract_class.constructor(diamondcut_array, [
        hatch_address, keyAddress, deadline, eIDAS, ageOfMajority,
        QI_hash, QE_hash, customerId]).transact({
10          'from': QAXH_ADDRESS,
11          'gas': 9000000,
12          'gasPrice': w3.toWei(gasPrice_in_gwei, 'gwei'),
13          'value': w3.toWei(0.02, "Ether")
14      })
```

Listing 3.3: Creazione del Diamond da parte del HatchServer

```
1   constructor(IDiamondCut.FacetCut[] memory _diamondCut, DiamondArgs
        memory _args) payable {
2
3           LibDiamond.diamondCut(_diamondCut, address(0), new bytes
        (0));
4           LibDiamond.setContractOwner(_args.parentAddress);
5           LibDiamond.setupUtils(_args.parentAddress);
6
7           LibIdentityManager.IdentityStorage storage identityStore =
         LibIdentityManager.identityStorage();
8           identityStore.safeType = 1;
9           identityStore.safeVersion = "9.3";
10          identityStore.checkHash = "None";
11          identityStore.QI_hash = _args.QI_hash;
12          identityStore.QE_hash = _args.QE_hash;
13          identityStore.safeActivation = 0;
14          identityStore.customerId = _args.customerId;
15          identityStore.identityLevel = _args.identityLevel;
16          identityStore.ageOfMajority = _args.ageOfMajority;
```

```
17
18          LibDiamond.DiamondStorage storage ds = LibDiamond.
    diamondStorage();
19          LibKeyManager.KeyStorage storage keystore = LibKeyManager.
    keyManagerStorage();
```

Listing 3.4: Contratto Diamond

```
1  enum FacetCutAction {Add, Replace, Remove}
2  struct FacetCut {
3          address facetAddress;
4          FacetCutAction action;
5          bytes4[] functionSelectors;
6      }
```

Listing 3.5: Strutture FacetCut

Folder-Type contract

Un contratto di tipo *Folder* è uno Smart Contract che interagirà con i "Safe" del framework Qaxh.io al fine di rispondere alle esigenze del business digitale. Una *ServiceFolder* è strutturata principalmente intorno a 3 componenti:

- Un insieme di variabili, che viene ridefinito alla creazione della serviceFolder e NON può essere modificato. Da notare come variabili obbligatorie.
  1. Un parentAddress.
  2. Un CID (id cliente).
  3. Un numero di versione dello smartContract.
  4. Un "safeType" che rappresenta la tipologia di *Folder*.
  5. Una variabile di attivazione, impostata a "0" da hatchServer, impostata da "1" (attivata) o "2" (chiusa definitivamente) dal parentAddress.
- Un elenco di coppie (indice, dati) dinamiche, facilmente customizzabili sia in fase di creazione che non.
- Una o più strutture dati (liste, struct, mapping) e funzioni utili a soddisfare i requisiti di business.

3.3.2 *HabilitationFolder*

Chiarita la definizione di *FolderType* Contract è possibile fornire i dettagli dell'implementazione del servizio di *Gestione dei poteri bancari*. Come già anticipato nella sezione 1.3 del capitolo 1 , questo sistema si basa sulla possibilità di creare da

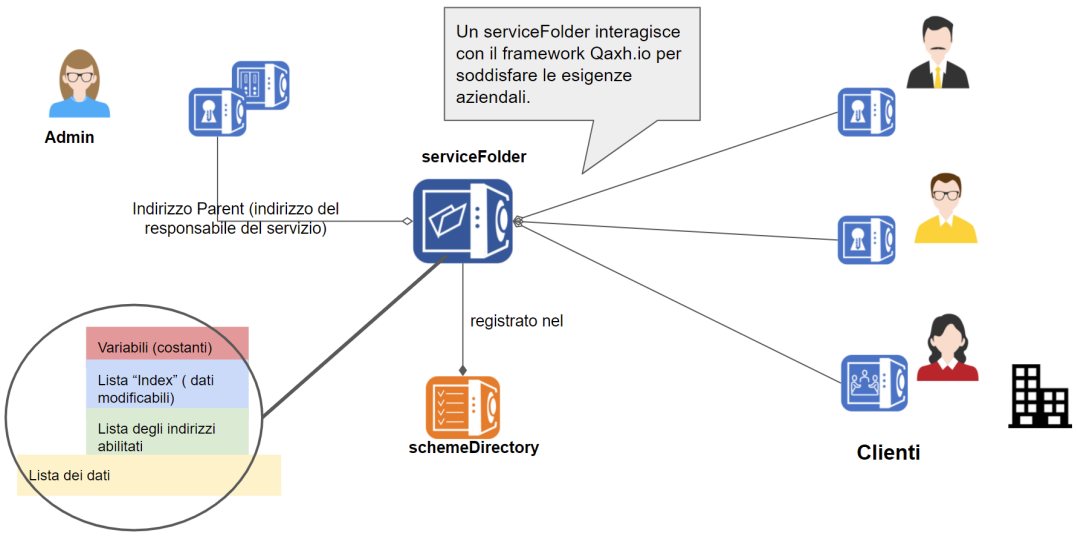

Figura 3.22: Contratto di una *Folder* in QAXH.IO

parte di un "Issuer" (utente che emette ordini) in collaborazione con un "Acquirer" (istituto che riceve ed elabora ordini, una banca nella maggior parte dei casi) un registro immutabile e condiviso per memorizzare gli ordini finanziari e tutte le sue caratteristiche.

Strutture & Librerie

```
contract HabilitationDirectory is Context {

  using SafeMath for uint256;
    /**************\
    Private Variables
/************************/

  address private ownerAddress;
  address private acquirerServer;
  address private parentAddress;
  address private issuerAddress;
  string public safeVersion = "01.04";
  uint private customerId;
  uint private safeType ;
  uint private issuerQB;
  string[] private IndexList;
  string[] private Categories;
  string private listCategory;
  string private listCurrencies;
```

```
20    uint private CountCategories;
21    uint private acquirerActivationStatus;
22    uint private issuerActivationStatus;
23    uint private CountIndexHabiliatedList;
24    uint private CountIndexOrders;
25
26    /* Lists for Orders, Issuer habilitated, limits for operations
       */
27
28    mapping (uint => LibHabilitation.Order) private orders;
29    mapping (address => LibHabilitation.HabStruct) private
       habilitatedAddressesInfo;
30    mapping (uint => address) private habilitatedAddressesList;
31    mapping (address => mapping (string => uint)) private
       habilitatedAddressAuthTrans;
```

Listing 3.6: Variabili della HabilitationFolder

Come ogni FolderType che si rispetti mantiene alcune classiche variabili come quella per l'attivazione (doppia in questo caso) o come il riferimento ad un *parentAddress*. Tuttavia, in tale contesto è risultato fondamentale inserire tra le variabili anche i vari indici delle molte liste che caratterizzavano il contratto. Tra i "mapping" implementati si nota:

- **orders**: Lista degli ordini del Folder.
- **habilitatedAddressesInfo**: Mapping che per ogni indirizzo restituisce le varie autorizzazioni di creazione/firma.
- **habilitatedAddressesList**: Lista degli Issuer abilitati a creare/firmare.
- **habilitatedAddressAuthTrans**: Doppio mapping che da un indirizzo e una categoria di ordini restituisce la somma massima autorizzata.

```
1  library LibHabilitation {
2  struct HabStruct {
3
4        uint encryptionKeyIndex;
5        bytes32 iban;
6        bool Hab_create;
7        bool Hab_sign;
8        uint valid ;
9        uint index ;
10   }
11
12   struct TypeAmount{
13      string cat;
14      uint amount;
15
```

```
16  }
17  struct Order {
18      bytes32 merkleHeader;
19      uint IndexTransaction;
20      uint IndexIssuer;
21      uint IndexAcquirer;
22      uint IndexHeader;
23      bool signed;
24      uint LastStatus;
25      mapping(uint => address) transactAddress;
26      mapping(uint => uint) VectorHeader;
27      mapping(uint => bytes32) Header;
28      mapping(uint => StatusOrder) IssStates;
29      mapping(uint=> StatusAcquirer) AcqStates;
30  }
```

Listing 3.7: Libreria HabilitationFolder

E' utile soffermarsi sull'insieme di strutture dati chiamate *VectorHeader, transactAddress, Header* in quanto all'interno dell'ordine le regole del business impongono una struttura Header, con una dimensione variabile, per ogni Address coinvolto nell'ordine (*transactAddress*). Tale struttura dalle dimensioni variabili è stata implementata sfruttando un mapping generale che raggruppi tutti gli headers dell'ordine spezzettati in formato fisso bytes32 [1] e un mapping Indice che permetta di ricostruire i vari pezzi di header per ogni indirizzo.

```
1  abstract contract Context {
2      // Empty internal constructor, to prevent people from
       mistakenly deploying
3      // an instance of this contract, which should be used via
       inheritance.
4      constructor () { }
5      // solhint-disable-previous-line no-empty-blocks
6
7      function _msgSender() internal view returns (address) {
8          return msg.sender;
9      }
10
11     function _msgData() internal view returns (bytes memory) {
```

[1] E' stata utilizzata questa struttura particolare per due ragioni:

- *Bytes32* è maggiormente efficente rispetto a *String*.
- L'acquisizione dei dati tramite funzione è gia gestita in tal maniera a causa del poco supporto di Solidity per i vettori di *String* (Doppio indirizzamento).

```
12          this; // silence state mutability warning without
    generating bytecode - see https://github.com/ethereum/solidity/
    issues/2691
13          return msg.data;
14      }
15
16      /* if you encode some different types -> you try abi.
    encodePacked(1Type,2type...) */
17
18          function _KeccakHashFromString(string memory _data, uint
    _i, string memory _M) public pure returns(bytes32)
19      {
20          return keccak256(abi.encodePacked(_uint2str(_i),_M,_data))
    ;
21
22      }
```

Listing 3.8: Ereditarietà del HabilitationFolder

```
1   modifier onlyParent() {
2       require(_msgSender() == parentAddress || _msgSender() ==
    acquirerServer, "Only parent or Acquirer server");
3       _;
4   }
5
6   modifier onlyIssuer() {
7       require(_msgSender() == issuerAddress, "Only Issuer");
8
9
10      _;
11  }
12
13
14  modifier PermissionToCreate() {
15      require(habilitatedAddressesInfo[_msgSender()].valid == 1, "
    Not habilitated");
16      require(habilitatedAddressesInfo[_msgSender()].Hab_create ==
    true, "Not allowed");
17      _;
18  }
19
20
21  modifier checkActive() {
22      require(issuerActivationStatus == 1 &&
    acquirerActivationStatus == 1, "Not activated");
23      _;
24
25  }
26
27  modifier checkCategory(string memory category) {
```

```
28      bool found = _checkElementInArrayString(category,Categories,
        CountCategories);
29      require(found == true, "This category is not in the folder ")
        ;
30      _;
31  }
```

Listing 3.9: Modifiers del HabilitationFolder

Costruttore & Attivazione

```
1   constructor(address _parentAddress,address _issuerAddress,uint
      _issuerQB, uint _customerId, string[] memory indexes, string
      memory _listCategory,string[] memory categories, string memory
      _listCurrencies , address _acquirerServer) {
2       require(_parentAddress != address(0),"Not Valid");
3       require(_issuerAddress != address(0),"Not Valid");
4       require(_customerId > 0 ,"Not valid customerId");
5       require(_issuerQB >= 0 ,"Not valid issuerQB");
6       require(indexes.length == 12 ,"Not valid indexes");
7
8       ownerAddress = _msgSender();
9       parentAddress = _parentAddress;
10      issuerAddress = _issuerAddress;
11      /*.....*/
```

Listing 3.10: Costruttore del HabilitationFolder

```
1   function ActivationSignIssuer(uint _status) public onlyIssuer {
2
3       require(_status == 1 || _status == 2, "  not allowed ");
4       issuerActivationStatus = _status;
5
6    }
7     // @notice: Change status folder activation by parent Address
8    function ActivationSignAcquirer(uint _status) public onlyParent{
9
10      require(_status == 1 || _status == 2, "  not allowed ");
11      acquirerActivationStatus = _status;
12
13   }
```

Listing 3.11: Funzioni di attivazione del HabilitationFolder

Indirizzi abilitati (Habilitated addresses)

	habilitedList L2 set by ONLY the issuerAddress	(keys authorised to manage the habilitation records)			
	habilitedAddress	encryptIndex	create	sign	encryptIBAN
	0X....	5	1(True)	1	iban
	0X....	7	1	0	
	0x	3	0(False)	1	

Figura 3.23: Struttura degli indirizzi abilitati

```
1  function IssuerCreate(address _habilitated, uint
       _encryptionKeyIndex, bytes32 _iban) public  onlyIssuer
       checkActive   {
2
3        require(_habilitated != address(0), "address not valid");
4        require(habilitatedAddressesInfo[_habilitated].Hab_create ==
        false ,"address already habilitated to create");
5
6
7        if(habilitatedAddressesInfo[_habilitated].valid == 1)
8        {
9            require(habilitatedAddressesInfo[_habilitated].
       encryptionKeyIndex == _encryptionKeyIndex , "Habilitation to
       create refused");
10           habilitatedAddressesInfo[_habilitated].Hab_create = true
        ;
11       }else {
12       habilitatedAddressesInfo[_habilitated] = LibHabilitation.
       HabStruct(_encryptionKeyIndex,_iban,true,false,1,
       CountIndexHabiliatedList);
13       habilitatedAddressesList[CountIndexHabiliatedList] =
       _habilitated;
14       CountIndexHabiliatedList = CountIndexHabiliatedList.add(1);
15     }
16   }
```

Listing 3.12: Funzione per autorizzare un indirizzo (issuer) alla creazione di un ordine

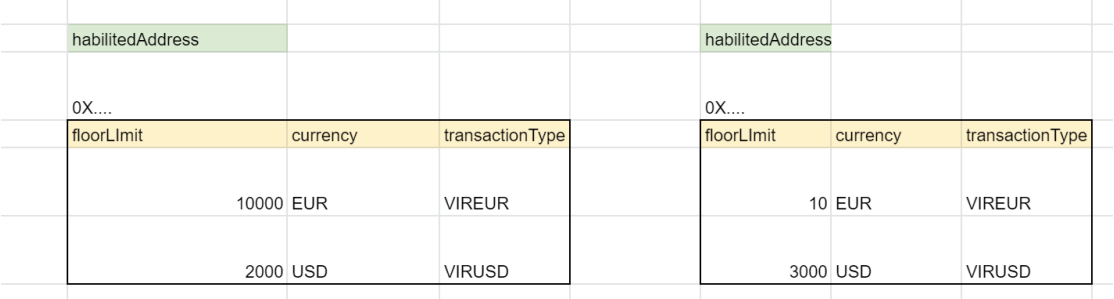

Figura 3.24: Indirizzi autorizzati con le caratteristiche delle varie categorie

E' possibile nuovamente sottolineare la gestione di un array di stringhe, che in fase di memorizzazione delle categorie (*IssuerSign*), è memorizzato come *String* (3.24) e in fase di restituzione/acquisizione è tradotto come array di *bytes32*, in quanto meglio supportato dal compilatore Solidity.

```
1  function IssuerGetData(address _habilitated) public valid(
    _habilitated) view returns(uint, bool, bool,string memory,
    bytes32[] memory, uint[] memory) {
2      LibHabilitation.HabStruct memory HabilitatedAddr =
    habilitatedAddressesInfo[_habilitated];
3      uint[] memory amount = new uint[](CountCategories);
4      bytes32[] memory _categories = new bytes32[](CountCategories)
    ;
5      //string[] memory cat = new string[](HabilitatedAddr.
    IndexCategories);
6      for(uint i = 0 ; i < amount.length ; i++)
7      {
8          amount[i] =  habilitatedAddressAuthTrans[_habilitated][
    Categories[i]];
9          _categories[i] = _stringToBytes32(Categories[i]);
10
11     }
12
13     return (HabilitatedAddr.encryptionKeyIndex,HabilitatedAddr.
    Hab_create,HabilitatedAddr.Hab_sign,_bytes32ToString(
    HabilitatedAddr.iban), _categories,amount);
14 }
```

Listing 3.13: Getting data of an Issuer 3.23

```
1  for(uint i = 0 ; i < _cat.length; i++)
2      {  require(_amount[i] >= 0, "Amount of Transactions not
    valid ");
3      habilitatedAddressAuthTrans[_habilitated][_bytes32ToString(
    _cat[i])] = _amount[i];
4
```

```
5       }
```

Listing 3.14: Codice di *IssuerSign* con *_bytes32ToString* 3.24

```
1  function _stringToBytes32(string memory source) internal pure
     returns (bytes32 result) {
2      bytes memory tempEmptyStringTest = bytes(source);
3      if (tempEmptyStringTest.length == 0) {
4          return 0x0;
5      }
6
7      assembly {
8          result := mload(add(source, 32))
9      }
10    }
11 function _bytes32ToString(bytes32 _bytes32) internal pure returns
     (string memory) {
12         uint8 i = 0;
13         while(i < 32 && _bytes32[i] != 0) {
14             i++;
15         }
16         bytes memory bytesArray = new bytes(i);
17         for (i = 0; i < 32 && _bytes32[i] != 0; i++) {
18             bytesArray[i] = _bytes32[i];
19         }
20         return string(bytesArray);
21     }
```

Listing 3.15: String to Bytes32 **Context.sol**

Creare o firmare un ordine (Create/Sign)

orderList (L3)								
orderIndex	merkleHeader	header sublist (L31) with L2 address and parentAddress	encrypted Data of the subList	issuer Status sublist (L32)	issuer TimeStamp Status	habilitatedAddress (who asks for this status)	acquirer Status sublist (L33)	acquirer TimeStamp Status
1	0Xabdcef1234	0x...	AZAZAZ	1 (created)		0x...	10 (creation acknowledge)	
		0x...	BRBRBR	2 (deleted)		0x...	11 (header file issue)	
		0x...	CGCGCG	3 (signed)		0x...	20 (deletion acknowledge)	
		0x...	FFFFFFF	4 (recall)		0x...	30 (sign acknowledge - no error)	
2							31 (header file issue)	
							32 (file not received)	

Figura 3.25: Lista degli ordini con delle sotto-liste

```
1  function OrderCreate(bytes32 _merkleheader,address[] memory
     _TransAddress, bytes32[] memory _TransData,uint[] memory
     _vectorHeader)  public checkActive PermissionToCreate  {
2
3
4        require(_TransAddress.length > 0 && _TransAddress.length ==
     _vectorHeader.length , "Not valid");
5        require(_TransData.length    == _vectorHeader[_TransAddress.
     length - 1], "Not valid vectors");
6        createOrderInternal(_merkleheader,_TransData,_TransAddress,
     _vectorHeader);
7
8
9      }
10
11       function createOrderInternal(bytes32  _m, bytes32[] memory
     _header, address[] memory _trans, uint[] memory _vectorHeader)
     private {
12                  LibHabilitation.Order storage r = orders[
     CountIndexOrders];
13                  CountIndexOrders = CountIndexOrders.add(1);
14
15
16                   r.merkleHeader = _m;
17       r.IndexTransaction = _trans.length;
18       r.IndexIssuer = 1;
19       r.IndexAcquirer = 0;
20       r.signed = false;
21       r.LastStatus = 1;
22       for(uint i = 0 ; i < r.IndexTransaction ; i++)
23       {
24           r.transactAddress[i] = _trans[i];
25           r.VectorHeader[i] = _vectorHeader[i];
26       }
27       for(uint i = 0; i < _header.length ; i++)
28       {
29           r.Header[i] = _header[i];
30       }
31       r.IndexHeader = _header.length;
32       r.IssStates[0] = LibHabilitation.StatusOrder(1,block.
     timestamp, _msgSender());
33
34
35
36         }
```

Listing 3.16: Creazione di un ordine 3.25

La funzione, mostrata nel *Listing 3.17*, permette la firma di un ordine esistente da parte di un indirizzo che ne ha il diritto (Modifier *ValidSign()*).

Nelle prime righe del codice è possibile notare i vari *require*:

- Controllo sull'autorizzazione per quella categoria.
- Controllo dell'importo autorizzato per quella categoria.
- Controllo del merkle header (1.2) : calcolo della stringa attraverso il parametro l'*M*, l'importo e la categoria, confronto con il *MerkleHeader* inserito in fase di creazione nel *Listing 3.16*.

Questi check a livello contrattuale, in particolare l'ultimo della precedente lista, permettono di esonerare l'azienda e soprattutto i ruoli dirigenziali, quali CEO o CFO, da un continuo controllo sulla correttezza dei dettagli di un ordine. Questo rimuove quasi totalmente la componente umana dall'elaborazione degli ordini, garantendo la precisione e il determinismo perfetto del codice all'interno dei processi bancari ed aziendali.

```
1  function OrderSign(uint _index, uint _amount, string memory
       _category,string memory _M) public checkActive checkCategory(
       _category)  checkIndex(_index,CountIndexOrders) validSign(
       _index) {
2
3      require(habilitatedAddressAuthTrans[_msgSender()][_category
       ]!= 0 ,"Category not auth");
4      require(habilitatedAddressAuthTrans[_msgSender()][_category]>
        _amount ,"Cat VALID,amount not valid");
5      require(orders[_index - 1].merkleHeader ==
       _KeccakHashFromString(_category,_amount,_M),"not valid merkle")
       ;
6
7      /* _KeccakHashFromString from Context.sol */
8
9
10     orders[_index - 1].LastStatus = 3;
11     orders[_index - 1].signed = true;
12     orders[_index - 1].IssStates[orders[_index - 1 ].IndexIssuer]
        = LibHabilitation.StatusOrder(3,block.timestamp, _msgSender())
       ;
13     orders[_index - 1 ].IndexIssuer = orders[_index - 1 ].
       IndexIssuer.add(1);
14   }
```

Listing 3.17: Firma di un ordine

```
1  struct StatusOrder {
2
3      uint issuerStatus;
4      uint IssuerTimeStampStatus;
5      address habilitatedAddress;
```

```
6
7   }
8   struct StatusAcquirer {
9       uint AcquStatus;
10      uint AcquirerTimeStampStatus;
11  }
```

Listing 3.18: Sotto liste in un ordine

La funzione mostrata nel seguente *Listing* è riservata all'istituzione della Folder Habilitation che prende in carico gli ordini. Il suo ruolo verrà approfondito nella sezione 3.4.

```
1   function OrderAcquirerAdd(uint _index, uint _newStatus) public
     onlyParent checkActive  checkIndex(_index,CountIndexOrders)  {
2
3
4       orders[_index - 1].AcqStates[orders[_index - 1 ].
     IndexAcquirer] = LibHabilitation.StatusAcquirer(_newStatus,
     block.timestamp);
5       orders[_index - 1 ].IndexAcquirer = orders[_index - 1 ].
     IndexAcquirer.add(1);
6     }
```

Listing 3.19: Aggiornamento di stato per un *ParentAddress*

HabilitationFolder - UserSafe

Al fine di integrare anche questo business service all'interno dello Usersafe, distribuito sulla piattaforma *Qaxh.io*, è necessario realizzare una *Facet* da aggiungere alla struttura Diamond ad ogni creazione. Le tappe principali di questa operazione, svolta dal *HatchServer*:

- Compilare e distribuire la Facet al fine di ricavare un .json con un indirizzo (*Truffle*).
- Recuperare un'istanza del contratto distribuito.
- Aggiungerlo alla struttura del *DiamondCut*.

In tal modo sarà possibile per il diamond creare una struttura che estragga i selettori delle funzioni di ogni sua Facet, proponendole come funzioni indipendenti per lo usersafe.

```
1
2   habilitation_facet_data = utils.deserialize_diamond_json("
     HabilitationDirectoryFacet")
3
```

```
4   habilitationfacet_instance = w3.eth.contract(address=utils.
     get_contract_address(habilitation_facet_data),abi=
     habilitation_facet_data['abi'])
5
6
7   diamondcut_array.append(utils.set_contract_cut_array('Add',
     habilitationfacet_instance))
```

Listing 3.20: Aggiunta di una Facet durante la creazione

Una "Facet" non è altro che un insieme di funzioni che richiamano le funzioni del contratto principale (*Activation, IssuerCreate...*), distribuito in precedenza, restando tuttavia all'interno della struttura "Safe".

```
1
2  import "../AccessController.sol";
3  import "../../HabilitationDirectory.sol";
4  import "../../LibHabilitation.sol";
5
6
7  contract HabilitationDirectoryFacet is AccessController {
8
9      ////////////////////      HABILITATION FUNCTIONS
     ////////////////////////
10
11     function ActivationUserSignIssuer(address _habilitateSafe,
     uint _status) public filterAndRefundOwner(false) {
12         HabilitationDirectory safe = HabilitationDirectory(
     _habilitateSafe);
13        safe.ActivationSignIssuer(_status);
14     }
15
16     function ActivationUserSignAcquirer(address _habilitateSafe,
     uint _status) public filterAndRefundOwner(false) {
17         HabilitationDirectory safe = HabilitationDirectory(
     _habilitateSafe);
18        safe.ActivationSignAcquirer(_status);
19     }
20
21     function IndexUserCreate(address _habilitateSafe) public
     filterAndRefundOwner(false) {
22         HabilitationDirectory safe = HabilitationDirectory(
     _habilitateSafe);
23        safe.indexCreate();
24     }
```

Listing 3.21: Facet Habilitation in Diamond-UserSafe

```
1
2    modifier filterAndRefundOwner(bool includeQaxh) {
```

```
3         LibKeyManager.KeyStorage storage keystore = LibKeyManager.
    keyManagerStorage();
4         require(((keystore.isKeyActive[msg.sender] == true) || (
    includeQaxh && (msg.sender == keystore.parentAddress))),
5                 "This method can only be called by the owner of
    the safe");
6         _;
7     }
```

Listing 3.22: Modifier from AccessControl.sol

3.4 Piattaforma Server

La piattaforma QAXH.IO, al fine di rendere i vari servizi maggiormente fruibili e dinamici, implementa alcune API, con il framework *Flask*, per ogni Folder/Safe che la contraddistingue. Queste application programming interface, implementate su alcune macchine messe a disposizione da *Natixis*, una controllata *BPCE*, permettono una completa interazione con i contratti distribuiti su Qaxh.io, fornendo i strumenti principali a chiunque voglia creare e distribuire *Safe* completi [4] [29].

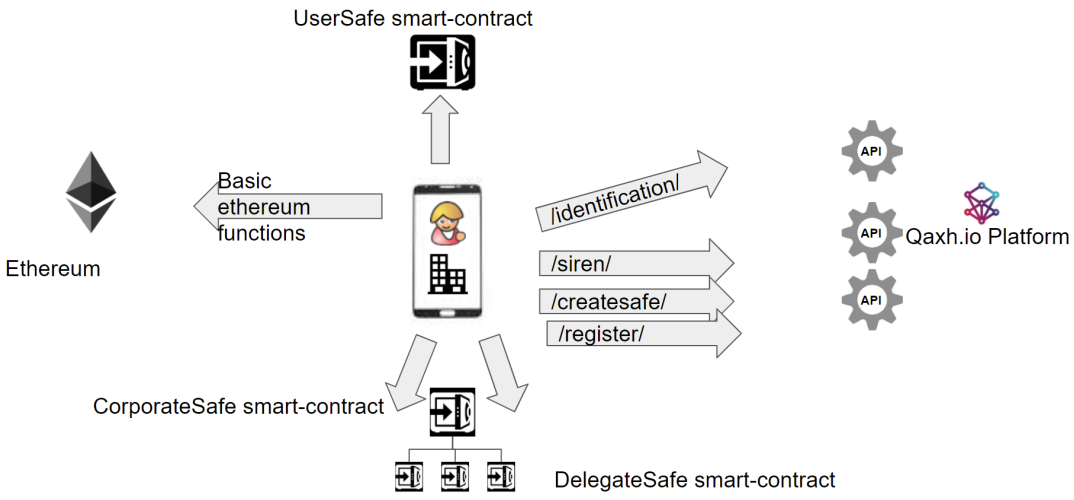

Figura 3.26: API in QAXH.IO

3.4.1 Server *QAXH.IO*

E' possibile riportare una panoramica di alcune funzionalità offerte dalla piattaforma che corrispondano ad un rispettiva API:

Principali API della piattaforma Qaxh.io

- ***user.qaxh.io/testid/id?***:
  Creazione di uno *UserSafe* attraverso *Mobile Connect* e *France Connect*.
- ***user.qaxh.io/testid/idregister?***
  Registrazione dentro lo *SchemeDirectory* dello *UserSafe*.
- ***deployment.qaxh.io/deploy/emoneycreate?***:
  Creazione di una moneta interoperabile.
- ***corporate.qaxh.io/pdf/signinit?***:
  Creazione di un sistema di firme per documenti aziendali.
- ***deployment.qaxh.io/hatch/hatchcreate?***:
  Creazione di un *HatchSafe*.
- ***deployment.qaxh.io/hatch/schemecreate?***:
  Creazione di un *SchemeDirectory*.

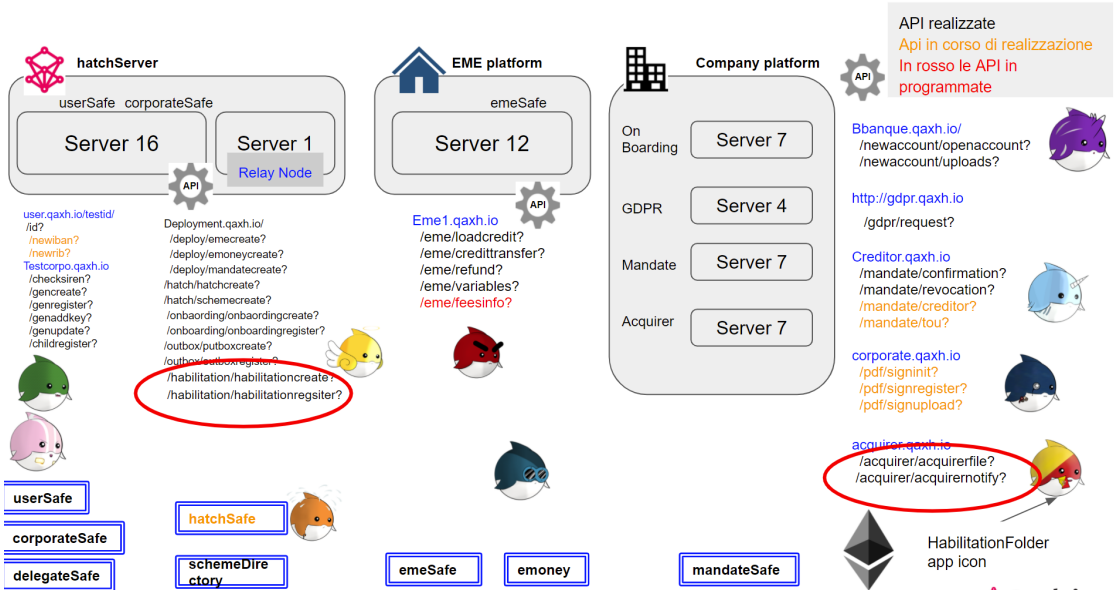

Figura 3.27: API Overview

Principali API-Services Qaxh.io

- ***deployment.qaxh.io/onbaording/onbaordingcreate?***:
  Creazione della Folder *OnBoarding*.

- ***deployment.qaxh.io/deploy/mandatecreate?***:
  Creazione del servizio di *Mandate*.

- ***deployment.qaxh.io/habilitation/habilitationcreate?***:
  Creazione della Folder *Habilitation*.

- ***creditor.qaxh.io/mandate/confirmation?***:
  Api di servizio di conferma per un addebito all'interno di *Mandate*.

- ***acquirer.qaxh.io /acquirer/acquirernotify?***:
  Api di servizio per un acquirer per ricevere un quadro aggiornato su alcuni ordini di un determinato contratto di *HabilitationFolder*.

Tutte le API mostrate in precedenza sono state implementate in *Python* sfruttando la libreria, messa a disposizione dalla *Ethereum Fondation*, *Web3py*. La libreria in questione permette di inviare transazioni, interagire con i contratti intelligenti, leggerea dei dati dei blocchi e una varietà di altri casi d'uso; tuttavia l'intero funzionamento della libreria si basa sull'impostazione da parte del programmatore del punto di accesso alla rete Blockchain scelta. Su tale punto focale è possibile distinguere due metodologie di accesso, che corrispondono a due tecnologie differenti, sfruttate in *QAXH.IO*:

- **Infura**: un'applicazione che si connette alla blockchain di Ethereum, interagisce con la blockchain di Ethereum e gestisce i nodi per conto dei suoi utenti [32]. Nata per risolvere i principali problemi di qualsiasi connessione alla blockchain:

  1. La memorizzazione dei dati su Ethereum è costosa.
  2. Può essere complessa l'impostazione di una connessione alla blockchain.
  3. La sincronizzazione con la blockchain può risultare lenta.
  4. La struttura della blockchain occupa molto spazio.

  Molte applicazioni basate su Ethereum si affidano a Infura per connettersi alla blockchain di Ethereum ed effettuare transazioni per i propri utenti. Ma Infura è un servizio centralizzato ed è quindi vulnerabile ad attacchi che potrebbero limitarne la funzionalità e la correttezza. Inoltre questa centralizzazione potrebbe essere utilizzata per censurare le transazioni da parte di governi o terze parti.

  Più servizi lo utilizzano, maggiormente centralizza la blockchain di Ethereum attorno a una società. Questo andando contro l'idea stessa di decentralizzazione, con cui nasce la blockchain, ha portato a sviluppare soluzioni poco centralizzate attraverso alcuni particolari strumenti come *Geth*.

- **Nodo Geth**: Il nodo Geth [35] è un vero e proprio nodo della blockchain, sviluppato su uno *screen* linux del server *Natixis*, vedi la figura 3.13. Questo nodo permette gli scenari piu comuni all'interno di Ethereum:

  1. Creazione di account.
  2. Trasferimento di asset.
  3. Distribuzione ed interazione con i contratti.

  Tuttavia è pensato anche per i classici scenari per gli sviluppatori :

  1. Permette l'accesso alle reti di test piu comuni.
  2. Permette la creazione di una propria rete interna.
  3. Permette la gestione di un *Miner* interno.

  Questo tool sviluppato dalla fondazione Ethereum fornisce ulteriori possibilità ad un neofita della Blockchain, sostituendo e integrandosi perfettamente con tecnologie eistenti come *Metamask*.

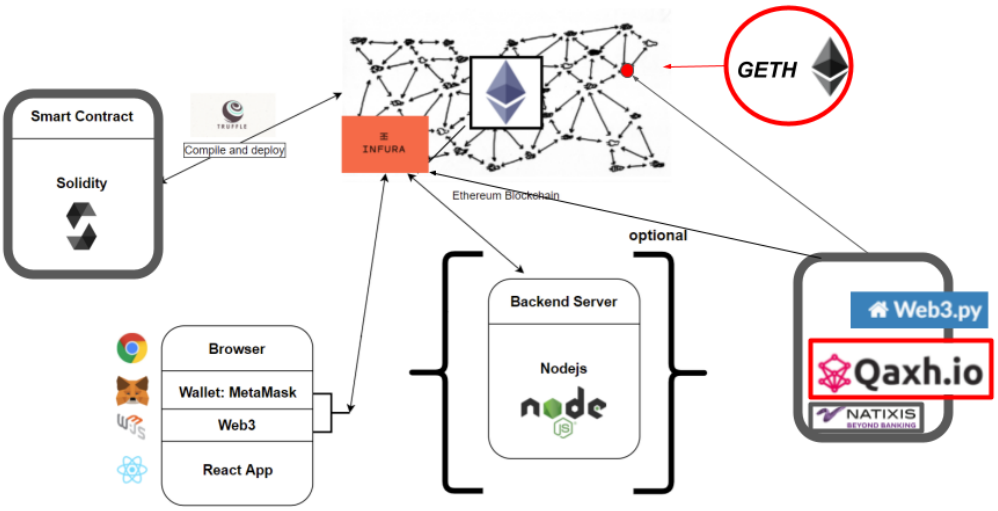

Figura 3.28: Overview dei punti d'accesso

Punto di accesso alla Blockchain

```
infura_url = "https://rinkeby.infura.io/v3/<Infuralink>"

logging.info('web3 connection')

web3 = Web3(Web3.HTTPProvider(infura_url))
```

Listing 3.23: Classica connessione tramite Infura

```
class W3Connection:
    class __W3Connection:
        def __init__(self):
            path = os.path.join(PATH_DIR_NODE, "geth.ipc")
            provider = Web3.IPCProvider(path, True, timeout=10)
            #provider = Web3.HTTPProvider("http://rinkeby.qaxh.io/")
            #An attempt to implement a tool like infura internal to Qaxh.io

            if provider is None:
                raise Exception(f"Couldn't connect to geth node via IPC at path {path}")
            self.w3 = Web3(provider)

            # XXX: This is only necessary when using rinkeby
            self.w3.middleware_stack.inject(middleware.geth_poa_middleware, layer=0)

    __instance = None

    def __init__(self):
        if not W3Connection.__instance:
            W3Connection.__instance = W3Connection.__W3Connection()

    def __getattr__(self, attr):
        return getattr(W3Connection.__instance.w3, attr)
```

Listing 3.24: Nodo geth *(usato sul Server)*

Key for Hatch & Scheme creation

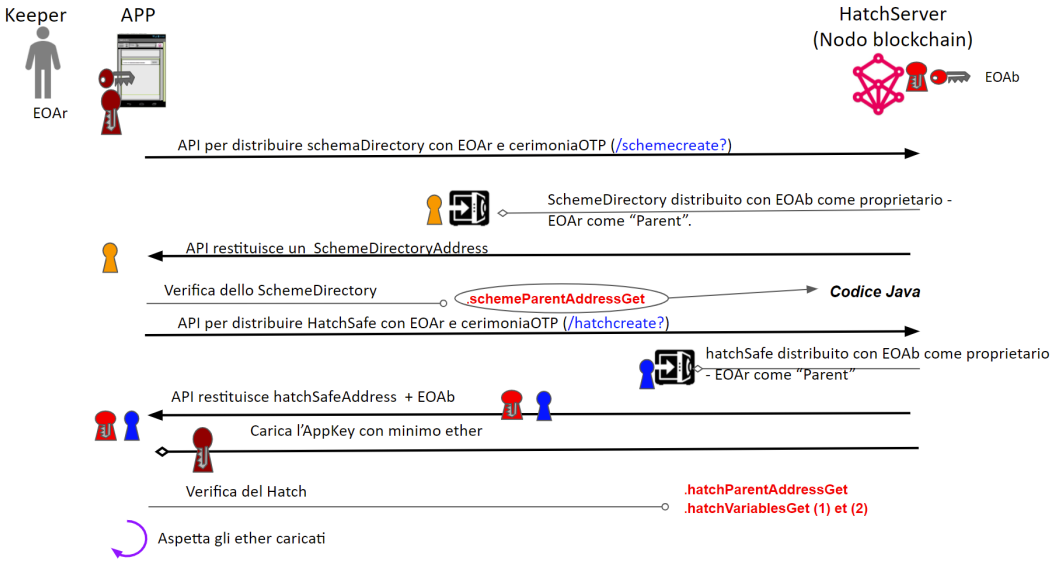

Figura 3.29: Scheme & Hatch creation

```
@app.route("/hatch/schemecreate", methods=["POST"])
def schemecreate ():

    logging.info("[SCHEME CREATE]")
    response = {}
    response["tx_hash"] = ""

    otp = str(request.args.get('ceremonyotp'))

    result = db.get_otp_validity(otp)
   """ check result
   """
    db.set_otp_used(result)

    eoar = request.args.get('eoar')

   """
   Checksum Eoar for web3
   """
    info_contract = deploy_Scheme_contract(parentAddress)
    return jsonify(info_contract)
```

Listing 3.25: Creazione dello scheme con OTP & eoar

```
@app.route("/hatch/hatchcreate", methods=["POST"])
```

```
2  def hatchcreate():
3
4      """
5      setting response...
6
7      Otp operations...
8
9      Extract arguments as ParentAddress and HatchName..
10
11     """
12      #deployment of HatchSafe
13      info_contract = deploy_HatchSafe_contract(parentAddress, mode,
       hatchName, urlId, urlRegister, otpId)
14
15      return jsonify(info_contract)
```

Listing 3.26: Creazione del Hatch (stesso modello dello Scheme)

Diamond Server

L'implementazione del Diamond Server, ovvero il server per la creazione di un'identità "Safe"in QAXH.IO, è anch'esso implementato in python, sfruttando il framework *Flask*. Quest'ultimo permette l'importazione di un toolkit di nome *Blueprint* sfruttato per estendere le funzionalità offerte dalle API di creazione, quali *autenticazione* e *accounting*. L'implementazione della famiglia delle API *user.qaxh.io/testid/* esula dallo scopo di tale tesi quindi è possibile limitarsi a riportare come alla base del funzionamento vi sia un *Redirect HTTP* al fine di reindirizzare la richiesta di autenticazione al provider di identità digitale *Mobile Connect et moi*, già citato negli scorsi capitoli. Per quanto riguarda la creazione di un *UserSafe* è possibile ricollegarci alla sezione 3.3.1, mostrando per intero la costruzione di un *Diamond-Identità* a partire dalle varie *Facets*, già distribuite sulla Blockchain:

1. Compilazione di una *Facet* con conseguente creazione di un file JSON.
2. Distribuzione di una *Facet* su Ethereum attraverso tool come *Truffle* (le metodologie verranno approfondite nel prossimo capitolo).
3. Estrazione dell'indirizzo appena distribuito dal file JSON.
4. Estrazione dell'*ABI* dal file JSON.
5. Creazione di un'istanza del contratto a partire dagli ultimi due elementi estratti.
6. Creazione di un array di strutture che raggruppi le coppie Indirizzo-ABI.

7. Creazione del *Diamond* con l'array descritto nel *6* come parametro di input.

```
1   habilitation_facet_data = utils.deserialize_diamond_json("
    HabilitationDirectoryFacet")
2
3
4   habilitationfacet_instance = w3.eth.contract(address=utils.
    get_contract_address(habilitation_facet_data), abi=
    habilitation_facet_data['abi'])
5
6   # preparing the DiamondCut Array
7   diamondcut_array.append(utils.set_contract_cut_array('Add',
    habilitationfacet_instance))
8   """
9   other facets..
10  """
11
12  diamond_data = utils.deserialize_diamond_json("Diamond")
13  diamond_abi = diamond_data["abi"]
14  diamond_bytecode = diamond_data["bytecode"]
15  contract_class = w3.eth.contract(abi=diamond_abi , bytecode=
    diamond_bytecode)
16
17  logging.info("\nUnlocking ...\n")
18  w3.geth.personal.unlockAccount(QAXH_ADDRESS, QAXH_PASSWORD)
19  logging.info("\nDeploying Diamond Usersafe ...\n")
20  logging.info("\nDiamondcut_array:" + json.dumps(
    diamondcut_array) + "\n")
21  tx_hash = contract_class.constructor(diamondcut_array, [
    hatch_address, keyAddress, deadline, eIDAS, ageOfMajority,
    QI_hash, QE_hash, customerId]).transact({
22      'from': QAXH_ADDRESS,
23      'gas': 9000000,
24      'gasPrice': w3.toWei(gasPrice_in_gwei, 'gwei'),
25      'value': w3.toWei(0.02, "Ether")
26  })
27  logging.info(f"Waiting for Diamond userSafe deployment {
    tx_hash.hex()}")
28  w3.geth.personal.lockAccount(QAXH_ADDRESS)
```

Listing 3.27: Diamond steps

Folder-Type Server

Il Server-side per i business services delle Folder è principalmente basato su due API principali:

- ***/qaxh.service/foldercreate***: API di creazione del contratto che riceve come argomenti i vari parametri da inserire nel costruttore del contratto.

- ***/qaxh.service/folderregister***: API di registrazione della Folder all'interno dello *SchemeDirectory*.

Come già visto nel *Listing 3.27* in questa sottosezione è utile mostrare l'implementazione classica delle strutture *Web3* che permette una transazione all'interno della blockchain.

```
    # extract abi from .sol
    hatch_abi = utils.deserialize_json("HatchSafe")["abi"]

    # Instance contract from Address and ABI
    scheme = w3.eth.contract(schemeAddr, abi=hatch_abi)

    # sign the transaction
    #  t = func.buildTransaction(
    #   {'nonce': .. ,'from': '0xE5c6dEA048EB', 'gas': 6200000, '
gasPrice': w.eth.gasPrice})
    # signed_tx = w.eth.account.signTransaction(t, KeyPrivate)
    # tx = w.eth.sendRawTransaction(signed_tx.rawTransaction)
    # It's the same with "unlockAccount" for account of node geth
in the server
    w3.personal.unlockAccount(QAXH_ADDRESS, QAXH_PASSWORD)

    # call a function with the parameters
    tx_hash = scheme.functions.registerSafe(safe, safeType, CID,
0).transact(
        {'from': QAXH_ADDRESS, 'gas': 6200000, 'gasPrice': w3.eth.
gasPrice})

    w3.personal.lockAccount(QAXH_ADDRESS)
```

Listing 3.28: Codice per la registrazione nello Scheme

In ambito di Folder-Type è ragionevole riportare le differenze tra le API del server n.16 e le API del server n.7 in quanto il primo rappresenta il vero server della piattaforma qaxh.io, un vero "HatchServer", al contrario il secondo rappresenta un'altro protagonista dei servizi di ogni *Folder*, l'*Acquirer Server*. Su quest'ultimo sono implementate tutte le API fornite e pensate da QAXH.IO per gli actor-Acquirer, quali banche, istituzioni ed entità governative. Quest'ultimi grazie a questi strumenti implementati sui loro server, mantenendo un semplice indirizzo Ethereum, potranno gestire tutti i servizi delle Folder (Smart Contract) che li vedono protagonisti, usufrendo di tutti gli elementi tipici della blockchain, quali autorizzazione e confidenzialità.

3.4.2 Gestione dei poteri bancari

Procedendo nello stesso modo della sezione precedente, è possibile fornire i dettagli dell'implementazione server del servizio di *Gestione dei poteri bancari*.

Server 16 - Creazione:
/habilitation/habilitationcreate?

```
1     data = utils.deserialize_json(name)
2     contract_class = w3.eth.contract(abi=data["abi"] , bytecode=
      data["bytecode"])
3
4
5     # Send the contract deployment transaction and retrieve its
      tx_hash:
6
7     nonce = w3.eth.getTransactionCount(QAXH_ADDRESS)
8     w3.personal.unlockAccount(QAXH_ADDRESS, QAXH_PASSWORD)
9     tx_hash = contract_class.constructor(parentAddress,
      issuerAddress, qb, customerId, indexes, categoryList,
      categories, currencies, acquirerAddress).transact({'from':
      QAXH_ADDRESS, 'gas': 6200000, 'gasPrice' : w3.eth.gasPrice})
10    w3.personal.lockAccount(QAXH_ADDRESS)
```

Listing 3.29: Codice per la creazione

```
[INFO/root:53] 2021-07-07 11:06:46 (MainProcess/Thread-13)
    > Start deploying contract HabilitationDirectory from : 0x872B5b37D829fC03Ad
D37e6395a30E5B00f04b6b

[INFO/root:37] 2021-07-07 11:06:47 (MainProcess/Thread-13)
    > Start deploying contract : HabilitationDirectory (tx_hash: 0x718cf01df22fc
c4b675f82be2ca7a43fc152c6c06909e267a47b250c3e446d6a)

[INFO/root:69] 2021-07-07 11:07:07 (MainProcess/Thread-13)
    > Deployed contract `HabilitationDirectory` at address 0x6C0FA6D915fA7b61D70
62b837f0F4438Ee56E6C8

[INFO/root:74] 2021-07-07 11:07:07 (MainProcess/Thread-13)
    > Deployed all contracts in 0:00:21.210466

[INFO/werkzeug:225] 2021-07-07 11:07:07 (MainProcess/Thread-13)
    > 127.0.0.1 - - [07/Jul/2021 11:07:07] "POST /habilitation/habilitationcreat
e?parentaddress=0x872B5b37D829fC03AdD37e6395a30E5B00f04b6b&CID=12&issueraddress=
0x872B5b37D829fC03AdD37e6395a30E5B00f04b6b&acquireraddress=0xDaf5BcFB2FaF1100924
7582E176B64D590fa5259&issuerqb=1&categories=VIREUR;VIRGBP&currencies=EUR;GBP&ind
ex1=https%3A%2F%2Fdocs.google.com%2Fdocument%2Fd%2F1kS8tkiovdjpJISgF_tVtyGN1OJdF
XALObK5iZjHyFRo%2Fedit%3Fusp%3Dsharing&index2=Banque+Populaire+Alsace+-Lorraine-
Champagne&index3=0&index4=logo-14707.png&index5=Pouvoirs+Bancaires&index6=http%3
A%2F%2Facquirer.qaxh.io%2Facquirer%2Facquirerfile%3F&index7=http%3A%2F%2Facquire
r.qaxh.io%2Facquirer%2Facquirernotify%3F&index8=https%3A%2F%2Fdocs.google.com%2F
document%2Fd%2F1kS8tkiovdjpJISgF_tVtyGN1OJdFXALObK5iZjHyFRo%2Fedit%3Fusp%3Dshari
ng&index9=1.7&index10=1%2F4%2F2021&index11=0x12345&index12=0x HTTP/1.0" 200 -
```

Figura 3.30: Log file

Server 16 - Registrazione:
/habilitation/habilitationregister?

```
1     hatch_abi = utils.deserialize_json("HatchSafe")["abi"]
2     scheme = w3.eth.contract(schemeAddr, abi=hatch_abi)
3     w3.personal.unlockAccount(QAXH_ADDRESS, QAXH_PASSWORD)
4     tx_hash = scheme.functions.registerSafe(safe, safeType, CID,
    0).transact(
5         {'from': QAXH_ADDRESS, 'gas': 6200000, 'gasPrice': w3.eth.
    gasPrice})
6     w3.personal.lockAccount(QAXH_ADDRESS)
7     logging.info(f"transaction before mining send {tx_hash.hex()}"
    )
8     #Wait that the transaction is mined
9     # is better that the server doesn't wait the receipt ( next
    two lines )
10    tx_receipt = w3.eth.waitForTransactionReceipt(tx_hash.hex(),
    2000)
11    tx_receipt = tx_receipt["transactionHash"]
```

Listing 3.30: Codice per la regitrazione nello Scheme

E' utile soffermarsi sulla linea 10 e 11 di codice del *Listing 3.30* in quanto attendono che la transazione specificata da *tx_hash* venga inclusa in un blocco (allo stesso modo la creazione). Tuttavia in un'ipotetica versione di produzione di questo servizio, questa attesa dovrà essere eliminata in quanto il compito di aspettare e verificare il buon esito della transazione spetterà allo *User* al fine di evitare possibili attacchi DOS sul server.

Server 7 - Gestione documenti: */acquirer/acquirerfile?*

```
1  #  numpy / pandas libraries used for reading and managing file
     data
2
3  logging.info('Calculate Merkle and check with the Merkle in the
     contract')
4     file = {'ref': rows[:, 0], 'date': [s[:10] for s in rows[:,
     1]], 'amount': [int(x) for x in rows[:, 4]],'currency':rows[:,
     5],'category': rows[:, 6]}
5
6     for c in file['category']:
7         if c not in listCategory:
8             return [471,tx_hashs]
9
10
11    for c in file['currency']:
12        if c not in listCurrency:
13            return [472,tx_hashs]
14
15
```

```
16      file = ps.DataFrame(file, columns=['ref', 'date', 'amount', '
    category'])
17
18      index = 0
19      for h in Headers:
20          head = h.split(';')
21          date = head[1][:10]
22          cat = head[2]
23          totalAmount = int(head[3])
24          res = list(itertools.accumulate(list(file[(file.category
    == cat) & (file.date == date)]['amount']), operator.add))[-1]
25          if res != totalAmount:
26              return [474,tx_hashs]
27
28          merkle = ContractHab.functions.OrderGetMerkleHeader(orders
    [index]).call()
```

Listing 3.31: Codice per la gestione degli ordini da parte del server bancario

Server 7 - Notifica:
/acquirer/acquirernotify?

```
1   for Index in orders:
2           [OrdersStatus, Time, Addr] = ContractHab.functions.
    OrderGetHabilitated(Index).call()
3           #return last [OrdersStatus, Time, Addr]
4           bod.append({'index_Order': Index ,'Status': OrdersStatus
    [-1], 'Timestamp': Time[-1], 'Address': Addr[-1]})
```

Listing 3.32: Codice per la lettura degli ordini

```
sergio_giardina@nodejs-1-vm-7:~$ sudo su - qaxh_admin
qaxh_admin@nodejs-1-vm-7:~$ curl "http://acquirer.qaxh.io/acquirernotif
y?habilitation_folder=0xDa8C5ce6F20894383274EB9d07d496e351686fF5&order_index=1;2
"
{
  "body": "Last Issuer for index :1 0xE5c6C2704C32F4a79B32092977f3cF1adEA048EB;3
;1624956114\nLast Issuer for index :2 0xE5c6C2704C32F4a79B32092977f3cF1adEA048EB
;1;1624955813\n",
  "return_code": 200
}
```

Figura 3.31: Richiesta Http di notifica

3.5 User experience

L'ultimo elemento da approfondire in questo capitolo riguarda la creazione di un'applicazione web. L'idea che sta alla base della *User experience* del framework è

quella di creare rapidamente un prototipo di applicazione web tramite App Inventor, strumento molto semplice che con alcune basilari conoscenze di programmazione consente di creare rapidamente un'applicazione personalizzata. App Inventor è un semplice ambiente di sviluppo per applicazioni Android, creato da Google, ma ora di proprietà del Massachusetts Institute of Technology. Questo ambiente di sviluppo è stato ideato appositamente per gli individui che desiderino programmare semplici applicazioni Android per uso personale. Queste app possono essere pubblicate anche sul Google Store. Durante lo sviluppo , invece, è possibile installare le applicazioni (file *.apk*) inviandole direttamente al cellulare/tablet tramite WiFi o USB, oppure utilizzare la versione online di App Inventor direttamente sullo smartphone/tablet. Per quanto riguarda la *User experience* della piattaforma QAXH.IO è fondamentale rammentare come l'idea di decentralizzazione, caratterizzante della blockchain, è perseguita anche per tale modello. A dimostrazione di ciò, gli sviluppatori del progetto hanno puntato a separare i vari servizi e gestioni di entità "Safe" in applicazioni differenti, totalmente indipendenti l'una dall'altra. Tale approccio oltre a garantire uno sviluppo delle singole applicazioni piu veloce ed efficente dal punto di vista tecnico, si dimostra vincente anche dal punto di vista commerciale in quanto è possibile costruire profili cliente diversi a partire da applicazioni diverse.

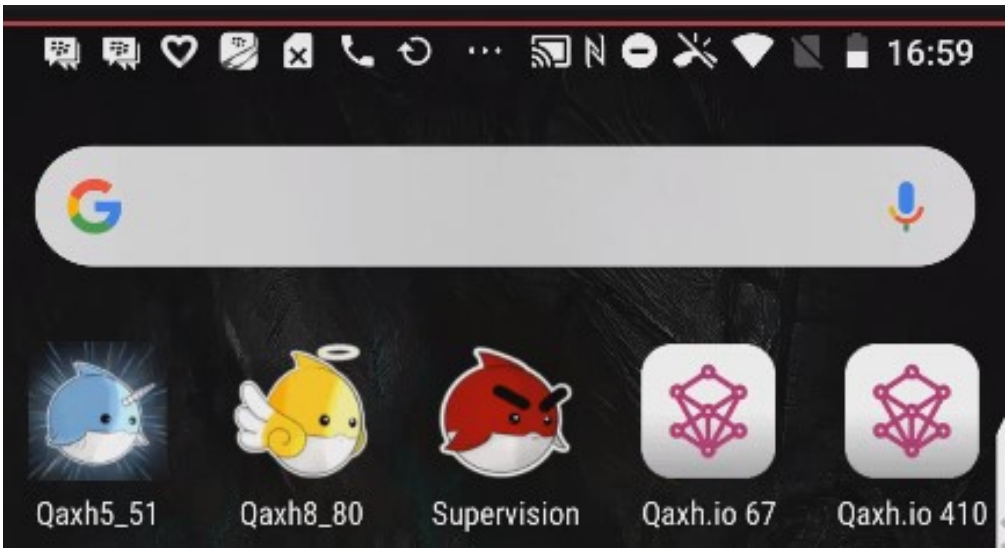

Figura 3.32: Icone App Qaxh.io

Al fine di mantenere un buon livello di controllo per gli amministratori della piattaforma si è provveduto anche allo sviluppo di alcune applicazioni per i *Keepers* con autorizzazioni di vario livello.

3.5.1 Frontend: *AI 2*

Questa modalità di lavoro coinvolge da un lato App Inventor che garantisce il layout e la costruzione grafica di una pagina comprensibile per gli utenti, dall'altro codice Java (o per meglio dire funzioni Java) che utilizzando un modulo fornito da Ethereum (*Web3j*) interagisce con i vari contratti della blockchain, semplicemente partendo dai parametri forniti come l'indirizzo di un Wallet blockchain. Partendo da questa idea, è possibile costruire la logica di un'applicazione procedendo alla composizione di alcuni blocchi funzionali indipendenti [36]:

- Blocchi semplici 3.33(Controllo, Logici, Testuali, Liste).

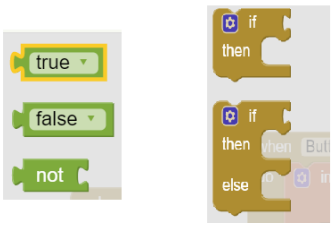

Figura 3.33: Funzioni logiche

- Blocchi di procedure complesse fornite da estensioni (*Qaxh_Eth* 3.37).

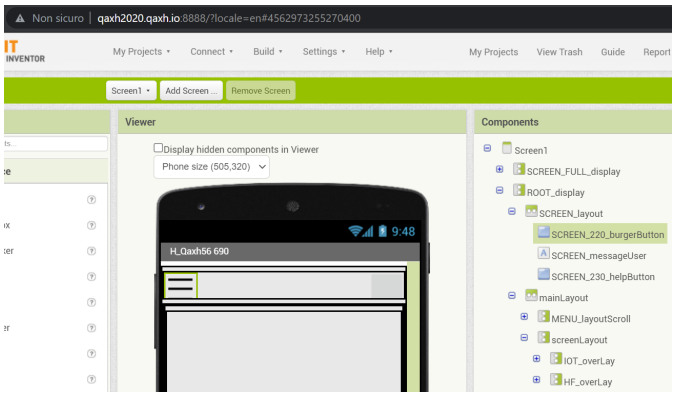

Figura 3.34: Server Web Qaxh.io per App Inventor

Nella figura 3.35 è possibile distinguere partendo dall'alto verso il basso, e procedendo da sinistra verso destra, i seguenti elementi:

- Il testo "Mining in corso" per un'operazione in corso sulla blockchain.
- L'indirizzo del contratto di *Habilitation Folder* già distribuito su *Rinkeby*.

Figura 3.35: Screenshot - *Register folder*

- Il QRCode dei dati sulla destra prodotto con una delle estensioni citate nella sezione 3.5.2
- I dati della *Folder* letti attraverso la funzione *habilitationGetVariables* riportata nel *Listing 3.36*, nella sezione 3.5.3.
- L'indirizzo dello *SchemeDirectory* non ancora settato.
- Il bottone di attivazione per l'*Acquirer* che usa la funzione *habilitationAcquirerActivationUserSign* riportata nel *Listing 3.36*, nella sezione 3.5.3..
- Il tasto che permette la registrazione nello *SchemeDirectory* che è rappresentanto dal blocco logico riportato nella figura 3.36.

- Il bottone per condividere il QRCode all'esterno dell'applicazione.
- Il tasto che permette di accedere alla pagina di *Update* dei dati della *Folder*.

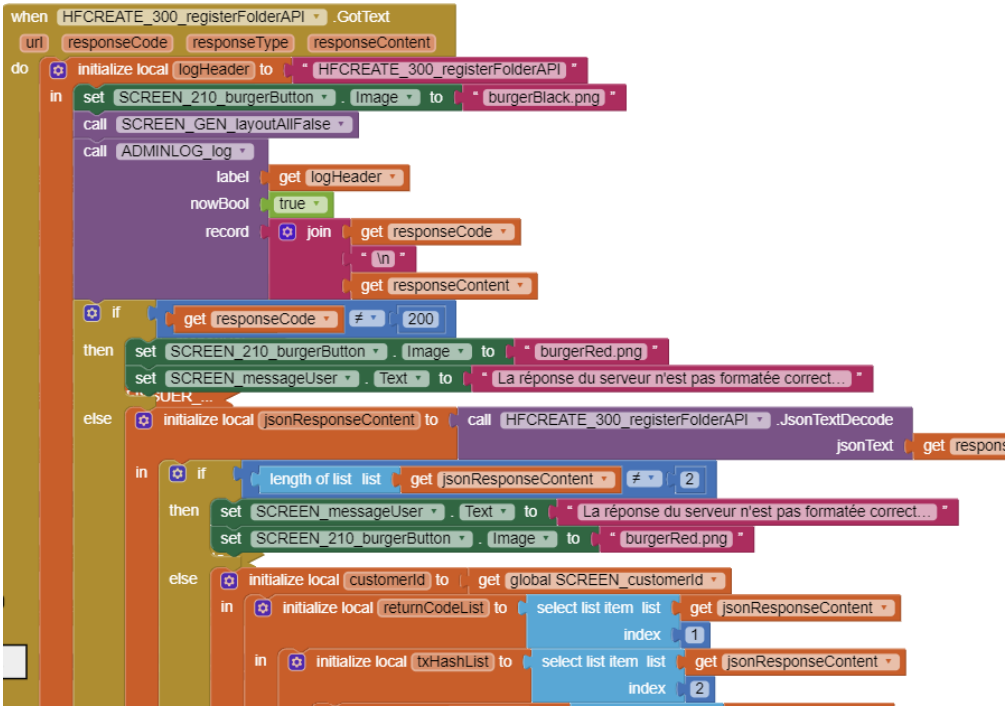

Figura 3.36: Blocchi per registrare una folder

3.5.2 Backend: Java-blocchi

Il backend di un'applicazione QAXH.IO si basa sulla possibilità di inserire un'estensione nell'ambiente App Inventor che permette di implementare del codice "esterno" (3.37) richiamato dai diversi "blocchi" (una variabile o una funzione in AI è chiamato blocco) d'App Inventor, figura 3.15. [37]

L'utilizzo di queste estensioni rende notevolmente complessa l'intera compilazione di App Inventor, per tale ragione, allo scopo di produzioni di app Qaxh.io, è stato sviluppato un server web che fornisce uno "speciale" web IDE App Inventor 3.34. Quest'ultimo basandosi sul codice open source, messo a disposizione dal *MIT*, supera alcune limitazioni di compilazione permettendo l'inserimento di codice e librerie esterne in maniera totalmente libera e customizzabile (*Doppia Compilazione* 3.5.3). Tale strumento ha permesso nel corso degli anni lo sviluppo di varie estensioni, fornendo la quasi totalità di funzioni basilari che caratterizzano il sistema Qaxh.io:

- Utilizzo della fotocamera per l'acquisizione di QRCode.

Figura 3.37: Funzioni di *Qaxh_eth*

- La produzione di QRCode univoci a partire da alcuni dati.
- L'acquisizione dei dati testuali a partire dall'immagine di un testo.
- La verifica di dati e firma di uno specifico documento.
- L'interazione e la lettura di dati sulla blockchain.
- La cifratura di dati, la creazione e la distribuzioni delle chiavi private.

Le funzionalità descritte in precedenza sono implementate in estensioni diverse:

- ***ActivityOCR***: Strumenti per OCR.
- ***Qaxh_ETH***: Blocchi per l'interazione con la blockchain Ethereum o una sua TestNet.
- ***Qaxh_QR***: Blocchi per la produzione di codici QR a partire da uno o piu dati combinati.
- ***Qaxh_JPG***: Sfrutta l'OCR per riconoscere documenti specifici.

3.5.3 I blocchi *Qaxh_ETH*

```
1  public class QAXH_Eth extends AndroidNonvisibleComponent
       implements Component {
2
3      // VARIABLES
4
5      private static final String LOG_TAG = "QaxhEthComponent";
6      private Web3j web3;
7      private BigInteger nonce;
8      private String privHexKey;
9      private BigInteger gasLimit;
10     private BigInteger gasPrice;
11
12     // CONSTRUCTOR
13
14     /**
15      * Creates a QAXH_Eth component.
16      *
17      * @param container container, component will be placed in
18      */
19     public QAXH_Eth(ComponentContainer container) {
20         super(container.$form());
21         StrictMode.ThreadPolicy policy = new StrictMode.
       ThreadPolicy.Builder().permitAll().build();
22         StrictMode.setThreadPolicy(policy);
23
24         Security.insertProviderAt(new org.spongycastle.jce.
       provider.BouncyCastleProvider(), 1);
25
26         /**
27         * Used Web3j provided by Ethereum Foundation
28         *
29         * Used Infura node in the web application for the access
       to the Blockchain
30         */
31         web3 = Web3jFactory.build(new HttpService("https://rinkeby
       .infura.io/v3/<Infura-link>"));
32         gasLimit = BigInteger.valueOf(500000); // default gas
       limit
33         nonce = null;
34
35         try {
36             gasPrice = web3.ethGasPrice().send().getGasPrice();
37         }
38         catch (IOException e) {
39             e.printStackTrace();
40         }
```

```
41      }
```

Listing 3.33: Codice per il costruttore di classe QAXH_Eth

```
1  /**
2       * Setup the private key and nonce private variables used in
     the extension.
3       *
4       * @param privHexKey appInventor private key.
5       */
6       // @SimpleFunction is a command for App Inventor
7      @SimpleFunction(description="Setup the private key and nonce
     references in QAXH_Auth extension.")
8      public void blockchainCreateAccount(String privHexKey) {
9          /****/
10     }
11
12     /**
13      * Generate an ethereum private / public key pair.
14      *
15      * @return the keys and address of the account, in format : 0x
     <privateKey> /0x04 <publicKeys> /0x <adress>
16      */
17     @SimpleFunction (description = "Generate an ethereum private /
     public key pair.")
18     public String blockchainCreateKeyTriplet() {
19         ECPublicKey publicKey;
20         ECPrivateKey privateKey;
21         /****/
22
23         return "0x" + priv.toString(16) + "/0x04" + pubX.toString
     (16) +  pubY.toString(16) + "/0x" + Keys.getAddress(pubX.
     toString(16) + pubY.toString(16));
24     }
```

Listing 3.34: Codice per funzioni generali della blockchain

```
1  // Private functions
2
3      private List callViewFunction(String contractAddress, Function
     function) {
4          Credentials credentials = Credentials.create(this.
     privHexKey);
5          String address = credentials.getAddress();
6          String encodedFunction = FunctionEncoder.encode(function);
7          Transaction transaction = Transaction.
     createEthCallTransaction(address, contractAddress,
     encodedFunction);
8
9          try {
```

```
10              EthCall response = web3.ethCall(transaction,
    DefaultBlockParameterName.LATEST).sendAsync().get();
11              return FunctionReturnDecoder.decode(response.getValue
    (), function.getOutputParameters());
12          } catch (Exception e) {
13              e.printStackTrace();
14          }
15
16          return null;
17      }
18
19      private String callNonViewFunction(String contractAddress,
    Function function) {
20          Credentials credentials = Credentials.create(this.
    privHexKey);
21          try {
22              String encodedFunction = FunctionEncoder.encode(
    function);
23              RawTransaction rawTransaction = RawTransaction.
    createTransaction(nonce, gasPrice, gasLimit, contractAddress,
    encodedFunction);
24              byte[] signedMessage = TransactionEncoder.signMessage(
    rawTransaction, credentials);
25              String hexValue = Numeric.toHexString(signedMessage);
26              String txHashLocal = Hash.sha3(hexValue);
27              EthSendTransaction ethSendTransaction = web3.
    ethSendRawTransaction(hexValue).send();
28              nonce = nonce.add(BigInteger.valueOf(1));
29              return txHashLocal;
30          } catch (Exception e) {
31              e.printStackTrace();
32              return "error: callNonViewFunction";
33          }
34      }
```

Listing 3.35: Codice per funzioni interne (call/execute) nel contratto

Qaxh_ETH per *HabilitationFolder*

```
1   @SimpleFunction(description="USERSAFE: activation Acquirer")
2     public String habilitationAcquirerActivationUserSign(String
    userSafeAddress, String HabilitationAddress, String newStatus){
3           String txHash =  null;
4           String QaxhModule = getQaxhModule(userSafeAddress);
5
6           Function UserSign = new Function(
7                   "ActivationUserSignAcquirer",
8                   Arrays.<Type>asList(
```

```
                        new Address(HabilitationAddress),
                        new Uint256(new BigInteger(newStatus))
                ),
                Collections.<TypeReference<?>>emptyList()
        );

        try {
            txHash = callNonViewFunction(QaxhModule, UserSign);

            return txHash;
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return txHash;

    }

@SimpleFunction(description="Get Variables of Habilitation")
            public List<String> habilitationGetVariables(String
    HabilitationAddress) {
        List<String> result = new ArrayList<String>();
       /* getVariables in the "HabilitationAddress" */
        Function getVariables = new Function(
                "getVariables",
                Collections.<Type>emptyList(),
                Arrays.<TypeReference<?>>asList(new TypeReference<
    Address>(){},
                        new TypeReference<Address>(){},
                        new TypeReference<Uint256>(){}
                        ,new TypeReference<Uint256>(){},
                        new TypeReference<Uint256>(){},
                        new TypeReference<Utf8String>(){},
                        new TypeReference<Utf8String>(){},
                        new TypeReference<Utf8String>(){},
                        new TypeReference<Uint256>(){},
                        new TypeReference<Uint256>(){},
                        new TypeReference<Address>() {}
                )
        );

        List<Type> st = callViewFunction(HabilitationAddress,
    getVariables);
        result.add(st.get(0).toString());
        result.add(st.get(1).toString());
        result.add(st.get(2).getValue().toString());
        result.add(st.get(3).getValue().toString());
        result.add(st.get(4).getValue().toString());
        result.add(st.get(5).toString());
```

```
55          result.add(st.get(6).toString());
56          result.add(st.get(7).toString());
57          result.add(st.get(8).getValue().toString());
58          result.add(st.get(9).getValue().toString());
59          result.add(st.get(10).toString());
60          return(result);
61      }
```

Listing 3.36: Codice per attivazione e lettura di variabili

E' possibile notare nel seguente *Listing*, come già anticipato nella sezione 3.3.2, l'uso degli array di *Bytes32* in sostituzione dei classici array di *String*. Inoltre, è curioso osservare come il secondo ambiente di compilazione *App Inventor* vada ad influenzare il codice Java compilato nell'estensione, ad esempio la dimensiona degli array che hanno un'indicizzazione che parte da 1 (linee 6,7 del *Listing 3.37*).

```
1   @SimpleFunction(description="USERSAFE : Make Address safe
    enabled to sign")
2    public String habilitationIssuerAddressUserSign (String
    userSafeAddress, String HabilitationAddress,String Habilitated,
    String KeyInd, List<String> categories , List<String> amounts,
    String Iban) {
3         String QaxhModule = getQaxhModule(userSafeAddress);
4         String txHash =  null;
5         int i ;
6         int SizeListAmount = amounts.size() /* - 1 */ ; // Size
    with App inventor -> SizeListAmount - 1 ( so if you test with
    Maven  you must add - 1 here !!!)
7         int SizeListCategories = categories.size() /* - 1 */; //
    Size with App Inventor -> SizeListCategories - 1 ( so if you
    test with Maven you must add - 1 here!!! )
8         //  Java with App inventor : First Element is in List[1]
    and last element is in List[List.size()]
9
10        ArrayList<Uint256> ListAmount = new ArrayList<>();
11        for(i = 1 ; i <= SizeListAmount ;i++)
12        {
13            ListAmount.add(new Uint256(new BigInteger(amounts.get(
    i))));}
14
15        ArrayList<Bytes32> ListCategories = new ArrayList<Bytes32
    >();
16        for(i = 1 ; i <= SizeListCategories ;i++)
17        {
18            ListCategories.add(JavaStringToHexString32B(categories
    .get(i)));
19
20        }
21        /****  txHash = callNonViewFunction... *****/
22
```

```
23
24      @SimpleFunction(description="Get data of Issuer Habilitated.")
25      public List<String> habilitationIssuerGetData(String
    habilitationAddress, String Habilitated)
26      {
27      /*****/
28
29      }
```

Listing 3.37: Code piece for enabling to sign an address / reading data of address habilitated

Nel seguente *Listing* è riportata una funzione di cifratura utilizzata all'interno della *Business Folder*, totalmente indipendente dall'ambiente *Ethereum*.

```
1  @SimpleFunction(description="Encrypt Data from PublicKey")
2     public String blockchainEciesEncryptData(String PublicKey,
    String plaintext){
3          byte[] PublicKeyBytes;
4          // PublicKey.getBytes(StandardCharsets.UTF_8);
5          if(checkHex(PublicKey))
6          {
7              PublicKey = HexStringToAsciiString(PublicKey);
8              PublicKeyBytes = PublicKey.getBytes(StandardCharsets.
    UTF_8);
9              }
10
11         else {
12             PublicKeyBytes = PublicKey.getBytes(StandardCharsets.
    UTF_8);
13         }
14         byte[] keyBytes = Base64.getDecoder().decode(
    PublicKeyBytes);
15         X509EncodedKeySpec spec = new X509EncodedKeySpec(keyBytes)
    ;
16         try {
17             KeyFactory keyFactory = KeyFactory.getInstance("ECDSA"
    , "SC");
18             ECPublicKey key = (ECPublicKey) keyFactory.
    generatePublic(spec);
19             Cipher dCipher = Cipher.getInstance("ECIESwithAES");
20             dCipher.init(Cipher.ENCRYPT_MODE, key);
21             byte[] priTText = dCipher.doFinal(plaintext.getBytes(
    StandardCharsets.UTF_8));
22             return "0x"+bytesToHex(priTText);
23         }
24         catch(Exception e) {
25             e.printStackTrace();
26             return null;
27         }
```

```
28      }
```

Listing 3.38: Codice per cifratura ECIES

Double Compilation

In tale progetto è nota l'implementazione di una *doppia compilazione* da parte del server web Qaxh.io-App Inventor , figura 3.38:

- 1°-*Compilatore App inventor*: Struttura di App Inventor con tutte le sue componenti grafiche.
- 2°-*Ant Apache*: Compilazione Java con l'inserimento di alcune librerie specifiche per l'interazione con Ethereum come Web3j per Android.

Tali passaggi consentono una costruzione "semplificata" di un'applicazione totalmente customizzata con le potenzialità logiche di una vera e propria applicazione mobile.

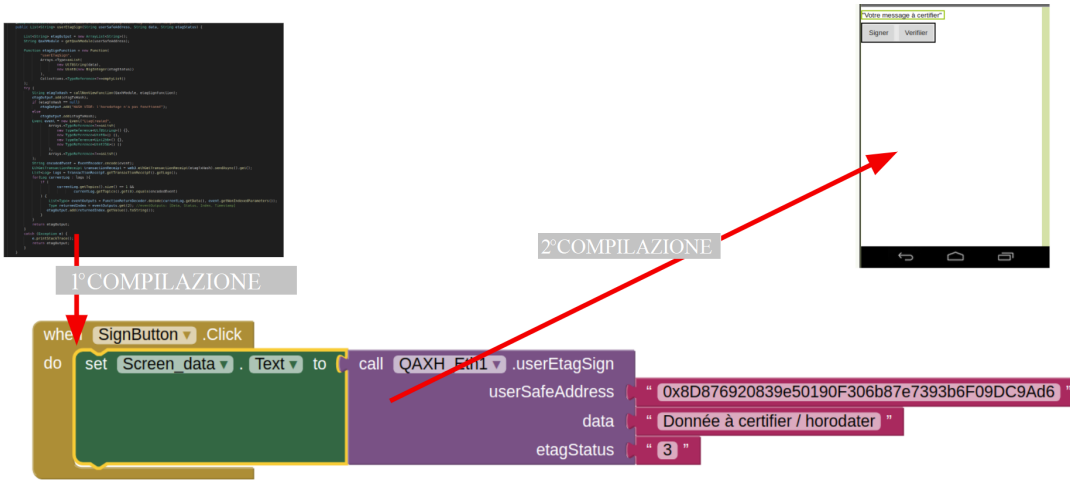

Figura 3.38: Doppia compilazione

3.6 Analisi dei costi

Nella sezione seguente, si riporta un'analisi dei costi degli elementi necessari al progetto QAXH.IO e in specifico alla *Folder* di gestione dei poteri bancari.

3.6.1 Costi strutturali

Il costo di utilizzo di alcuni server può essere analizzato a partire da due approcci differenti:

- Server dedicato (come nel caso di QAXH.IO con i server *Natixis*).
- Server condivisi sul *Cloud*.

Un server dedicato richiede da un lato degli esperti in grado di gestire e amministrare l'hardware fisico e dall'altra una gestione attenta e precisa delle operazioni di manutenzione. Al contrario per le soluzioni *Cloud* la manutenzione fisica dell'hardware è di competenza esclusiva del provider, che gestisce autonomamente il gruppo server. In aggiunta, dal momento che il server virtuale è ridondato su molte macchine, è pressoché impossibile interromperne il funzionamento anche a fronte di gravi guasti.

Ultimo aspetto da affrontare (ma uno dei più rilevanti) è quello dei costi. In genere le soluzioni cloud sono più convenienti proprio in virtù di un'economia di scala: la potenza è facilmente modulabile e si paga solo ciò che effettivamente si utilizza secondo modelli *pay-per-use*. Eppure, al salire delle richieste di banda e risorse, il vantaggio economico del cloud rispetto alle soluzioni dedicate si assottiglia, mentre questi ultimi presentano alti costi iniziali che possono essere ammortizzati nel tempo. Provando ad ipotizzare la miglior soluzione per un ambiente indipendente come QAXH.IO, che sfrutta i server per operazioni complesse ma isolate (Creazione,Registrazione ecc), è possibile individuare un server cloud come la soluione più adatta ed efficente.

3.6.2 Ethereum

I prezzi della rete Ethereum sono correlati ai costi delle commissioni di transazione, che non dipendono che dalla quantità di dati della transazione e dal prezzo del gas in ether, nel momento dell'intervento sulla Blockchain. Il prezzo per ogni unità di gas definisce il tempo della transazione in quanto un tempo di conferma per una transazione molto dilatato consumerà meno gas. Il progetto QAXH.IO, nel suo specifico utilizzo della *Folder* dei poteri bancari, esegue varie operazioni, coinvolgendo attori diversi; ciò porta il prezzo della transazione a dipendere da molti fattori. La seguente analisi dei prezzi è correlata al valore attuale dell'Ether che è di 2,076.18 euro.

Al fine di riportare un dato approssimativo dei costi di transazioni di un servizio completo di Gestione dei poteri bancari per un'azienda o una banca (con un'identità "Safe"), è utile procedere con le seguenti assunzioni sul numero di transazioni:

- 1 Creazione.

Gas Price	**Confirmation Time**	**Transfer Price**
17 gwei - low	> 5 minutes	$ 0.74
23 gwei - average	<= 2 minuti	$ 0.96
27 gwei - high	30 seconds	$ 1.09

Tabella 3.2: Tabella di confronto Gwei - Price al 09-07-2021 [38] [39]

- 1 Registrazione.
- 2 Attivazioni.
- circa 10 dati indice modificati due volte -> 20 Transazioni di *Update*
- 5 indirizzi da abilitare per Creazione/Firma ordini , considerandole sempre come operazioni distinte son 10 transazioni.
- 30 Ordini a settimana , almeno 4 al giorno. Almeno 3 operazioni per ordine (Creazione, Firma, Recall ecc).
- Gestione file per la banca: 3 operazioni di aggiornamento degli stati dell'ordine per ogni ordine firmato dall'azienda.
- Le letture non costano gas.

Estendendo in maniera grossolana questi numeri ad un intero anno si prevede un numero di almeno 8000-10000 transazioni annuali. Trascurando i costi delle strutture server in quanto dipendenti dall'infrastruttura scelta e condivisibili tra le varie *Folder*, il costo complessivo dell'esito delle transazioni per un *HabilitationFolder* è di 10000x0,74 = 7400 $ per azienda.

Capitolo 4

Risultati

4.1 Obiettivi conseguiti

Gli obiettivi conseguiti con il lavoro di tesi in questione possono essere valutati sia per natura tecnica, logica e non ultima di business. L'obiettivo principale al quale tendere è quello di rendere maggiormente efficente, praticamente in maniera automitazzata, l'intera gestione dei poteri bancari, garantendo allo stesso tempo le caratteristiche di affidabilità ed immutabilità, intrinseche alla Blockchain [40]. L'obiettivo può essere analizzato da prospettive molto diverse:

- **Obiettivo tecnico**
  1. *Sicurezza*: E' stata possibile implementando dei Smart Contract, con determinati vincoli, che seguendo la regola del *CODE IS LAW* impediscano automaticamente ad utenti non autorizzati di eseguire certe operazioni.
  2. *Tracciabilità*: E' stato possibile affidandosi al sistema transazionistico della Blockchain che tiene traccia di tutte le operazioni bancarie riferite al sistema, e conserva, inoltre, le singole operazioni di ogni utente.
- **Obiettivo logico**
  1. *Semplificazione* : l'obiettivo è semplificare il sistema di gestione dei poteri bancari; tutti i passaggi all'interno del sistema sono gestiti come transazioni, archiviati nel ledger Blockchain ed aggiornati attraverso precisi e veloci Smart contract.
  2. Sostenibilità: l'intero processo mira a supportare la sostenibilità. Se da un lato il ricorso alla tecnologia Blockchain, molto dispensiosa energeticamente, può sembrare un controsenso, la sostenibilità sta nell'utilizzo di un sistema che sostituisce moltissime mansioni umani permettendo all'azienda un compromesso costo-benefici sostenibile.

3. Automazione: l'utilizzo dei Smart Contract permette l'automazione di moltissime operazioni, minimizzando i possibili errori e/o interventi umani.

- **Obiettivo di Business** [41]

1. *Redditività*: Creare reddito significa fare in modo che le entrate rimangano al di sopra dei costi. Dunque, il controllo dei costi grazie al sistema descritto è notevolmente migliorato.
2. *Produttività*: l'utilizzo di tecnologie distribuite e totalmente autonome permette una minor preoccupazione della manutenzione della strumentazione/ formazione dei dipendenti, garantendo una maggior produttività.
3. *Gestione del cambiamento*: La gestione del cambiamento, in questo caso evoluzione dei sistemi classici, è una strategia che prepara l'impresa alla crescita e alla creazione di processi che vadano ad incidere efficacemente in un mercato in continuo sviluppo.

4.2 Verifica dell'efficienza : Ambienti di testing

Una volta terminata l'analisi tecnica del lavoro di tesi nel capitolo precedente, occorre verificare non solo l'aderenza di questo alle specifiche progettuali (per valutare se esistano mancanze o imprecisioni), ma occorre analizzare aspetti come la qualità del software in generale.

Questa sezione è dedicata alla verifica dell'efficienza. Quest'ultima può essere facilmente valutata analizzando le caratteristiche del prodotto e verificando l'esattezza delle sue risposte agli stimoli esterni (valori inseriti, calcoli richiesti ecc.), eventuali anomalie possono essere corrette dagli stessi sviluppatori, una volta individuatene le cause, e quindi occorre includere solitamente importanti controlli sulle prestazioni, robustezza e modularità del prodotto. Date le diverse tecnologie, sfruttate per la realizzazione del progetto, ne conseguono svariate verifiche dell'efficienza.

4.2.1 Smart Contracts

Per quanto rigurda il testing del *Core* dell'intero progetto sono stati utilizzati strumenti diversi, alcuni già accennati nella sezione 3.1.2:

- *Remix.*
- *MyCrypto.*
- *Truffle.*

Il primo non è che un IDE Solidity per la compilazione/ distribuzione con l'utilizzo di *MetaMask*. Permette un testing manuale delle varie funzioni. Allo stesso modo *MyCrypto*, leggermente piu indipendente dalla compilazione di Remix, permette un testing efficiente ma limitato. *Truffle*, invece, è uno strumento completo in ambito di gestione degli Smart Contract permettendo, come già descritto nel 3.1.2, in maniera molto dinamica la distribuzione, la compilazione, l'interazione e il testing. Quest'ultimo è permesso grazie alla realizzazione, e conseguente esecuzione da parte di Truffle, di particolari script *Javascript* che distribuiscono un contratto con pre-stabiliti parametri e ne chiamano le varie funzioni, controllando l'efficienza attraverso alcuni comandi di *Assert.equal*.

```
1  const ownedAddress = "0x919df9FCa073D9281B59c6eeD20C0820c45D1Dc7";
2  const issuerAddress= "0x919df9FCa073D9281B59c6eeD20C0820c45D1Dc7";
3
4
5  const HabilitationDirectory = artifacts.require("
       HabilitationDirectory");
6
7
8  module.exports = async function(deployer) {
9   await deployer.deploy(HabilitationDirectory, ownedAddress,
       issuerAddress,0, 12, ["https://docs.google.com/document/d/1
       kS8tkiovdjpJISgF_tVtyGN1OJdFXALObK5iZjHyFRo/edit?usp=sharing","
       Banque de Flux","0","ce1.png","Pouvoirs Bancaires","https://
       deployment.qaxh.io/habiliation/acquirefile?","https://
       deployment.qaxh.io/habilitation/acquirenotification","https://
       docs.google.com/document/d/1kS8tkiovdjpJISgF","1.7","1/4/2021",
       "0x12345"]);
10 }
```

Listing 4.1: Codice per la distribuzione di un HabilitationFolder

```
1  const HabilitationFolder = artifacts.require('./
       HabilitationDirectory.sol');
2  const truffleAssert = require('truffle-assertions');
3
4  let tryCatch = require("./exceptions.js").tryCatch;
5  let errTypes = require("./exceptions.js").errTypes;
6
7  contract('Checking deployement of the Habilitation Folder',
       function (accounts) {
8
9      const owner = "0x919df9FCa073D9281B59c6eeD20C0820c45D1Dc7";
10
11     before(async function() {
12         Habilitation = await HabilitationFolder.deployed();
13     });
14
```

```
15      it("Get last index", async function(){
16          let getLast = await Habilitation.indexGetLast();
17
18    assert.equal(getLast, 12);
19      })
20
21      it("Create index", async function(){
22          let create = await Habilitation.indexCreate();
23          let indexGetLast = await Habilitation.indexGetLast();
24
25    assert.equal(indexGetLast, 13);
26      })
27
28      it("Update index and index get data", async function(){
29          let update = await Habilitation.indexUpdate(1, "Coucou");
30          let indexData = await Habilitation.indexGetData(1);
31
32    //console.log(indexData);
33    assert.equal(indexData, "Coucou");
34      })
35
36      it("Check for variables", async function(){
37          let variables = await Habilitation.geVariables();
38
39    assert.equal(variables[0], owner);
40          assert.equal(variables[3], 43);
41      })
42
43      /* Other tests...*/
```

Listing 4.2: Codice per un test HabilitationFolder

4.2.2 *Qaxh_eth* Java

Passando al codice Java è utile mostrare un diverso ambiente di testing. In questo caso i test delle funzioni Java, realizzate per l'interazioni classiche tra App Inventor e la Blockchain, sono stati implementati grazie all'IDE JetBrains *Intellij Idea* e all'uso del framework *Apache Maven*.

Quest'ultimo è uno strumento di gestione di progetti software basati su Java e build automation. Per funzionalità è similare ad Apache Ant, ma basato su concetti differenti. Maven usa un costrutto conosciuto come Project Object Model (POM); un file XML che descrive le dipendenze fra il progetto, le varie versioni di librerie necessarie nonché le dipendenze fra di esse. In questo modo si separano le librerie dalla directory di progetto utilizzando questo file descrittivo per definirne le relazioni. Maven effettua automaticamente il download di librerie Java e plug-in Maven dai vari repository definiti, scaricandoli in locale o in un repository

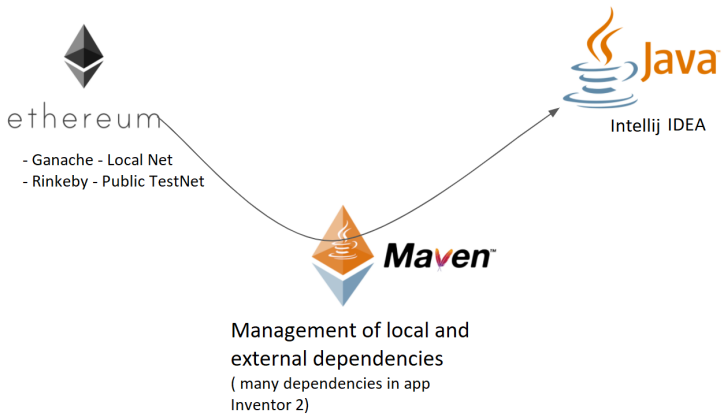

Figura 4.1: Testing *Qaxh_Eth*

centralizzato. Questo permette di recuperare in modo uniforme i vari file JAR e di poter spostare il progetto indipendentemente da un ambiente all'altro, avendo la sicurezza di utilizzare sempre le stesse versioni delle librerie. Questo framework insieme all'utilizzo di un nodo *Infura* permette un totale *debugging* di tutte le funzionalità realizzate .

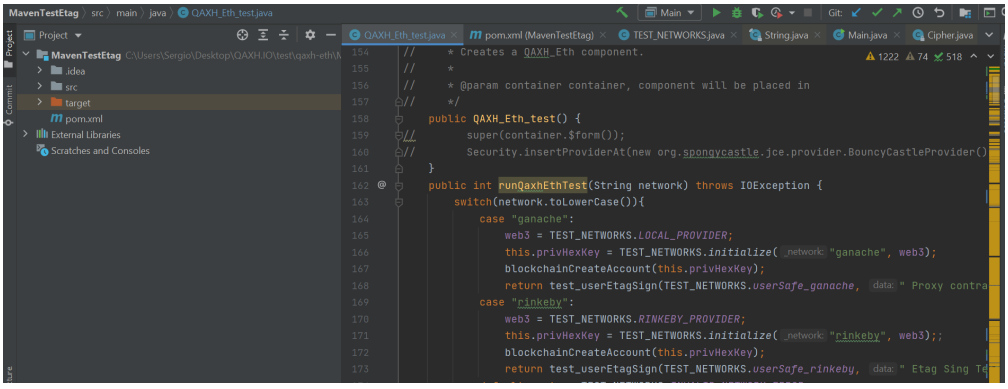

Figura 4.2: Intellij IDEA IDE

4.2.3 API Test

Al fine di testare le Application programming interface del nostro sistema si è preferito, per semplicità, non utilizzare una vera libreria di test, ma il tool linux , *cURL* che permette di chiamare dei *URL* ed interpretarne le risposte. Quest'ultimo

è un progetto software che fornisce una libreria (libcurl) e uno strumento da riga di comando (curl) per il trasferimento di dati utilizzando vari protocolli di rete. Il tool in questione ha permesso in primo luogo di verificare l'effettiva operatività delle varie API ed in seguito di verificarne la correttezza delle risposte e dei vari parametri inseriti.

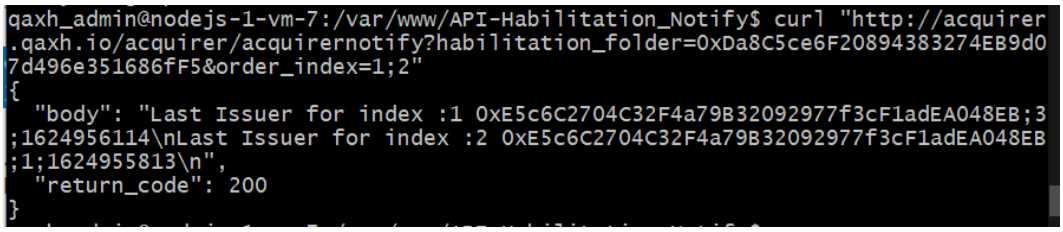

Figura 4.3: Utilizzo di *Curl*

4.3 Verifica della qualità: *HabilitationFolder*

Per verifica della qualità o validazione del software si intende quel processo di controllo di una determinata applicazione, al fine di valutare se essa risulta conforme agli usi previsti ed in particolare alle esigenze dell'utente, agli aspetti di sicurezza e ai requisiti d'uso richiesti ad inizio progetto.

In pratica tale processo serve a verificare che il software *faccia ciò che deve fare* e impedisca di fare *cose che non dovrebbe essere in grado di fare.*

Le sotto sezioni che seguiranno illustrano il funzionamento generale del sistema realizzato da i vari punti di vista dell'utilizzatore.

4.3.1 Identità e Attori

Oltre all'identità registrata nella userSafe, l'encryptID permette di aggiungere un particolare indirizzo al QRcode dell'identità iniziale. Questo indirizzo punta a una chiave pubblica registrata nei dati dello UserSafe. La chiave pubblica, quindi, non è nel QRcode evitando il rischio di modifica illecita di quest'ultimo. Il destinatario dell'encryptID leggerà la chiave pubblica in userSafe e potrà verificare la corrispondenza tra lo UserSafe e questa chiave pubblica. Questo nuovo strumento consente di fornire all'utente dati crittografati, garantendone quindi la confidenzialità.

E' possibile delineare 4 figure con 4 rispettivi ruoli all'interno del processo bancario riguardante la gestione dei permessi:

- **CEO - Issuer**: L'amministratore che deciderà di utilizzare il servizio fornito dal *habilitationFolder* e avrà i pieni poteri all'interno dello stesso. (Figure 4.6, 4.8, 4.15)

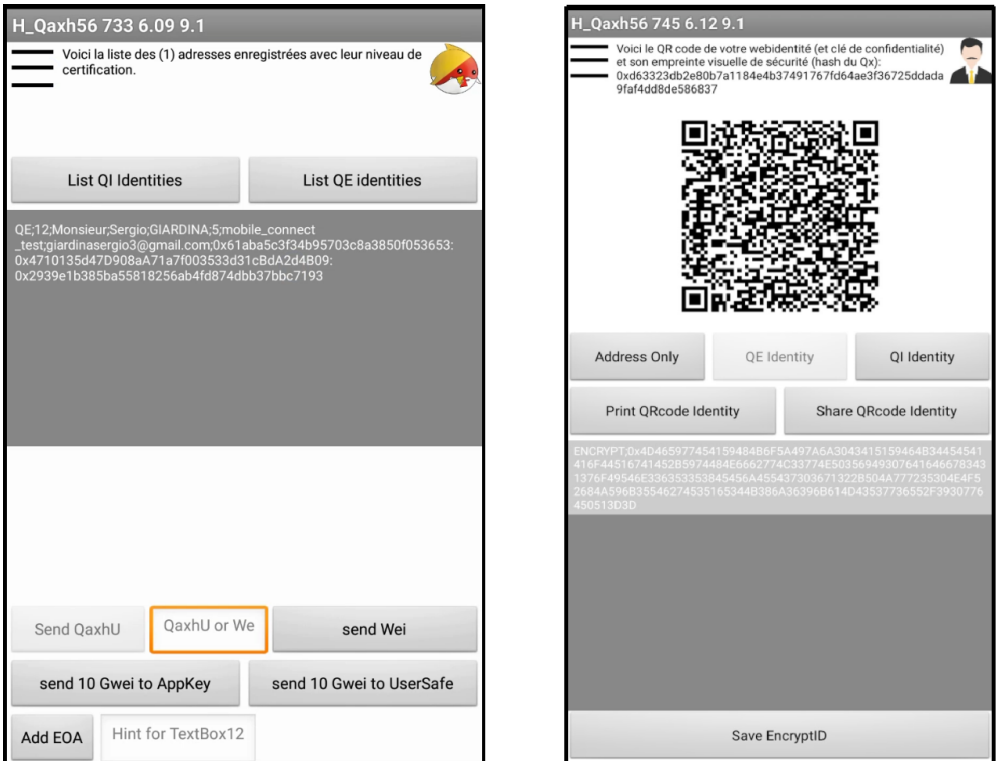

Figura 4.4: Sinistra : Identità fornita da Mobile Connect; Destra: Identità in forma di QRcode

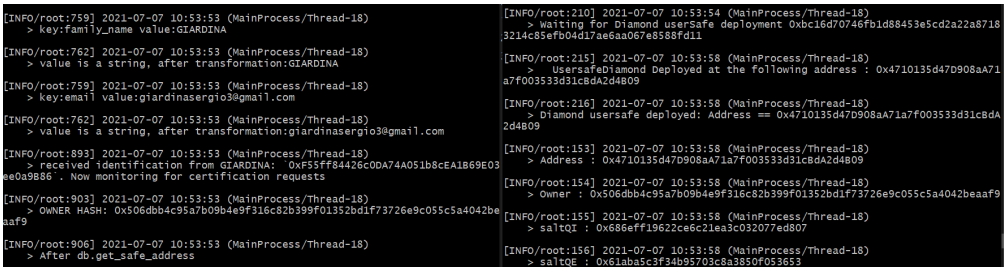

Figura 4.5: Log Server 16 - Creazione di un'identità

- **Contabile**: Il rappresentante d'azienda che potrà "creare" gli ordini nel sistema. (Figure 4.15, 4.13, 4.18, 4.19)
- **CFO**: Il compontente d'azienda che potrà "firmare" gli ordini all'interno del sistema.(Figure 4.17, 4.19)
- **Acquirer**: Il manager di banca che gestirà e offrirà il servizio. (Figure 4.9, 4.11, 4.21)

E' importante elencare i precedenti ruoli in quanto le varie sezioni dell'app ogni volta saranno dedicate ad uno degli attori (l'attore pensato per quella pagina è rappresentanto in alto a destra). Inoltre considerando che ogni attore rappresenta un indirizzo presente o meno sul contratto, in alcune funzioni specifiche per attore, il contratto eseguirà le giuste verifiche di autorizzazioni. Nella 4.6 è presente anche un comando di *Read HabilitationFolder* che permette di leggere un indirizzo di una *Folder* già creata, garantendo le stesse funzionalità su dispositivi diversi.

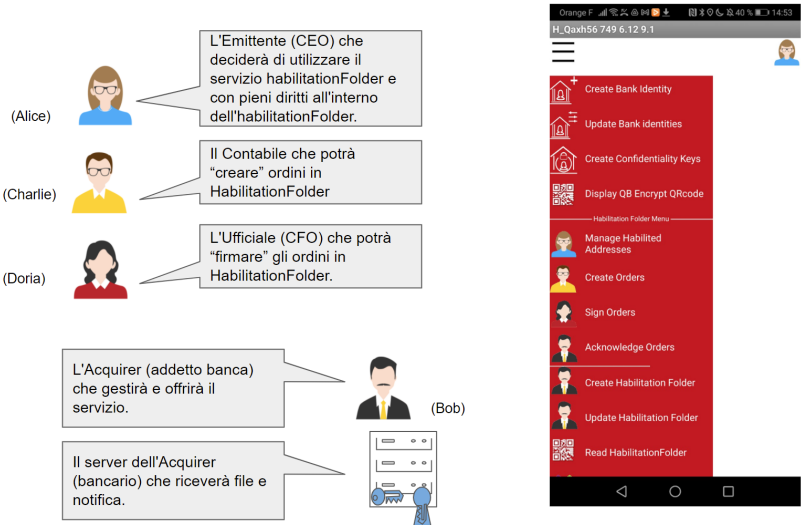

Figura 4.6: Attori in *HabilitationFolder*

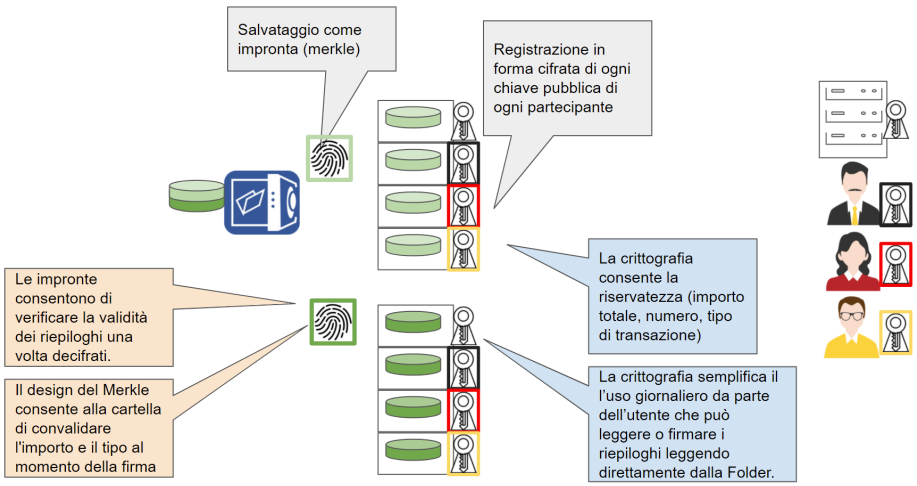

Figura 4.7: Chiavi per identità

4.3.2 Creazione e Registrazione

La figura 4.8 mostra alcuni step che precedono la vera attivazione del contratto di *HabilitationFolder*. Quest'ultimi rappresentano l'invio di identità da parte dell'*Issuer* che riceve in cambio l'indirizzo della *HabilitationFolder* distribuita sulla Blockchain. Da qui l'attivazione lato *Issuer*, figura 4.15, e l'attivazione lato *Acquirer*, figura 4.11.

Date le figure in 4.11 nelle prossime pagine, è possibile analizzare lo schema presente in 4.9 che mostra i vari passaggi chiave della fase di *creazione* del contratto e della *registrazione* nel *DirectoryScheme*. Seguono le varie funzionalità e di conseguenza le tecnologie coinvolte:

- Dal lato *Issuer* i comandi in rosso non sono che funzioni Java esterne disponibili per uno UserSafe grazie all'estensione *Qaxh_eth*.
- Nelle frecce piu lunghe che vanno dall'*Acquirer* all'*HatchServer* sono rappresentate le API in *Python* coinvolte.
- All'interno dell'implementazione di queste due funzionalità sono chiamate le varie funzioni disponibili del contratto distribuito sulla Blockchain.

Per quanto riguarda i dettagli delle due schermate (4.11) è utile chiarificare che dati sul centro-destra della seconda schermata a destra, sotto il titolo *FolderVariables*, non sono altro che i parametri della *Folder* passati nell'API di creazione.

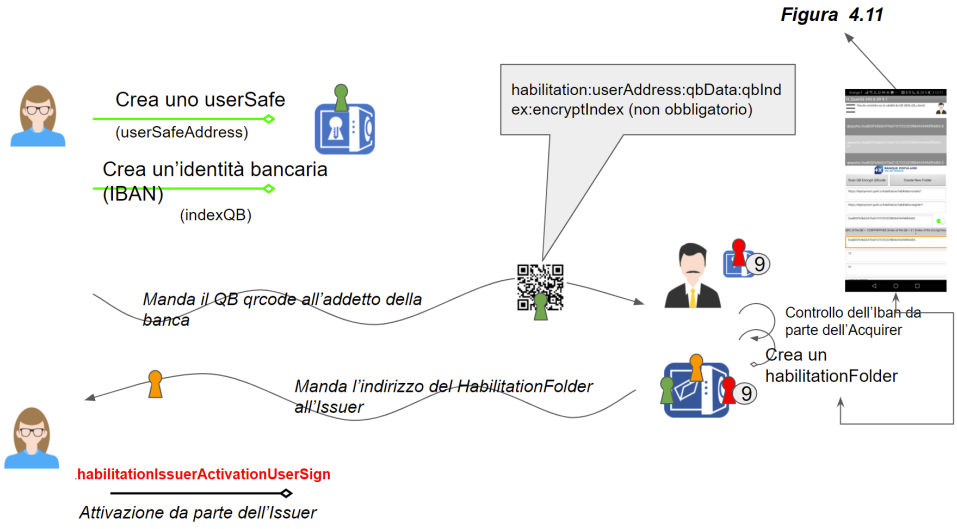

Figura 4.8: Pre-*HabilitationFolder*

Nella figura 4.10 è possibile notare 3 stringhe esadecimanli, dall'alto verso il basso:

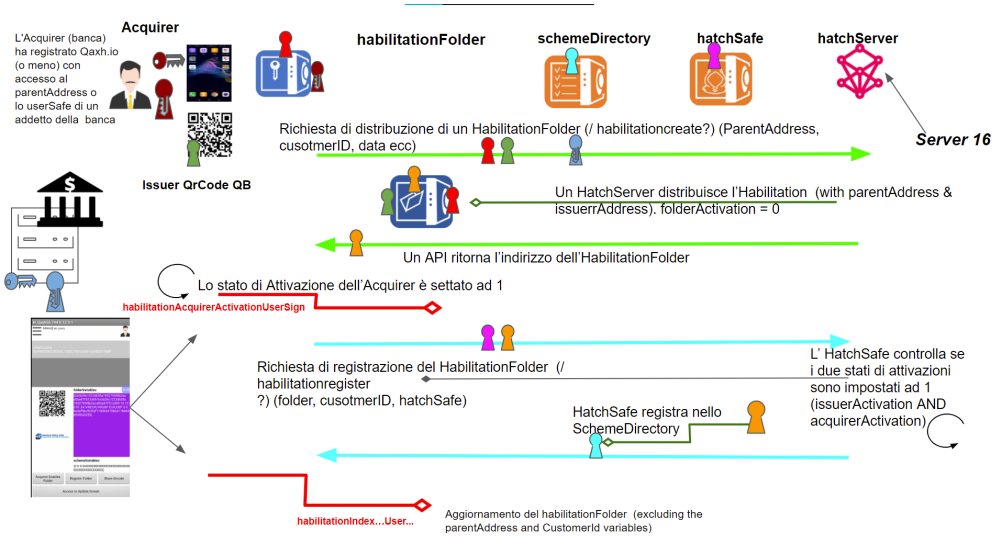

Figura 4.9: Scenario per creazione e registrazione del *HabilitationFolder*

```
[INFO/root:53] 2021-07-08 14:27:18 (MainProcess/Thread-2)
    > Start deploying contract HabilitationDirectory from : 0x0639c1223d05Fa1952
795f8B2ECd06e87f512D59

[INFO/root:37] 2021-07-08 14:27:20 (MainProcess/Thread-2)
    > Start deploying contract : HabilitationDirectory (tx_hash: 0x0e4a2b1e82534
a6aaf25c7f25d47d5dd902081bd2f540d0199beece34c2a6317)

[INFO/root:69] 2021-07-08 14:27:35 (MainProcess/Thread-2)
    > Deployed contract `HabilitationDirectory` at address 0x99D020EE2B3DdC1383C
1581bC897a3fdE30198BF

[INFO/root:74] 2021-07-08 14:27:35 (MainProcess/Thread-2)
    > Deployed all contracts in 0:00:16.649075

[INFO/werkzeug:225] 2021-07-08 14:27:35 (MainProcess/Thread-2)
    > 127.0.0.1 - - [08/Jul/2021 14:27:35] "POST /habilitation/habilitationcreat
e?parentaddress=0x0639c1223d05Fa1952795f8B2ECd06e87f512D59&CID=12&issueraddress=
0x0639c1223d05Fa1952795f8B2ECd06e87f512D59&acquireraddress=0xDaf5BcFB2FaF1100924
7582E176B64D590fa5259&issuerqb=0&categories=VIREUR;VIRGBP&currencies=EUR;GBP&ind
ex1=https%3A%2F%2Fdocs.google.com%2Fdocument%2Fd%2F1kS8tkiovdjpJISgF_tVtyGN1OJdF
XALObK5iZjHyFRo%2Fedit%3Fusp%3Dsharing&index2=Banque+Populaire+Aquitaine+Centre-
Atlantique&index3=0&index4=logo-10907.png&index5=Pouvoirs+Bancaires&index6=http%
3A%2F%2Facquirer.qaxh.io%2Facquirer%2Facquirerfile%3F&index7=http%3A%2F%2Facquir
er.qaxh.io%2Facquirer%2Facquirernotify%3F&index8=https%3A%2F%2Fdocs.google.com%2
Fdocument%2Fd%2F1kS8tkiovdjpJISgF_tVtyGN1OJdFXALObK5iZjHyFRo%2Fedit%3Fusp%3Dshar
ing&index9=1.7&index10=1%2F4%2F2021&index11=0x12345&index12=0x HTTP/1.0" 200 -
```

Figura 4.10: Log Server 16 sulla creazione

1. Indirizzo dello UserSafe che chiama l'API.
2. Hash della transazione che crea il contratto.
3. Indirizzo del contratto appena creato.

La figura 4.12 mostra la corretta esecuzione della transazione su *Rinkeby*.

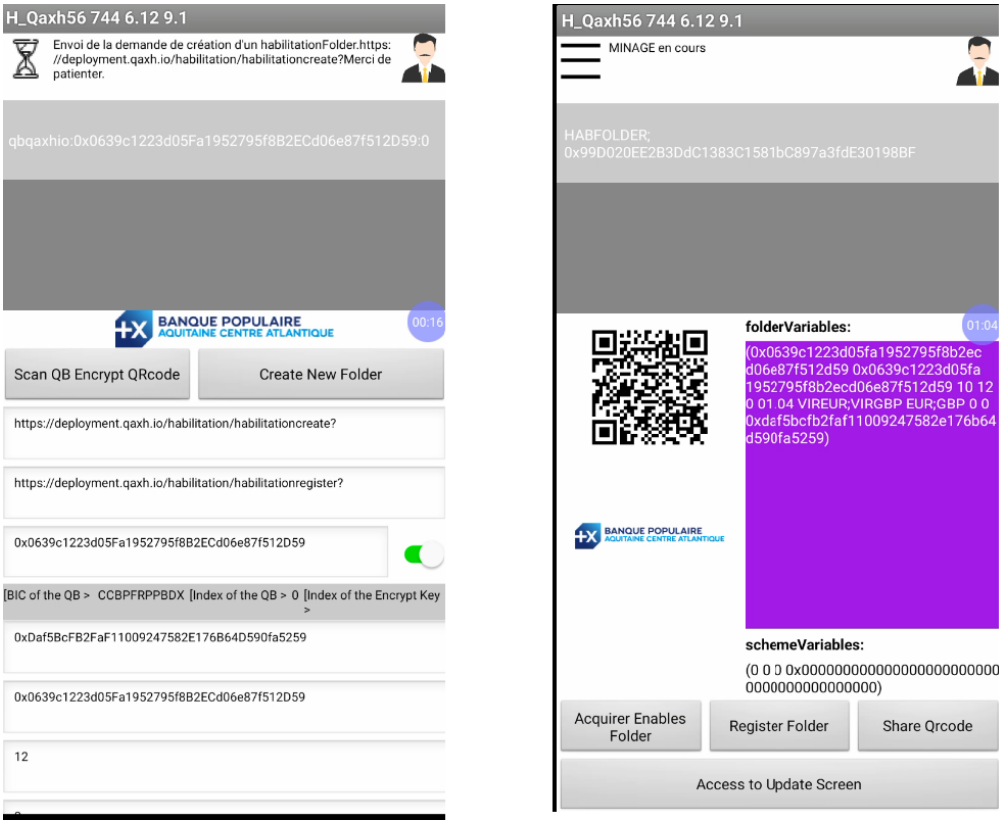

Figura 4.11: Screen di creazione e registrazione per il Manager di banca (Acquirer)

Figura 4.12: Etherscan: Stato della transazione

4.3.3 Produttore di ordini (Issuer)

La seguente sotto-sezione approfondirà le opportunità dell'*Issuer*, un classico rappresentante di azienda nel nostro sistema. Nei due schemi seguenti verrà mostrato:

1. **4.13**: La struttura ideale (Indirizzo,Autorizzazioni) che viene costruita dall'Issuer, *Alice* in questo caso, sullo smart contract .

2. **4.14**: I passaggi principali di una fase di creazione di un ordine da parte di un contabile, *Charlie* in questo caso.

Nella seguente 4.15 vengono mostrate due schermate d'uso, rispettivamente per *Alice* e *Charlie*.

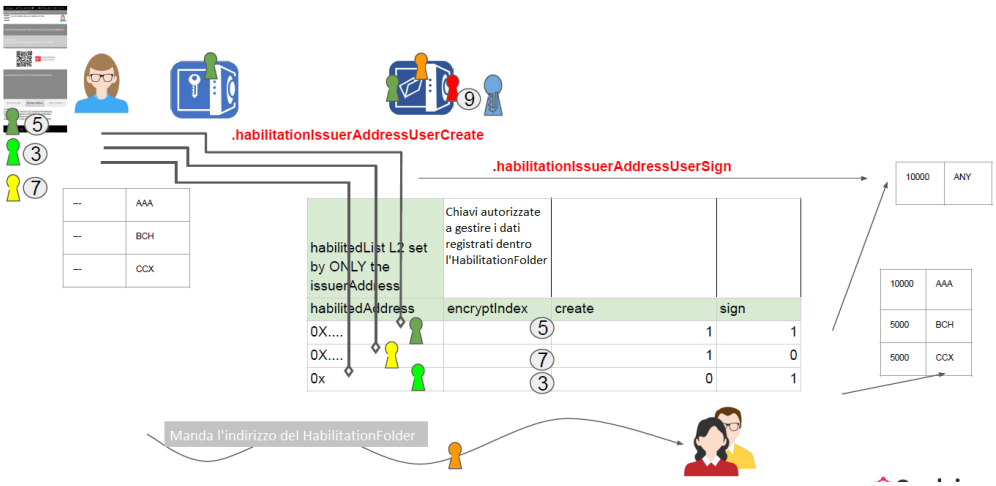

Figura 4.13: Scenario per Alice (CEO)

Officer CFO

I due screen nella figura 4.17 mostrano la firma di un ordine da parte di un *CFO*. In dettaglio:

- In alto abbiamo i dati principali dell'ordine: numero d'ordine e *MerkleHeader* dei dati della 4.16.

- In basso cerchiati abbiamo in ordine da sinistra verso destra: uno stato per l'azienda (1 - created); il timestamp dell'operazione; l'indirizzo-utente che ha inserito quello stato. Nello screen a destra, dopo la firma, è mostrato lo stato 3 (3 - signed).

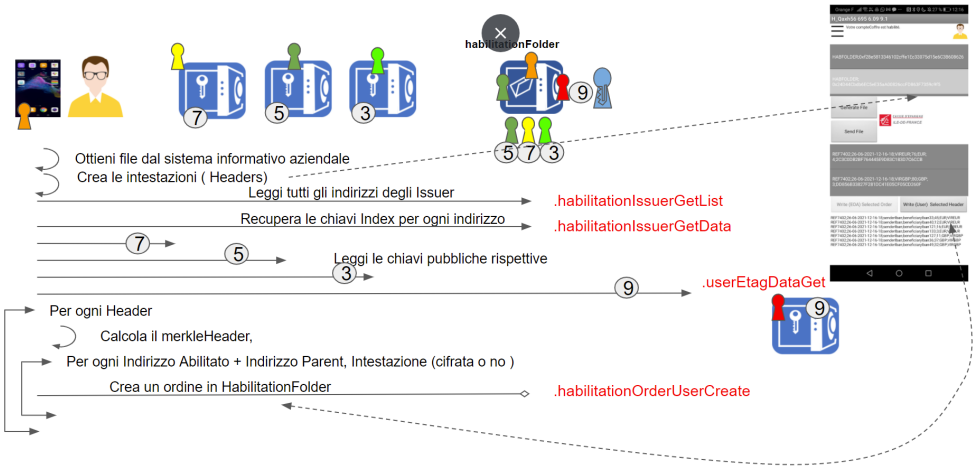

Figura 4.14: scenario per Charlie (CFO)

In questa operazione è fondamentale la presenza di uno smart contract che esonera il CFO da molte verifiche, prendendosi carico dei compiti piu onerosi:

1. Verifica se lo stato esistente dell'ordine sia "1" (1-created).
2. Calcola il merkleheader con M (hash intermedio del merkleheader), importo e categoria.
3. Confronta il merkleHeader calcolato con quello indicato nell'ordine (all'interno della *HabilitationFolder*).
4. Verifica che quel UserSafe sia abilitato a firmare.
5. Verifica che il limite minimo di quel UserSafe per quella categoria non sia superiore all'importo indicato nei parametri dallo *Issuer*.
6. Controlla che non esista già un ordine con quell'indice.

Se tutti i controlli hanno successo, l'*HabilitationFolder* cambia lo stato dell'Issuer in "3" (3-signed): Al fine di comprendere meglio il punto 2 e il punto 3 è riportata la figura 4.16 che mostra la struttura di un Ordine e del suo *Header* in dettaglio.

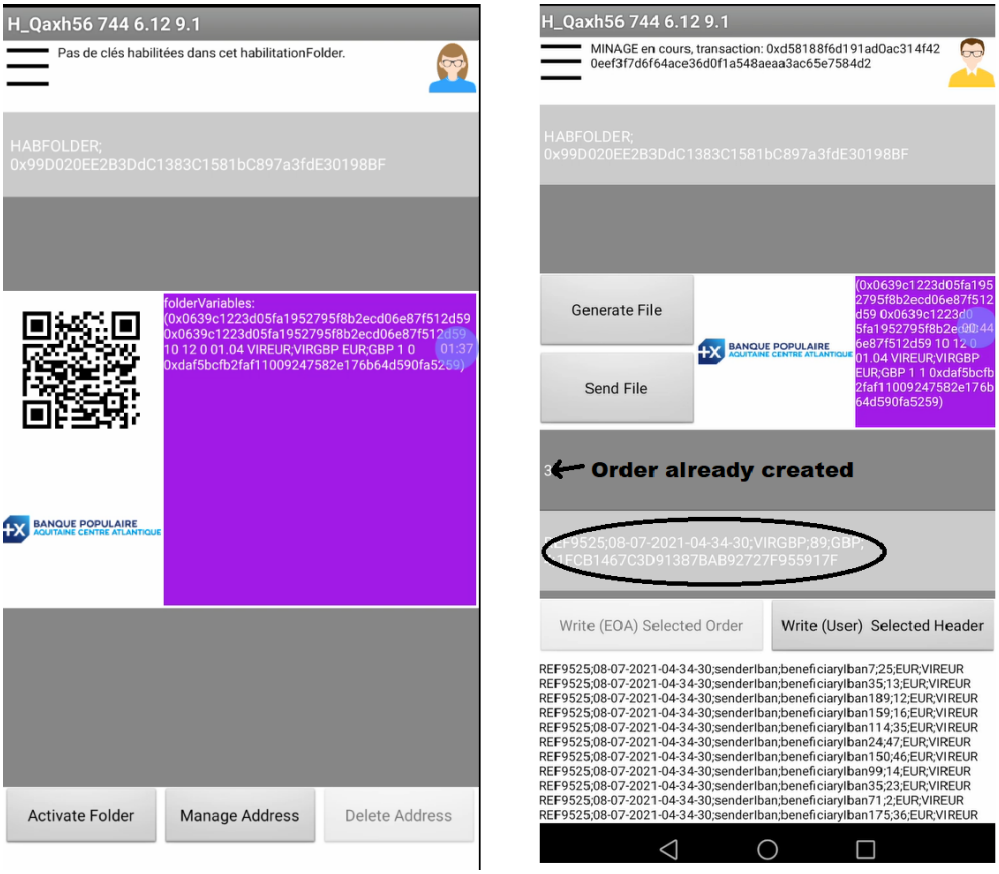

Figura 4.15: Pagina del CEO - Pagina del contabile

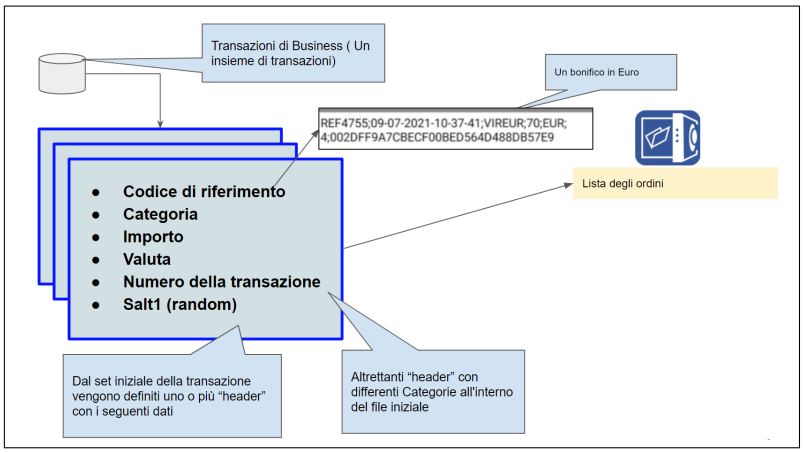

Figura 4.16: Ordini in dettaglio

Figura 4.17: Pagina del CFO

4.3.4 Consumatore di ordini (Acquirer)

Tra le opportunità pensate per la struttura Acquirer (banca,istituzione ecc), oltre alla creazione e registrazione della Folder, figura 4.11, sono implementate delle API interne al server d'acquirer che permettono la gestione di un proprio ruolo all'interno di una *HabilitationFolder*. Queste 2 API, implementate sul server 7 (Server test d'azienda) sono riportate negli schemi 4.18 e 4.19:

- *Acquirerfile?* L'istituzione-Acquirer fonirà la possibilita di utilizzo di tale API al contabile d'azienda, al fine di sottoppore, alla stessa istituzione, un documento, ben dettagliato, sugli ordini già definiti, e possibilmente firmati, che corrispondono ad alcuni Header inseriti nella *HabilitationFolder* (4.16 , 4.14).

- *Acquirernotify?* L'istituzione-Acquirer fonirà la possibilità di utilizzo di tale API al contabile d'azienda o al CFO, al fine di ricevere una notifica sui possibili

cambiamenti di stato relativi ad un ordine.

L'*Acquirerfile* sarà possibile grazie al tasto disponibile mostrato nello screen a destra della 4.15. Come è stato anticipato, tale funzionalità sarà una prerogativa del contabile. In maniera simile l'API di notifica sarà avviata nel momento in cui si toccherà uno degli elementi (ordini) della lista nella figura 4.21, di conseguenza sarà possibile notare un automatico aggiornamento del *Bank Status* sulla destra.

API: Invio del File

Come già descrive la figura 4.18, l'API bancaria si occuperà di verificare la corrispondenza tra ciò che viene emesso nel contratto e i dettagli dell'ordine riportati sul file ricevuto dal contabile. In dettaglio si occuperà dei seguenti controlli, ritornando un codice di stato corrispondente in caso di fallimento:

- **470**: Verifica del tipo di richiesta HTTP.
- **471**: Verifica dell'integrità del file: Una categoria non appartenente alla lista delle categorie della Folder.
- **472**: Verifica dell'integrità del file: Una valuta non appartenente alla lista delle valute della Folder.
- **473**: Verifica dell'integrità del file: gli importi degli ordini non sono in cifre.
- **474**: Verifica dell'esistenza e della validità degli Header del contratto.
- **475**: Verifica dello stato di attivazione dell'*HabilitationFolder* (Attivata sia per Issuer che per Acquirer).
- **476**: Verifica e ricalcolo del *Merkleheader*.
- **477**: Verifica la connessione alla blockchain e la validità del contratto passato come argomento.
- **478**: Verifica la validità dell'indice della lista degli ordini.
- **479**: Verifica dell'esistenza del file degli ordini.

Risponderà con il codice **200** se tutti i controlli precedenti avranno esito positivo.

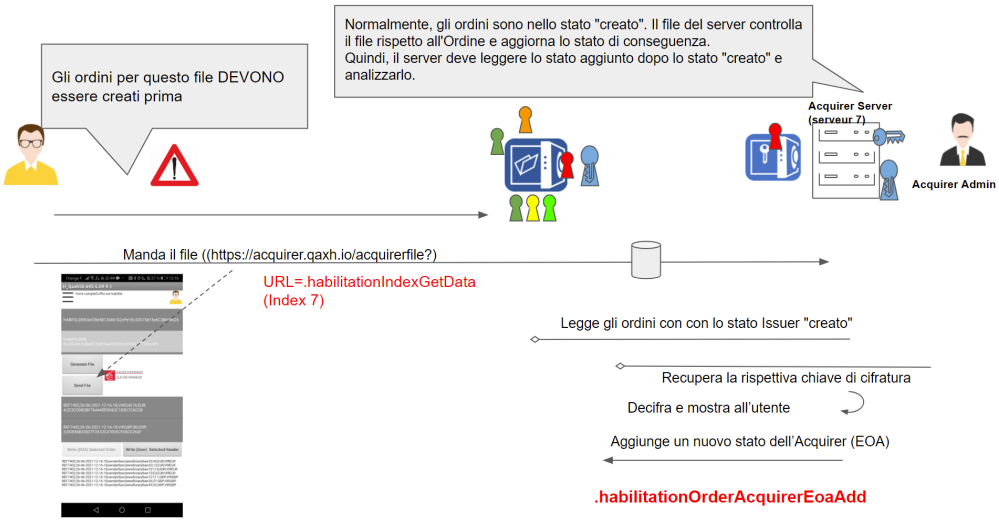

Figura 4.18: Scenario per addetto bancario: *Gestione file*

API: Invio della notifica

Come già riportato nella figura 4.19, l'API di notifica si occuperà di verificare la corrispondenza temporale tra gli stati dell'Issuer e i propri stati Acquirer, aggiornando quest'ultimi in caso di necessita. Ritonerà i seguenti codici di stato:

- **470**: Uno o piu problemi della seguente lista.
- **475**: Indirizzo non corrispondente ad una *HabilitationFolder*
- **477**: Impossibile collegarsi alla blockchain.
- **478**: Indice non valido della lista degli ordini.
- **200**: I controlli non hanno portato ad alcun fallimento, di conseguenza lo stato bancario è aggiornato. (*habilitationOrderAcquirerUserAdd*)

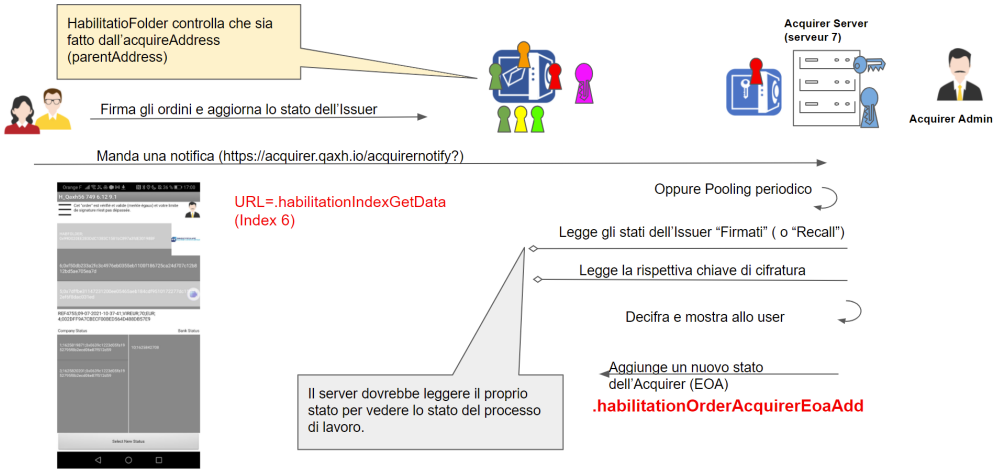

Figura 4.19: Scenario per addetto bancario: *Gestione notifiche*

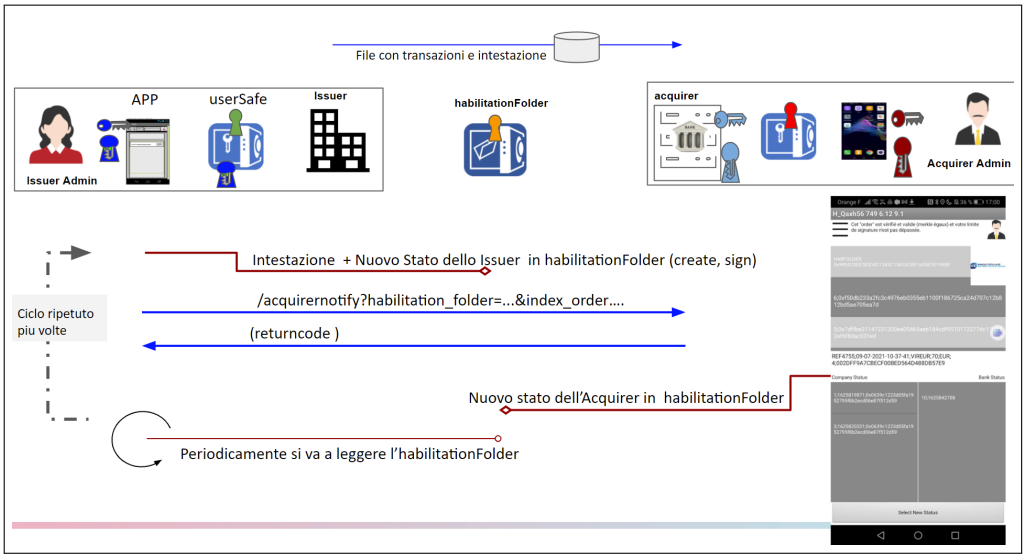

Figura 4.20: API di notifica

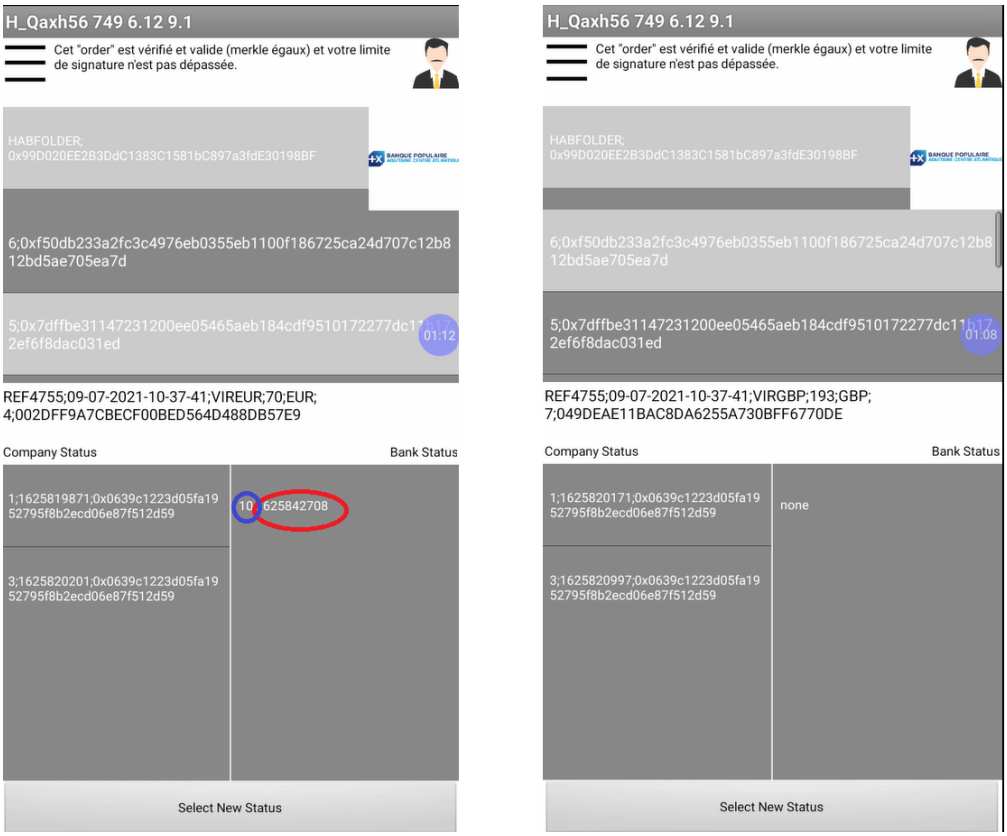

Figura 4.21: Applicazione mobile per l'addetto bancario

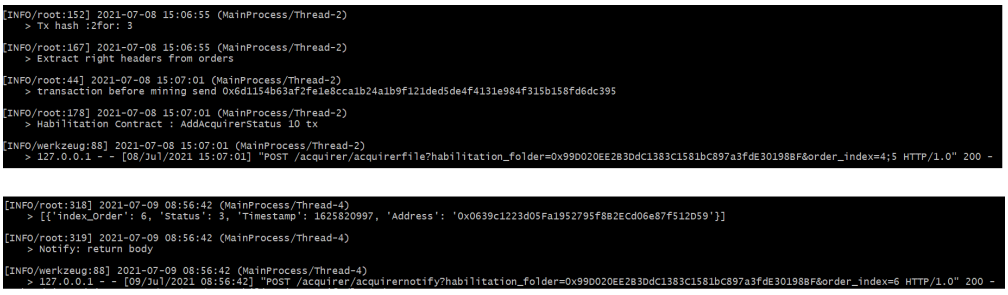

Figura 4.22: Server log per indici 4;5 , indice 6

Capitolo 5

Conclusioni e sviluppi futuri

Questo lavoro di tesi nasce, come già ampiamente descritto precedentemente, dalla necessità di dare forma e implementazione ad un sistema di permessi bancari, basato sulle opportunità d'utilizzo offerte dai sistemi blockchain. In tal senso, il caso d'uso descritto nel capitolo precedente, l'emissione e la verifica degli ordini in un rapporto *Issuer-Acquirer*, ha permesso di mettere in luce la realizzabilità implementativa di una soluzione efficiente al classico ostacolo della securizzazione dei rapporti tra due enti diversi, che *non si conoscono* e quindi *non si fidano*. La soluzione ampiamente descritta per il particolare caso d'uso analizzato è risultata essere innovativa, adattabile e facilmente integrabile nell'obiettivo di realizzare un framework bancario completo. L'architettura mostrata e pensata, vuole sottolineare una netta separazione tra le varie componenti tecnologiche, mostrando come l'utilizzo di un basilare ambiente di sviluppo sia facilmente adattabile ad un complesso sistema blockchain, constantemente in evoluzione. Tale separazione, in linea con il principio della *separation of concerns*, rende totalmente scalabile la soluzione a tutto il flusso dei dati , e di transazioni, di un caso d'uso reale.

Il framework QAXH.IO, nella peculiarità del lavoro di tesi, gestisce quindi le molteplici relazioni tra i protagonisti dell'*Issuer*, contabile, admin ecc, e quelli dell'*Acquirer*, manager bancario. L'intero processo è possibile grazie al totale e preciso controllo di validità realizzato su ogni punto di relazione (ogni riga di codice del contratto) tra i vari punti focali del caso d'uso: transazione, utente, azienda, permessi e identità.

Nel lavoro descritto, è possibile delineare come punto cardine dell'intero progetto la realizzazione di uno o più smart contract che fornissero garanzie dal punto di vista dell'automazione, della sicurezza e della tracciabilità dei processi bancari. Gli smart contract sono stati realizzati alla base di un'attenta riflessione sui consumi di gas generati con la loro creazione ed esecuzione. Al contempo i servizi server e utente che svolgono il ruolo di ente centrale del framework, mantengono una struttura del tutto in linea con quelle che sono le caratteristiche intrinseche della

tecnologia blockchain adottata, sfruttando le librerie di *Ethereum*. Presupposto ciò, l'applicazione nel suo complesso sfrutta la libreria *Web3*, facilitando l'interazione con gli smart contract, semplificando e riducendo il tutto, da un lato, ad una verifica di informazione comune, dall'altro, ad una semplice esecuzione di codice *Solidity* per tutti i nodi. Infine, è fondamentale rammentare come la soluzione proposta, per trasparenza, sicurezza, e automazione di operazioni di natura bancaria, sia del tutto innovativa nel settore, in quanto una verifica e una tracciabilità efficiente, intriseche ad un sistema Blockchain, sono l'unica risposta ad un aumento ed un'evoluzione della complessità del settore FinTech che la società moderna richiede.

Gli sviluppi futuri relativi alla tesi si presentano molteplici e variegati. Allo stato attuale dell'applicazione innanzitutto bisogna delineare con accuratezza le interazioni con i diversi dati , stabilirne i compiti e i meccanismi di funzionamento, per poi avanzare verso uno sviluppo di più ampie vedute e prospettive per l'intero progetto. In quest'ottica, il principio fondamentale è quello di adattatare il sistema progettato lungo tutto il flusso di utenti del caso di riferimento, definendo in maniera ancor più specifica i vari ruoli-utente nel rapporto emittente-consumatore , per poi avere una soluzione che può integrarsi a tutti gli effetti con il framework nella sua totalità, dove l'automazione sicura di un prodotto-utente è la chiave di volta. Un ulteriore sviluppo futuro, strettamente legato alle fasi di sviluppo del framework sarà l'implementazione di un'unica soluzione per l'intero progetto QAXH.IO, andando ad unire i differenti casi d'uso dimostrativi, sperimentati singolarmente, attraverso un'unica user-application. Molto interessante può essere anche l'ampliamento del peculiare caso d'uso dei poteri bancari con la tokenizzazione di alcuni servizi e l'introduzione della muti-firme. Per primo, l'inserimento nel sistema dei Token può risultare molto intrigante in quanto i token non-fungibili dimostrano degli asset che permettono sistemi di valore e di precedenza in un classico sistema *Issuer-Acquirer*. La possibile erogazione di alcuni servizi potrebbe avvenire in base ad un eventuale graduatoria stipulata proprio in base al possesso dei token ricevuti. In secondo luogo è possibile ampliare l'attuale offerta di gestione dei poteri bancari permettendo l'introduzione di piu firme e piu vincoli per i vari odini, aumentando di molto le possibilità d'uso per le aziende di alto livello. Lo studio della Blockchain, e del suo apparato distribuito, è in continua evoluzione, il che implica la necessità di analizzare con estrema attenzione le nuove tecnologie al fine di adattarle in merito a tracciabilità, trasparenza e sicurezza lungo tutto il processo di interazioni tra istituzioni e utenti fisici. [42] [24] [25]

Bibliografia

[1] *Fintech Blockchain*. URL: https://www.investopedia.com/terms/f/fintech.asp (cit. a p. iii).

[2] BPCE SA. *Data BPCE*. URL: https://bpce.opendatasoft.com/pages/home/ (cit. a p. 1).

[3] Naxitis. *Naxitis Data*. URL: https://www.im.natixis.com/it/home (cit. a p. 1).

[4] *La blockchain dans un groupe bancaire*. URL: https://www.89c3.com/news/la-blockchain-dans-un-groupe-bancaire-interview-de-cyril-vignet/ (cit. alle pp. 3, 62).

[5] *Satoshi Nakatomo : Bitcoin: A Peer-to-Peer Electronic Cash System*. URL: https://bitcoin.org/bitcoin.pdf (cit. a p. 8).

[6] Wikipedia Foundation. *Blockchain System*. URL: https://en.wikipedia.org/wiki/Blockchain (cit. a p. 8).

[7] Filippo Angeloni. *Consensus blockchain*. URL: https://filippoangeloni.com/diversi-tipi-di-consenso-nella-blockchain/#:~:text=La%20Blockchain%20e%20l'algoritmo,alcuni%20possibili%20fallimenti%20del%20network. (cit. a p. 10).

[8] *Byzantine fault tolerance*. URL: https://www.tutorialspoint.com/%20what-is-byzantine-fault-tolerance#:~:text=The%20concept%20of%%2020Byzantine%20Fault,values%20to%20misguide%20the%20network. (cit. a p. 10).

[9] *Proof of work*. URL: https://en.bitcoin.it/wiki/Proof_of_work (cit. a p. 10).

[10] academy.binance. *Proof of stake*. URL: https://academy.binance.com/blockchain/proof-%20of-stake-explained (cit. a p. 10).

[11] *Permissionless and permissioned blockchain diffusion*. URL: https://www.sciencedirect.com/science/article/abs/pii/S0268401219314586 (cit. a p. 14).

[12] *blockchain4innovation*. URL: https://www.blockchain4innovation.it/esperti/blockchain-perche-e-cosi-importante/ (cit. a p. 16).

[13] *Ethereum Virtual Machine*. URL: https://cryptoast.fr/quest-ce-que-la-machine-virtuelle-ethereum (cit. a p. 17).

[14] Wikipedia Foundation. *Smart Contract*. URL: https://it.wikipedia.org/wiki/Smart_contract (cit. a p. 19).

[15] Lessig L. «Code is Law: On Liberty in Cyberspace». In: (2020). URL: https://www.harvardmagazine.com/2000/01/code-is-law-html (cit. a p. 19).

[16] bit2me.com. *Cosa sono le dapps*. URL: https://academy.bit2me.com/it/che-cosa-sono-dapps/ (cit. a p. 20).

[17] Altcoinbuzz. *Crypto Mainnet vs Testnet*. URL: https://www.altcoinbuzz.io/bitcoin-and-crypto-guide/crypto-mainnet-vs-testnet/ (cit. a p. 22).

[18] Ethereum Foundation. *Different testnets*. URL: https://ethereum.stackexchange.com/questions/27048/comparison-of-the-different-testnets (cit. a p. 22).

[19] Ledgerinsights. *Ihs market Blockchain*. URL: https://www.ledgerinsights.com/ihs-market-blockchain-forecast-2-trillion/ (cit. alle pp. 24, 26).

[20] Modis Foundation. *Blockchain finance*. URL: https://www.modis.com/it-it/insights/articoli/blockchain-finance/ (cit. a p. 24).

[21] Gatteschi V. Lamberti F. Demartini C. Pranteda C. Santamaría V. «Blockchain and Smart Contracts for Insurance: Is the Technology Mature Enough?» In: Future Internet 10(2) (). URL: https://www.doi.org/10.3390/fi10020020 (cit. a p. 24).

[22] Deloitte. *Use cases in financial services*. URL: https://www2.deloitte.com/it/it/pages/financial-services/articles/5-blockchain-technology-use-cases-in-financial-services---deloit.html (cit. a p. 26).

[23] Cryptonomist. *banca-mediolanum-ethereum-blockchain*. URL: https://en.cryptonomist.ch/2019/04/04/banca-mediolanum-ethereum-blockchain/ (cit. a p. 26).

[24] businessbecause. *4 Ways Blockchain Is Revolutionizing FinTech*. URL: https://www.businessbecause.com/news/insights/7534/blockchain-fintech?sponsored (cit. alle pp. 28, 108).

[25] *How Fintech and Blockchain Can Shake the Foundations of the Financial World*. URL: https://theblockbox.io/blog/how-fintech-and-blockchain-can-shake-the-foundations-of-the-financial-world/ (cit. alle pp. 28, 108).

[26] *Ethereum*. URL: https://it.wikipedia.org/wiki/Ethereum (cit. a p. 31).

[27] *Metamask*. URL: https://metamask.io/ (cit. a p. 31).

[28] Web3. *Web3j documentation*. URL: http://docs.web3j.io/latest/ (cit. a p. 31).

[29] Web3. *Web3py documentation*. URL: https://web3py.readthedocs.io/en/stable/ (cit. alle pp. 31, 62).

[30] *Truffle*. URL: https://www.trufflesuite.com/ (cit. a p. 31).

[31] *Remix*. URL: http://remix.ethereum.org/ (cit. a p. 32).

[32] Infura F. *Infura system*. URL: https://infura.io/ (cit. alle pp. 32, 64).

[33] Ethereum Foundation. *Etherscan*. URL: https://etherscan.io/ (cit. a p. 32).

[34] Ethereum Foundation. *Solidity Documentation*. URL: https://docs.soliditylang.org/en/v0.8.3/ (cit. a p. 42).

[35] Ethereum Foundation. *Geth*. URL: https://geth.ethereum.org/docs/install-and-build/installing-geth/ (cit. a p. 65).

[36] MIT. *AI2 documentation*. URL: https://appinventor.mit.edu/ (cit. a p. 75).

[37] MIT. *AppInventor extensions*. URL: https://mit-cml.github.io/extensions/ (cit. a p. 77).

[38] *Etherscan Gas Tracker*. URL: https://etherscan.io/gastracker (cit. a p. 87).

[39] *EthGasStation*. URL: https://ethgasstation.%20info/.er (cit. a p. 87).

[40] *Platform for decentralization of healthcare and benefits administration*. URL: https://www.solve.care (cit. a p. 88).

[41] *Business in Blockchain*. URL: https://www.zerounoweb.it/cio-innovation/blockchain-cose-come-utilizzarla-e-come-cambiera-il-business/ (cit. a p. 89).

[42] R. Bianchi G. Chiap J. Ranalli. «Blockchain: Tecnologia e applicazioni per il business: tutto ciò che serve per entrare nella nuova rivoluzione digitale,Hoepli». In: ISBN 978-8820390075 (2019) (cit. a p. 108).