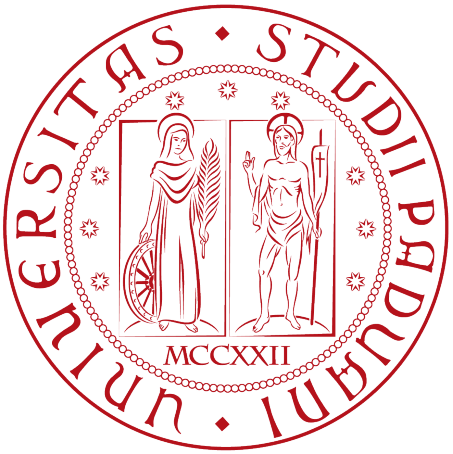
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Progettazione e sviluppo di un servizio di Identity Access Managment basato su blockchain

Tesi di laurea triennale

Relatore
Prof.Gilberto Filè

Laureando
Simone Ballarin

Anno Accademico 2017-2018

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

— Oscar Wilde

Dedicato a ...

Sommario

Il presente documento riassume il lavoro svolto durante il periodo di stage, della durata di 320 ore, presso l'azienda iVoxIT S.r.l. di Padova.
Lo scopo principale del prodotto sviluppato è quello di integrare all'interno dell'applicativo Monokee, un sistema di creazione e verificà dell'identità basato su tecnologia blockchain compatibile con lo standard SAML. Nel corso del documento verranno anche esposte le basi teoriche del prodotto, come la gestione delle identità e il Single Sign-On (SSO).
In una prima fase mi sono concentrato sullo studio di vari documenti forniti dall'azienda inerenti al progetto e a come questo si doveva integrate con l'attuale sistema, una volta capiti gli aspetti fondamentali mi sono dedicato alla ricerca e all'apprendimento di vari strumenti tecnologici confacenti ad un corretto sviluppo. Dopo aver identificato le principali funzionalità richieste e compreso il funzionamento di come erano definiti i flussi di lavoro ho iniziato la progettazione del servizio e quindi alla definizione di un'architettura che fosse estendibile, manutenibile e integrabile con quella esistente. Questo lavoro ha richiesto molti sforzi, ma il risultato, seppure all'altezza della aspettative dell'azienda, ha fatto emergere come un'approccio basato su blockchain risulti essere non adatto nella maggior parte dei contesti.

"Life is really simple, but we insist on making it complicated"

— Confucius

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. NomeDelProfessore, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero ringraziare con affetto i miei genitori per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme e le mille avventure vissute.

Padova, Agosto 2018 Simone Ballarin

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Introduzione al contesto applicativo.

Esempio di utilizzo di un termine nel glossario
Application Program Interface (API).

Esempio di citazione in linea
site:agile-manifesto

Esempio di citazione nel pie' di pagina
citazione[1]

1.1 L'azienda

L'attività di stage è stata svolto presso l'azienda iVoIT S.r.l. (logo in figura 1.1) con sede a Pavoda presso il centro direzionale La Cittadella. iVoxIT S.r.l. con Athesys

Figura 1.1: Logo aziendale iVoIT

S.r.l. e Monokee S.r.l. fa parte di un gruppo di aziende fondato nel 2010 dall'unione di professionisti dell' Information Tecnology[g](IT) con l'obiettivo di fornire consulenza ad alto livello tecnologico e progettuale. Tra le altre cose, Athesys S.r.l fornisce supporto

[1] **womak:lean-thinking**.

nell'istanziazione del processo di Identity Access Management[g](IAM) , con particolare attenzione alla sicurezza nella conservazione e nell'esposizione dei dati sensibili gestiti. L'azienda opera in tutto il territorio nazionale, prevalentemente nel Nord Italia, e vanta esperienze a livello europeo in paesi quali Olanda, Regno Unito e Svizzera. Grazie all'adozione delle best practise[g]definite dalle linee guida Information Technology Infrastructure Library[g](ITIL) e alla certificazione ISO 9001 il gruppo è in grado di assicurare un'alta qualità professionale.

1.2 L'idea

Nell'ottica di estendere le funzionalità del prodotto Monokee di Identity Access Management basato su Cloud, lo stage ha visto lo sviluppo di due moduli applicativi in ambito Blockchain. Il primo modulo è una'applicazione mobile (Wallet) contenente l'identità digitale dell'utente finale mentre il secondo modulo è un layer applicativo (Service Provider) per gestire gli accessi alle applicazioni di terze parti. In figura 3.1 un'immagine dei moduli da implementare e il loro posizionamento in un tipico scenario di accesso ai servizi. La tipologia di Blockchain da integrare è stata individuata in una

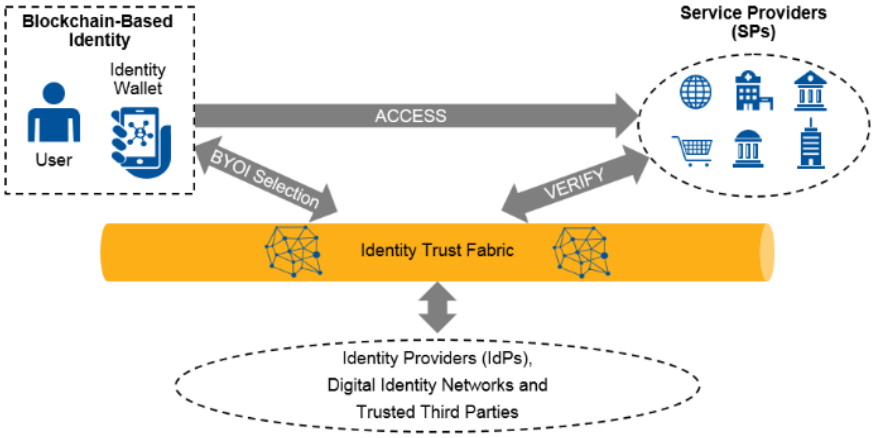


Figura 1.2: Diagramma Moduli

prima prima fase di analisi.

1.3 Organizzazione del testo

Il secondo capitolo descrive ...

Il terzo capitolo approfondisce ...

Il quarto capitolo approfondisce ...

Il quinto capitolo approfondisce ...

Il sesto capitolo approfondisce ...

Nel settimo capitolo descrive ...

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*[g];
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Processi e metodologie

Brevissima introduzione al capitolo

2.1 Processo sviluppo prodotto

Durante ogni attività è stata seguita una metodologia di sviluppo agile. L'azienda iVoxIT S.r.l. per la precisione attua in ogni suo progetto il metodo Scrum. Le attività sono state descritte in task secondo la modalità Scrum; ogni task veniva esposto e discusso in riunioni giornaliere con il tutor aziendale Dott. Sara Meneghetti. Inoltre erano previste riunioni settimanali con il responsabile del progetto Ing. Roberto Griggio.

2.1.1 Metodologie di sviluppo Agile

le metodologie di sviluppo agile si basano su quattro principi cardine:

* il software funzionante prima dei documenti;
* il rapporto con il cliente;
* i rapporti interno al team;
* rispondere al cambiamento;

Dal momento che i processi di pianificazione necessari ad uno sviluppo a cascata sono molto costosi e che se questi non vengono rispettati tutta la pianificazione deve essere completamente rivista, queste metodologie si basano su modelli incrementali. Gli approcci agile tendono a progettare il minimo indispensabile in modo tale da essere sempre più reattivi e proattivi possibili agli inevitabili. Inoltre, scrivere del software incrementale permette una progettazione iniziale molto snella, la quale con l'accrescere della conoscenza sul dominio può diventare sempre affinata. Questo in conclusione rende meno costoso il refactoring del codice e anche l'inserimento di nuove funzionalità. Questi metodi sono adatti per piccoli team, molto coesi e uniti, non dislocati in regioni diverse in quanto la comunicazione di persona è fondamentale. Il team in cui ero inserito difatti si componeva di tre membri. Un altro punto critico è che essendo l'approccio alla comprensione dei requisiti e alla progettazione poco concentrato all'inizio e molto diluito in tutto il progetto questo rende necessario un costante interesse da parte degli

stakeholders, cosa che in un nuovo progetto è ipotizzabile, mentre in un progetto in fase manutentiva no.

2.1.2 Scrum

Scrum è un metodo iterativo che divide il progetto in blocchi rapidi di lavoro chiamati Sprint, della durata massima di quattro settimane. Alla fine di ogni Sprint si ottiene un incremento del prodotto. Durante ogni fase dello stage si è seguito questo modello. Scrum prevede una prima fase di analisi e progettazione di massima, poi il lavoro viene suddiviso in unità' creabili e implementabili in un unico sprint e messi in un backlog, Prima di ogni sprint si sceglie una selezione di lavori in base alle priorità e si inseriscono nel backlog dello sprint. Gli sprint durano da 1 a 4 settimane e se il lavoro non viene portato a termine non viene prolungato lo sprint, ma rimesso nel backlog principale. Emerge come sia importante dare il giusto quantitativo di lavoro. È compito dello Scrum Master assicurarsi del rispetto dei tempi: quotidianamente effettua una breve riunione, della durata di circa quindici minuti, per valutare i progressi fatti (Daily Scrum). I rapporti tra i menbri del progetto sono stati importanti e per questo Scrum prevede riunioni giornaliere per rimanere aggiornati sullo stato dei lavori di ogni componente. In Figura 2.1 è mostrato un tipico ciclo Scrum.

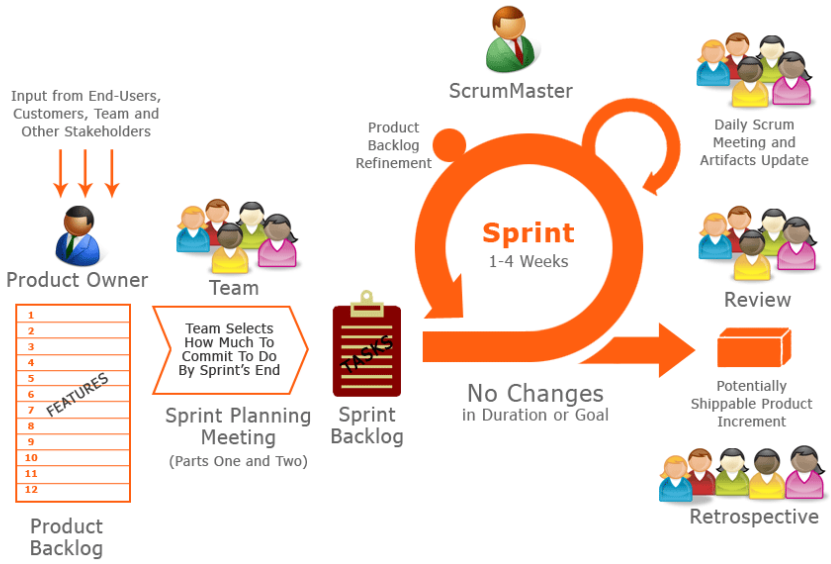


Figura 2.1: Diagramma flusso Scrum

Capitolo 3

Descrizione dello stage

Breve introduzione al capitolo

3.1 Introduzione al progetto

3.1.1 Descrizione del prodotto

Il progetto ha come scopo la creazione di un'estensione del servizio MonoKee basato su blockchain. L'estensione offre un sistema di Identity Access Management (IAM) composto da quattro principali fattori:

* **Identity Wallet** (IW);
* **Service Provider** (SP);
* **Identity Trust Fabric** (ITF);
* **Trusted Third Party** (TTP);

In sintesi l'estensione dovrà operare al fine di fornire la possibilità ad un utente di registrare e gestire la propria identità autonomamente tramite l'IW, mandare i propri dati (PII) all'ITF, il quale custodirà la sua identità e farà da garante per le asserzioni proveniente dai TTP. Inoltre il SP dovrà essere in grado con le informazioni provenienti da IW e ITF di verificare o meno l'accesso ai propri servizi. L'immagine in figura 3.1 dovrebbe chiarificare i vari componenti in gioco.

3.2 Studio Tecnologico Identity Trust Fabric

3.2.1 Sintesi dello studio tecnologico

Il capitolo procede descrivendo le caratteristiche del prodotto MonoKee e di come la tecnologia blockchain si possa collocare in un tale contesto. Vengono poi trattati i principali strumenti e librerie disponibili per sviluppare in Ethereum. L'analisi conclude facendo emergere come un utilizzo di Ethereum sia possibile, ma comunque non consigliato a causa di questioni prettamente legate alla scalabilità del sistema. Per ragioni di facilità di sviluppo e di time to market si è ritenuto comunque di adottare la scelta di Ethereum come base del componente ITF.

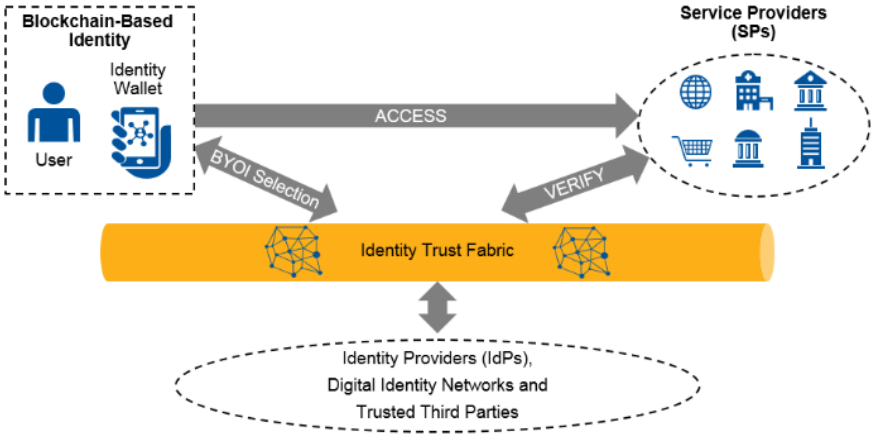


Figura 3.1: Diagramma Moduli

3.2.2 ITF – Identity Trust Fabric

Sulla base di un primo studio di fattibilità l'unico componente coinvolto nell'uso blockchain è l'Identity Trust Fabric. La sua principale funzione è quella di poter permettere ai vari Service Provider aderenti al servizio di poter verificare le informazioni rilasciate dai vari utenti tramite l'utilizzo del loro personale Identity Wallet (IW). Il componente mantiene al suo interno l'hash della chiave pubblica degli utenti (che rappresenta la loro identità) e le asserzioni fornite dai vari IW che possono essere potenzialmente certificate da una TTP (tramite una firma con la loro chiave privata). Le asserzioni devono poter essere modificate o eliminate in ogni momento, ovviamente ogni alterazione deve essere ogni volta certificata nuovamente. Anche da parte del TTP ci dev'essere la possibilità di revocare la certificazione di un'asserzione. Secondo lo studio Gartner in nota[1] una buona implementazione di una ITF deve possedere le seguenti caratteristiche:

* **Fiducia**: il contenuto presente nella ITF deve essere solo quello autorizzato, non ci devono poter essere manomissioni malevoli da parte degli utilizzatori della rete. Ogni componente deve potere aver fiducia nella veridicità dei dati.
* **Garanzia**: le regole logiche della ITF non devono poter essere manomesse. Deve essere possibile applicare le varie policy aziendali in ambito di gestione dei rischi.
* **Tracciabilità**: ogni informazione e cambio di stato relativo alle identità e alle asserzioni deve poter essere tracciato e verificabile sia in termini cronologici sia in termini di provenienza.
* **Sicurezza**: intesa come CIA. L'ITF deve rispettare i vincoli di confidenzialità, inalterabilità e disponibilità delle informazioni dentro lei contenute.
* **Scalabilità**: l'ITF deve fornire un elevato grado di scalabilità soprattutto in un'ottica in cui il prodotto potrebbe essere reso disponibile ad un uso Consumer.
* **Efficienza**: il funzionamento dell'ITF deve richiedere la minima quantità di risorse possibili.

[1] **farah:The-Dawn-of-Decentralized-Identity**.

3.2.3 Permissionless e Permissioned blockchain

Al fine di poter valutare la fattibilità dell'utilizzo di Ethereum quale blockchain sottostante all'ITF è necessario avere in mente le due principali categorie di blockchain: **permissionless** e **permissioned**.

Permissioned blockchain

Una permissioned blockchain pone dei vincoli sulla partecipazione alla rete. Soli i nodi autorizzati possono partecipare all'algoritmo di consenso dei blocchi. Le autorizzazioni possono essere date singolarmente, quindi i vari nodi possono avere o meno le seguenti possibilità:

* lettura dei blocchi;
* scrittura dei blocchi;
* esecuzione di codice (se prevista dalla blockchain);
* verifica dei nodi.

Permissionless blockchaing

Una permissionless blockchain è una rete in cui qualsiasi nodo può partecipare al processo di verifica dei blocchi. Ogni nodo ha tutte le precedenti quattro proprietà.

Ethereum Ethereum è una piattaforma decentralizzata pubblica ed open-source basata sulla creazione di SmartContract. Permette la creazione di applicazioni che operano su blockchain in modo che non ci sia alcuna possibilità di downtime, censura, frodi o interferenze da terze parti. Rappresenta una dei principali esempi di rete permissionless. La piattaforma è stata rilasciata nel corso del 2014 ed è mantenuta dalla Ethereum Foundation. Questo fa di Ethereum una delle più longeve blockchain disponibili. Ciò comporta la presenza di una documentazione abbastanza nutrita rispetto ai competitor e di un buon numero di strumenti già disponibili. L'elevata popolarità della tecnologia e alcune sue caratteristiche non presenti nei competitor, ha fatto sì che una notevole quantità di sviluppatori abbiano deciso di utilizzarla. Il sito [2] mantiene una vetrina di oltre milleseicento esempi. Sono presenti tutte le più significative applicazioni ora in produzione. Si fa notare che molte delle quali sono state le fonti dei più diffusi pattern Ethereum.

Programmare SmartContract Solidity è il principale linguaggio di programmazione usato per scrivere SmartContract. Nonostante sia presente un'implementazione basata su Go, questa è ancora acerba e non largamente utilizzata. Per questo motivo questa implementazione non verrà trattata in questo documento. Solidity è un linguaggio di programmazione ad oggetti ad alto livello. Il suo sviluppo è stato fortemente influenzato da linguaggi quali C++, Python e Javascript. Gli SmartContract così scritti vengono poi trasformati in bytecode e quest'ultimo viene eseguito dall'Ethereum Virtual Machina (EVM). Il linguaggio seppur non completamente maturo offre la maggior parte delle caratteristiche tipiche di un linguaggio ad oggetti. Infatti Solidity è fortemente tipato, supporta l'ereditarietà, librerie esterne e tipi definiti dall'utente. A sottolineare la bontà del linguaggio si evidenzia come in Solidity sia presente il

[2] site:state-dapps

concetto di interfaccia, caratteristica non presente in linguaggi ben più longevi. Queste caratteristiche rappresentano un notevole vantaggio per Ethereum rispetto ai diretti competitor, i quali spesso utilizzano linguaggi acerbi e/o a basso livello. Si ritiene che un linguaggio con le precedentemente descritte caratteristiche sia fondamentale per la buona riuscita del progetto, soprattutto in un'ottica di manutenibilità e estendibilità.

Breve nota sull'applicabilità dei pattern Nonostante il linguaggio permetta l'applicabilità dei più diffusi pattern si vuole far notare come nel contesto di una blockchain Permissionless questi risultino spesso controproducenti. Durante la progettazione e l'applicazione dei pattern vanno sempre ricordati i seguenti punti:

* l'esecuzione di un metodo che modifica la blockchain si paga in base al lavoro che viene effettivamente svolto;
* complessità lineari portano a costi difficilmente accettabili;
* plugin come Metamask calcolano il massimo costo possibile di una transazione, in caso il credito non sia sufficiente la transazione fallisce. Quindi un ciclo for su una lista di un elemento viene stimato presupponendo che la lista sia completamente piena;
* la velocità di esecuzione varia in base alla somma pagata per questa, anche con somme estremamente alte o su reti locali i tempi potrebbero essere considerati non giustificabili per la maggior parte degli utenti;
* ogni oggetto e campo dato si paga in base al loro spazio occupato;
* il costo della moneta e quindi delle transazioni è fortemente variabile. Approcci un giorno economici possono diventare economicamente insostenibili a distanza di giorni.

A seguito dei precedenti punti dovrebbe risultare più evidente come pattern che prevedono alta complessità temporale e spaziale siano inaccettabili su una rete Ethereum. Ad esempio i pattern Command e Decorator risultano difficilmente giustificabili. Sono invece presenti pattern pensati appositamente per Ethereum, questi sono presenti nella documentazione ufficiale Solidity[3]. Particolarmente utili al contesto del progetto in esame ritengo possano utili i seguenti pattern:

* Owner Pattern;
* Vote Pattern
* WhiteList Pattern.

Complessità e pratiche non convenzionali I punti precedentemente stilati nel paragrafo sull'applicabilità dei pattern portano anche delle notevoli differenze in termini di pratiche di stile di programmazione. Tra queste riporto:

* l'uso di liste e array sono fortemente sconsigliato, vanno preferite struttura con accesso costante. Solidity fornisce il tipo mapping;
* la creazione di oggetti (in termini Solidity contratti) ha un costo notevole. Una buona pratica è quella di utilizzare ADT (Abstract Data Type) differenti come le strutture;

[3] **site:solidity-documentation**.

* cicli for che portano complessità lineare dovrebbero essere evitati, elaborazioni di questo tipo dovrebbero essere affidate a server esterni o a livello client-side;
* l'utilizzo dei puntatori (in Solidity address) nasconde completamente il tipo dell'oggetto puntato rendendo vano il controllo dei tipi. Andrebbe evitato il più possibile.

Si fa notare come in particolare l'ultimo punto degeneri completamente il concetto di programmazione ad alto livello.

Strumenti Come già citato la relativa maturità della tecnologia ha portato alla creazione di alcuni utili strumenti.

* **Truffle**: è una suite di development e testing. Permette di compilare, buildare ed effettuare la migrazione degli SmartContract. Inoltre ha funzioni di debugging e di scripting. La suite offre la possibilità di effettuare test degli SmartContract sia in Javascript (con l'utilizzo di Chai), sia in Solidity. Si riporta di seguito il sito del progetto: **site:truffle**
* **Ganache**: è uno strumento rapido che permette di creare e mantenere in locale una rete blockchain Ethereum personale. Può essere usata per eseguire test, eseguire comandi e per operazioni di controllo dello stato mentre il codice esegue. Si riporta di seguito il sito del progetto: **site:ganache**
* **Mist**: è un browser sviluppato direttamente dal team Ethereum in grado di operare transazioni direttamente nella blockchain senza la necessità di possedere un intero nodo. È estremamente immaturo e non utilizzabile in produzione. Si riporta di seguito il sito del progetto: **site:mist**
* **Parity**: è un client Ethereum che permette di operare sulla rete senza necessità di possedere un intero nodo. Questa soluzione a differenza di Mist dovrebbe risultare più facilmente integrabile nel prodotto senza che l'utente ne abbi consapevolezza. Si riporta di seguito il sito del progetto: **site:parity** .
* **Metamask**: è uno plugin disponibile per i browser Chrome, Firefox, e Opera che permette di interfacciarsi alla rete Ethereum senza la necessità di eseguire in intero nodo della rete. Il plugin include un wallet con cui l'utente può inserire il proprio account tramite la chiave privata. Una volta inserito l'account il plugin farà da tramite tra l'applicazione e la rete. Metamask è utilizzato dalla maggioranza delle applicazioni Ethereum presenti on line, questo però rappresenterebbe un componente esterno compatibile con pochi browser desktop. Si riporta di seguito il sito del progetto: **site:metamask** .
* **Status**: è un progetto che propone una serie di Application Program Interface che permettono di sviluppare un'applicazione mobile nativa operante direttamente su blockchain senza la necessità di possedere un intero nodo. Il sito del progetto propone una serie di applicazioni che utilizzano Status tuttavia nessuna di queste applicazioni risulta attualmente rilasciate in nessuno store. Status risulta in early access ed è disponibile per Android e iOS. Il sito del progetto è il seguente: **site:status**
* **Microsoft Azure**: "Ethereum Blockchain as a Service" è un servizio fornito da Microsoft e ConsenSys che permette di sviluppare a basso costo in un ambiente

di dev/test/produzione. Permette di creare reti private, pubbliche e di consorzio. Queste reti saranno poi accessibili attraverso la rete privata Azure. Questa tecnologia rende facile l'integrazione con Cortana Analytics, Power BI, Azure Active Directory, O365 e CRMOL.

Valutazione applicabilità soluzione Ethereum Al fine di poter valutare correttamente da ogni punto di vista l'applicabilità di una soluzione basata su Ethereum quale base della componente ITF, si procede ad analizzare in maniera analitica le sei caratteristiche presentate nel capitolo 'ITF – Identity Trust Fabric'.

* **Fiducia**
  Questa caratteristica è ottenuta da Ethereum da una combinazione di diversi fattori quali:

  – utilizzo di incentivi economici, il pagamento per effettuare operazioni;
  – utilizzo di prove di interesse (Proof of Interest).

  Le prove di interesse possono essere di due tipi:

  – Proof of Stake, l'esibizione di un interesse;
  – Proof of Work, l'uso di potenza di calcolo per risolvere un problema matematico. Queste metodologie fanno in modo che solo chi realmente interessato possa influenzare l'algoritmo di consenso dei blocchi. Questo rende minore la possibilità di un "51 percent attack"[4]. C'è comunque da ricordare che un attacco di questo tipo e praticamente impossibile.

  Per queste ragioni si ritiene una rete Ethereum sia completamente soddisfacente per quanto riguarda l'aspetto fiducia, al pari di una rete di tipo permissioned.

* **Garanzia**
  Lo studio[5] evidenzia come questo rappresenti un punto critico. Infatti riporta che il raggiungimento di questo obiettivo è fortemente condizionata dall'efficacia dell'algoritmo di consenso e dai nodi presenti nella rete. Lo studio prosegue facendo notare che la presenza di nodi malevoli, oltre che mettere a rischio l'algoritmo di consenso, può compromettere anche il corretto funzionamento dell'ITF. Infatti trattandosi di una blockchain pubblica ogni nodo è in grado di visionare il contenuto di ogni singolo contratto, inclusi i dati e i metodi presenti. Per quanto riguarda i dati questo potrebbe non essere un problema in quanto si può immagazzinare una versione codificata del dato. Per quanto riguarda i metodi questo non è possibile, questo potrebbe rendere in grado ad un attaccante di trovare eventuali bachi e criticità dell'ITF. Il servizio Azure potrebbe permette di creare reti private.

* **Tracciabilità**
  Lo studio Gartner[6] evidenzia come in una rete permissionless la tracciabilità temporale non sia possibile, in quanto in una rete distribuita ogni nodo può avere un concetto di tempo proprio. Questo però non risulta possibile in nessun approccia risolutivo all'ITF basato su blockchain. Infatti le reti permissioned

[4] **site:51-attack**.
[5] **farah:The-Dawn-of-Decentralized-Identity**.
[6] **farah:The-Dawn-of-Decentralized-Identity**.

applicano timestamp a livello di blocco e non di transazione, anche ammettendo che ci sia un concetto di tempo comune tra i nodi, le transizioni rimarrebbero temporalmente non tracciabili. La cosa potrebbe permettere ad un blocco di alterare l'ordine delle transazioni. Questo problema in una rete permissioned può essere risolto creando blocchi immutabili e ogni volta si voglia fare una modifica si dovrà creare un nuovo blocco. In questo modo ci sarà solo una transazione di creazione blocco il cui timestamp coinciderà con il timestamp del blocco. Questo approccio in Ethereum rimane comunque impraticabile. Attualmente non sono note ulteriori tecniche per la tracciabilità temporale in Ethereum. Per questa ragione l'attribuzione di un riferimento temporale dovrà essere effettuato lato client, con i conseguenti limiti di sicurezza.

* **Sicurezza**
  La confidenzialità dei dati anche se non presente nativamente in Ethereum e facilmente ottenibile immagazzinando nei contratti solo un hash dei dati. L'integrità dei dati invece è garantita dalla prova di lavoro che utilizza la blockchain come già ribadito nella sezione Fiducia. La disponibilità invece è garantita dalle caratteristiche di distribuzione di ogni blockchain. Un ulteriore punto di considerazione da fare è che chiunque ha la possibilità di vedere il contenuto di ogni SmartContract incluso il codice dei metodi. Questo come già detto può comportare la possibilità da parte di un attaccante di individuare eventuali errori logici. Ogni contratto dovrà comunicare con gli altri attraverso chiamate a metodi pubblici, in quanto non c'è in Ethereum nessun concetto di visibilità dei metodi di tipo protected o package. Questo rende possibile da parte di qualsiasi utente della rete di utilizzare questi metodi in maniera malevole. Questo tipo di problematica è facilmente superabile applicando i dovuti pattern Solidity quali WhiteList Pattern e Owner Pattern. L'applicazione dei pattern però comporterebbe un notevole aumento in termini di complessità e costo soprattutto in presenza di logiche di accesso variegate e dinamiche. Inoltre, in caso di liste di utenti autorizzati l'immagazzinazione di queste liste potrebbe risultare oneroso in termini di costi.

* **Scalabilità**
  Ethereum per poter applicare l'algoritmo del consenso fa utilizzo di una prova di lavoro. Questa deve essere fatta in occasione di ogni transazione. La prova consiste nella risoluzione di un problema crittografico la cui difficoltà è dinamica in base a diversi fattori della blockchain quali valore dell'Ether, numero di utenti, numero di transazioni, etc. Inoltre si nota come anche in lettura ci sia una lentezza che difficilmente potrebbe essere ritenuta accettabile da un utente medio. Per avere prova di questo fatto si può prendere in esame una qualsiasi Dapp presente al seguente link[7]. La questione pone anche limitazioni come già citato in termine di costo.

3.2.4 Conclusioni

Da quanto è emerso l'utilizzo della tecnologia Ethereum quale base dell'ITF pone una serie di vantaggi e svantaggi. Di seguito si propone una sintetica trattazione dei punti fondamentali, per maggiori dettagli si consiglia la lettura dell'intero documento. I vantaggi sono:

[7] **site:state-dapps**.

* Ethereum offre un linguaggio ad alto livello e ad oggetti a differenza di altri competitor;
* Ethereum offre una notevole maturità e anche un'ampia platea di strumenti, molti dei quali estremamente maturi e largamente utilizzati.

Gli svantaggi sono:

* Ethereum è una rete pubblica, non è possibile fare nessuna restrizione di privilegi sui nodi partecipanti alla rete. Questo potrebbe rappresentare un problema di sicurezza;
* la comunicazione verso dispositivi mobili non è verificabile da quest'ultimi, in quanto dovrebbe avvenire tramite comunicazione REST;
* sono presenti forte limitazioni in termini di costo e velocità, il sistema risulterebbe lento ed estremamente costoso. Ciò comporta notevoli problemi in termini di Scalabilità del servizio.

Si ritiene che un approccio basato sul Ethereum sull'ITF sia possibile, le eventuali criticità di sicurezza e fiducia verso i dispositivi mobili sono superabili con una buona progettazione. L'unico fattore veramente critico risulta la scalabilità del sistema. Quest'ultimo fatto a mio parere è sufficiente per ritenere Ethereum non adatto all'utilizzo soprattutto in un'ottica commerciale. Quindi se pure possibile, non si consiglia l'utilizzo di Ethereum. Per ragioni di facilità di sviluppo e di time to market si è ritenuto comunque di adottare la scelta di Ethereum come base del componente ITF.

3.3 Studio di fattibilità Identity Wallet

3.3.1 Sintesi dello studio di fattibilità

Lo studio inizia descrivendo come l'IW si cali in questo contesto. Si prosegue analizzando le alternative di sviluppo mobile e Desktop. A seguito di un'analisi dei possibili utenti emerge una netta preferenza per lo sviluppo mobile. Vengono poi trattati i principali strumenti e librerie disponibili per lo sviluppo. Si ritiene di preferire uno sviluppo multi piattaforma, con target Android e iOS. L'analisi conclude facendo emergere una preferenza per il framework Xamarin.

3.3.2 Descrizione componente Identity Wallet

Il progetto ha come scopo la creazione di un Identity Wallet (IW). L'applicativo si colloca nel contesto di un'estensione del servizio Monokee basato su blockchain. L'estensione offre un sistema di Identity Access Management composto da quattro principali fattori:

* Identity Wallet (IW)
* Service Provider (SP)
* Identity Trust Fabric (ITF)
* Trusted Third Party (TTP)

In sintesi l'estensione dovrà operare al fine di fornire la possibilità ad un utente di registrare e gestire la propria identità automatamente tramite l'IW, mandare i propri dati (IPP) all'ITF la quale custodirà la sua identità e farà da garante per le asserzioni proveniente dai TTP. Inoltre il SP dovrà essere in grado con le informazioni provenienti da IW e ITF di garantire o meno l'accesso ai propri servizi. Il software IW, più dettagliatamente, dovrà assolvere ai seguenti compiti: nell'ambito della registrazione di un utente il Wallet deve:

* generare e immaganizzare in locale una chiave pubblica;
* generare e immaganizzare in locale una chiave private;
* creare l'hash della chiave la chiave pubblica e inviare l'hash all'ITF;
* incrementare le informazioni (PII) relative alle identità che il Wallet gestisce.

Nell'ambito della presentazione dei dati ad un Service Provider deve:

* invio della chiave pubblica al service provider;
* invio di un puntatore all'hash della chiave pubblica interna al ITF;
* invio di altre informazioni utili presenti nel ITF;
* gestire ulteriori layer di sicurezza, quali impronta digitale, QR code, autentificazione multi fattore)

nell'ambito della richiesta di accesso ad un servizio deve:

* inviare una richiesta di accesso ad un servizio con i dati relativi all'identità al Service Provider;
* attendere la risposta del Service Provider.

3.3.3 Studio del dominio

Dominio Applicativo

L'applicativo IW dovrà essere usato in un contesto prevalentemente lavorativo. Non si escludono però ulteriori applicazione future in ambito Consumer. In ogni caso il software dovrà poter essere usato da utenti senza specifiche conoscenze informatiche e con minimo training tecnologico. Dato il contesto applicativo il software dovrà essere il più accessibile e semplice possibile. Per queste ragioni si pensa ad un suo utilizzo prevalentemente in ambito mobile, anche se non si esclude a priori la possibilità di una versione Desktop. L'applicativo mobile deve essere disponibile per la più ampia gamma di utenti possibili.

Dominio Tecnologico

Un eventuale applicativo Desktop ha un'elevata probabilità di non rientrare dentro i tempi dello stage, per questo si ritiene di non tenerlo in considerazione. L'applicativo quindi dovrà essere fruibile tramite un'applicazione mobile multipiattaforma sviluppabile entro i limiti temporali della durata dello stage. Per queste ragioni lo studio del dominio tecnologico si incentrerà principalmente su tecnologie multi piattaforma mobili. Verrà comunque tenuto in considerazione anche lo sviluppo nativo.

Studio diffussione sistemi operativi mobili Procedo di seguito ad un'analisi sulla diffusione dei vari sistemi operativi mobili. I dati in figura 3.2 riportati provengono da Kantar[8] società di analisi inglese e fanno riferimento al trimestre che va da novembre 2016 a gennaio 2017. Da come si può evincere dai dati, Android, iOS, Windows Phone

Smartphone OS Sales Share (%)

Germany	3 m/e Jan'16	3 m/e Jan'17	% pt. Change
Android	74.2	75.5	1.3
iOS	19.3	21.3	2.0
Windows	5.9	2.9	-3.0
Other	0.7	0.2	-0.5
GB	**3 m/e Jan'16**	**3 m/e Jan'17**	**% pt. Change**
Android	52.6	54.4	1.8
iOS	38.6	43.3	4.7
Windows	8.6	1.9	-6.7
Other	0.2	0.3	0.1
France	**3 m/e Jan'16**	**3 m/e Jan'17**	**% pt. Change**
Android	71.9	72.9	1.0
iOS	19.3	24.2	4.9
Windows	7.8	2.8	-5.0
Other	0.9	0.2	-0.7
Italy	**3 m/e Jan'16**	**3 m/e Jan'17**	**% pt. Change**
Android	78.1	79	0.9
iOS	14.4	15.8	1.4
Windows	7.2	4.4	-2.8
Other	0.3	0.8	0.5
Spain	**3 m/e Jan'16**	**3 m/e Jan'17**	**% pt. Change**
Android	87.8	89.4	1.6
iOS	11.4	10.2	-1.2
Windows	0.8	0.4	-0.4
Other	0	0	0.0

USA	3 m/e Jan'16	3 m/e Jan'17	% pt. Change
Android	58.2	56.4	-1.8
iOS	39.1	42	2.9
Windows	2.6	1.3	-1.3
Other	0.1	0.3	0.2
China	**3 m/e Jan'16**	**3 m/e Jan'17**	**% pt. Change**
Android	73.9	83.2	9.3
iOS	25.0	16.6	-8.4
Windows	0.9	0.1	-0.8
Other	0.3	0.1	-0.2
Australia	**3 m/e Jan'16**	**3 m/e Jan'17**	**% pt. Change**
Android	52.6	55.7	3.1
iOS	41.2	42.4	1.2
Windows	5.4	1	-4.4
Other	0.8	0.8	0.0
Japan	**3 m/e Jan'16**	**3 m/e Jan'17**	**% pt. Change**
Android	48.7	49	0.3
iOS	50.3	49.5	-0.8
Windows	0.5	1.5	1.0
Other	0.5	0	-0.5
EU5	**3 m/e Jan'16**	**3 m/e Jan'17**	**% pt. Change**
Android	72.9	74.3	1.4
iOS	20.3	22.7	2.4
Windows	6.4	2.7	-3.7
Other	0.5	0.3	-0.2

Figura 3.2: Diffuzione sistemi mobili

rappresentano, in questa sequenza ed in ogni mercato, i sistemi più diffusi. I restanti sistemi non raggiungono cifre significative. Ponendo maggiore attenzione ai primi tre sistemi si nota come Android nell'area EU5 rappresenti i tre quarti del mercato. In Giappone, Stati Uniti, Australia e Gran Bretagna invece la situazione risulta più bilanciata con una sostanziale parità. Windows Phone in ogni mercato si pone in terza posizione con percentuali che non superano mai l'otto percento. Individuando nell'area EU5 il principale mercato per MonoKee si ritiene che il prodotto IW debba essere sviluppato per i sistemi Android e iOS, dando la precedenza al primo. Non si ritiene necessario lo sviluppo di un'applicazione Windows Phone in quanto difficilmente attuabile nei tempi dello stage.

Tecnologie per lo sviluppo Segue un approfondimento relativo alle potenziali tecnologie con cui sviluppare l'IW. Data la necessità di sviluppare sia per Android, che per iOS l'analisi si concentrerà principalmente su framework multi piattaforma senza comunque ignorare la possibilità di sviluppi nativi.

Sviluppo multipiattaforma Tra le principali alternati multi piattaforma si ritengono particolarmente interessanti le seguenti:

[8] **site:kantar-study**.

* React Native;
* Cordova;
* Xamarin;

Segue un'analitica descrizione dei vari framework.

React Native: è un framework di sviluppo mobile derivato da React. Il progetto e sviluppato e mantenuto da Facebook. React Native si focalizza nello sviluppo di UI tramite componenti scritti in JavaScript, su un approccio funzionale e flusso di dati unidirezionale. A differenza di React, React Native non manipola il DOM del browser, ma una struttura diversa. I componenti non vengono scritti a partire da elementi HTML o simili (i.e. Bootstrap o Grommet), ma bensì a partire da un set di componenti base presenti nella libreria. La libreria permette di sviluppare applicazioni per iOS e Android.

Cordova: è un framework open-source per lo sviluppo di applicazione mobili che propone un approccio ibrido e non nativo. Permette di usare tecnologie web ampiamente utilizzate, quali HTML5, CSS3, Javascript, per la codifica. Il software così prodotto verrà eseguito in appositi wrapper diversi per ogni piattaforma, quindi in maniera non nativa. Il framework è sviluppato da Apache ed ormai ha raggiunto un elevato grado di maturità. Rappresenta uno dei primi framework per lo sviluppo multi piattaforma.

Xamarin: è un framework per lo sviluppo di applicazioni native e multi piattaforma con C Sharp. Il framework si basa sul progetto open source Mono e offre pieno supporto non solo alle piattaforme Android e iOS ma anche a Windows Phone. La possibilità di sviluppare anche per Windows Phone potrebbe risultare un punto a favore rispetto agli altri framework. Xamarin si compone di tre componenti principali: Xamarin.Android, Xamarin.iOS, Xamarin.Forms. L'ultimo componente si pone come strumento completamento neutro rispetto alla piattaforma. Grazie a queste componenti è possibile gestire in C Sharp tutte le caratteristiche di Android, iOS e Windows Phone.

La tabella 3.1 riassume quanto appena detto. Data l'impossibilità degli approcci

Tabella 3.1: Tabella comparivi framework sviluppo applicazioni mobili

Framework	Approccio	Piattaforme supportate	Linguaggio
React Native	Nativo	iOS, Android	Javascript
Cordova	Ibrido	iOS, Android	HTML5, CSS3, Javascript
Xamarin	Nativo	iOS, Android, WP	C Sharp

ibridi, quali Cordova, di sfruttare a pieno le caratteristiche tipiche delle diverse piattaforme mobili si ritiene di scartare questo tipo di soluzioni. Inoltre si evidenziano difetti come una mancata o incompleta integrazione dell'aspetto grafico con la specifica piattaforma e una maggiore lentezza nell'esecuzione e accesso alle risorse locali. Per queste ragione si ritiene più opportuno l'utilizzo di un framework che permetta di scrivere applicazioni in maniera nativa. Richiudendo la visione ai soli approcci nativi, Xamarin rispetto a React Native lascia aperte le porte ad una eventuale applicativo Windows Phone. Inoltre utilizza C Sharp un linguaggio che rispetto a Javascript fornisce una tipizzazione forte e caratteristiche più orientate agli oggetti. Per queste ragioni si consiglia l'utilizzo di Xamarin o React Native con la preferenza per il primo.

Sviluppo Nativo Un'applicazione nativa è un'applicazione mobile sviluppata interamente nel linguaggio del dispositivo sul quale vengono eseguite. Quindi stiamo parlando di Java per Android e Swift o Object-C per iOS. Il loro utilizzo presenta diversi vantaggi rispetto allo sviluppo multi piattaforma:

* interazione con tutte le caratteristiche del dispositivo consentendo l'utilizzo al 100%;
* maggiore velocità offrendo quindi una User Experience di più alto livello;
* facilità di integrazione di terze parti tramite utilizzo di SDK ufficiali.

Il primo punto non dovrebbe rappresentare un plus in quanto non si ritiene che l'IW debba usufruire di feature particolari dei dispositivi. È da notare che uno sviluppo nativo richiede il doppio delle risorse necessarie in quanto prevede lo sviluppo di due applicazione completamente diverse (Android e iOS), con framework e quindi architetture potenzialmente diverse. Riassumendo, dato che:

* l'applicativo che si dovrà sviluppare non prevede particolari requisiti prestazionali;
* l'alto costo in termini orari di sviluppare soluzioni differenti ha una forte probabilità di non rientrare nei tempi previsti dall'attività di stage;

si ritiene non conveniente lo sviluppo parallelo di più applicazioni native.

Conclusioni sulla scelta del framework A seguito di quanto detto nelle sezioni "Sviluppo multipiattaforma" e "Sviluppo nativo" si ritiene quindi più conveniente lo sviluppo di un'applicazione multi piattaforma. Nello specifico si consiglia l'utilizzo di framework quali React Native e Xamarin con la preferenza di quest'ultimo.

Breve considerazione sullo sviluppo mobile Dall'analisi del dominio applicativo emerge come un'applicazione di tipo mobile sia la scelta più adatta. Questa scelta è basato sulla tipologia di utenti e sul tipico uso ipotizzato per l'applicazione. Tuttavia come precedentemente detto l'obbligatorietà di comunicare con l'ITF tramite API REST snaturerebbe il concetto stesso di blockchain, in quanto l'IW vedrebbe il componente REST come fonte centralizzata e non verificabile delle informazioni. Per queste ragioni si vuole far notare come un'applicazione Desktop risulterebbe notevolmente più appropriata dal punto di vista tecnologico, ma fuori contesto dal punto di vista dell'uso previsto. Al fine di effettuare una scelta finale bisogna tenere sempre in menti questi due fattori e considerare cosa si vuole ottenere. Ritengo che l'utente dell'IW non sia in grado di apprezzare questo concetto di fiducia e che potrebbe apprezzare maggiormente il fatto che l'applicativo sia mobile.

3.3.4 Motivazioni

Aspetti Positivi

A seguito dell'analisi sopra proposta sono stati individuati i seguenti aspetti positivi:

* lo sviluppo di un'applicazione mobile Android e iOS porterebbe MonoKee alla portata della quasi totalità dei possibili utenti;
* uno sviluppo con un framework multi piattaforma abbatterebbe i costi di produzione dell'applicazione, pur garantendo risultati accettabili;

* i framework multi piattaforma portati in esame (Xamarin e React Native) sono ampiamente utilizzati e supportati da grandi aziende IT. Questo garantisce un elevato grado di affidabilità e una ampia documentazione;
* un eventuale uso di Xamarin potrebbe facilitare una successiva implementazione di un'applicazione Windows Phone.
* seppur MonoKee utilizza una soluzione basata su blockchain, l'IW non risulta colpito da questa ulteriore complessità.

Fattori di rischio

Durante la fase di analisi iniziale sono stati individuati alcuni possibili rischi a cui si potrà andare incontro. Si è quindi proceduto a elaborare delle possibili soluzioni per far fronte a tali rischi.

1. Comunicazione IW-ITF
Descrizione: La comunicazione tra IW e ITF dovrebbe avvenire attraverso chiamate alla blockchain. Questo comporta l'uso di librerie per dispositivi mobili poco collaudate e piene di incognite..
Soluzione: Provvedere ad una comunicazione basata su protocollo RESTful..

2. Visione centralizzata
Descrizione: Un'applicazione mobile di questo tipo per l'IW potrebbe non essere considerata come strumento di IAM distribuito, ma potrebbe essere vista come centralizzata..
Soluzione: Inserire note interno all'applicazione o all'interno del sito di Monokee per rendere edotti gli utenti del reale funzionamento del servizio..

3. Inesperienza nello sviluppo Xamarin
Descrizione: Il team non ha esperienza nello sviluppo di applicazioni mobili..
Soluzione: Rendere edotto il responsabili del progetto, il quale mettera a disposizione del personale per impartire lezioni sul framework Xamarin..

3.3.5 Conclusione Studio di fattibilità IW

Da questo primo studio di fattibilità emerge come, da un punto di vista dell'utente, lo sviluppo di un'applicazione mobile sia maggiormente adatto. Invece, da un punto di vista tecnologico, risulta come ci siano delle problematiche inerenti alla comunicazione tra i componenti IW e ITF. Riguardo questo si ritiene che lo sviluppo di un applicativo Desktop risulterebbe più adatto, ma molto probabilmente mal visto dalla maggioranza degli utenti finali. Per quanto detto si conclude ribadendo la fattibilità del progetto come applicazione mobile sviluppata con un framework multi piattaforma. Per la scelta del framework si consiglia Xamarin.

3.4 Studio di fattibilità Service Provider

3.4.1 Sintesi dello studio di fattibilità

Lo studio inizia descrivendo come il SP si cali in questo contesto. Si prosegue analizzando il dominio applicativo. Da questo emerge un utilizzo da personale specializzato

in orario lavorativo. Si è effettuata, poi, un'analisi sui due principali tipi di sviluppo: distribuito o centralizzato. Alla fine di una breve analisi emerge una preferenza per l'ultimo. All'interno del documento sono presenti anche una trattazione di una serie di tecnologie (sia a livello di librerie per la comunicazione con la rete, che di librerie front end) che lo sviluppo di un applicativo di questo tipo potrebbe avere bisogno.

3.4.2 Descrizione Service Provider

Il progetto ha come scopo la creazione di un componente chiamato Service Provider (SP). L'applicativo si colloca nel contesto di un'estensione del servizio Monokee basata su blockchain. L'estensione offre un sistema di Identity Access Management composto da quattro principali fattori:

* Identity Wallet (IW);
* Service Provider (SP);
* Identity Trust Fabric (ITF);
* Trusted Third Party (TTP).

In sintesi, l'estensione dovrà operare al fine di fornire la possibilità ad un utente di registrare e gestire la propria identità automatamente tramite l'IW, mandare i propri dati (PII) all'ITF la quale custodirà la sua identità e farà da garante per le asserzioni proveniente dalle TTP. Inoltre, il SP dovrà essere in grado con le informazioni provenienti dall'IW e dall'ITF di garantire o meno l'accesso ai propri servizi. Si fa notare come il componente SP non rappresenta il reale fornitore del servizio, ma solo un elemento dell'architettura che lo rappresenta. Il reale servizio viene erogato da organizzazioni esterne le quali comunicano con il componente SP per garantire o meno l'accesso. Il software SP, più dettagliatamente, dovrà assolvere ai seguenti compito: nell'ambito della ricezione dei dati da un Identity Wallet (IW) deve:

* ricevere da parte dell'IW la chiave pubblica (o l'hash di questa);
* ricevere un riferimento alla locazione dell'hash della chiave pubblica all'interno dell'ITF;
* ricevere altre informazioni necessarie da parte dell'IW con relativo riferimento all'interno dell'ITF;
* gestire il trasferimento dei dati tramite codice QR.

Nell'ambito della verifica dei dati provenienti dall'IW deve:

* usare la chiave pubblica dell'IW e il riferimento per verificare l'identità e le varie altre informazione passate dal wallet;
* generare e comparare gli hash dei valori ottenuti con quelli presenti nell'ITF;
* verificare che l'identità e le altre informazioni ottenute siamo sufficienti a garantire l'accesso al servizio.

Nell'ambito dell'accesso il SP deve:

* a seguito della verifica comunica il risultato all'organizzazione che fornisce il servizio, in modo tale da garantire l'accesso all'utente dell'IW.

3.4.3 Studio del dominio

Dominio applicativo

L'applicativo SP dovrà essere usato come strumento abilitatore da parte dei vari fornitori di servizi a partecipare al progetto MonoKee. Da un primo studio si pensa che il target di questi servizi sarà lavorativo, successivamente potrà essere considerato l'introduzione di servizi Consumer. Si tratta sostanzialmente di un'applicazione di tipo Server con scopi essenzialmente di comunicazione. Data la vasta varietà di servizi e di necessità che potrebbe avere il fornitore non risulta definibile un comportamento standard che il SP dovrà tenere, ma si dovrà adattare caso per caso. Possiamo comunque ipotizzare che il suo funzionamento sia necessario solo durante l'orario di ufficio, quindi dalle 7.00 alle 18.00, fuori questi orari sarà possibile fare manutenzione. Nell'ottica dell'introduzione di servizi consumer si dovrebbe comunque tener conto di una disponibilità maggiore. L'applicativo dovrebbe offrire un'interfaccia di manutenzione, accessibile tramite interfaccia grafica da parte del personale del fornitore del servizio. Il software deve essere utilizzato dal personale IT delle varie organizzazione che utilizzano il servizio, per questa ragione si può dare per scontato che l'utente generico possegga delle competenze informatiche avanzate.

3.4.4 Dominio tecnologico

Il service provider deve operare come intermediario tra l'IW, l'ITF e il reale fornitore del servizio. Le comunicazioni dovrebbero seguire lo schema proposto in figura 3.3. A seguito dello studio di fattibilità relativo all'IW è emerso come la connessione tra IW e SP debba avvenire tramite protocollo REST. Mentre la comunicazione con l'ITF deve avvenire tramite blockchain. Relativamente alla comunicazione verso il fornitore vero e proprio non si possono fare considerazione in quanto queste possono variare significativamente.

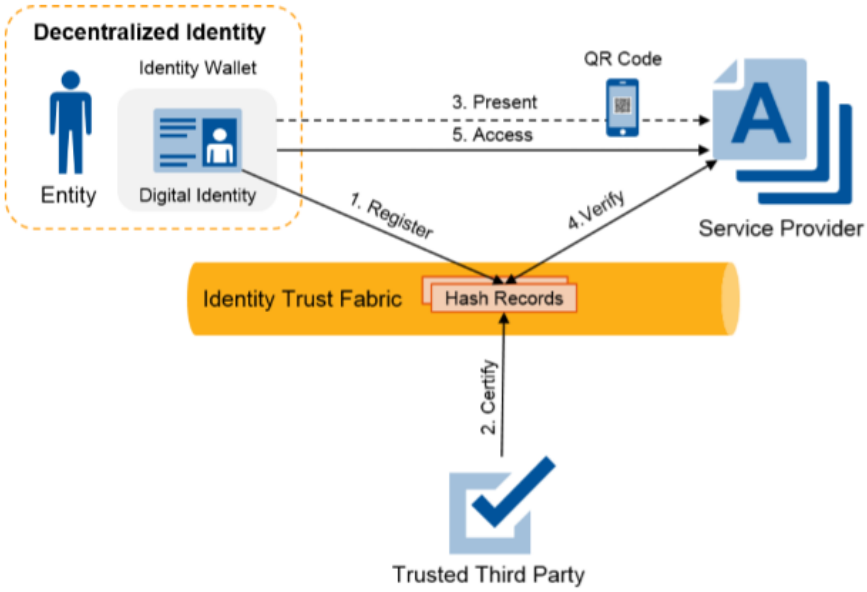


Figura 3.3: Diagramma flussi tra i vari componenti

Si evidenziano essenzialmente due principali opzioni per la costruzione di questo applicativo. Il primo è l'utilizzo, anche per esso come per l'ITF, di un approccio totalmente distribuito basato su blockchain. Il secondo approccio consiste i un'applicazione tradizionale.

Sviluppo distribuito

Questo approccio prevede che la logica applicativa sia totalmente affidata a codice eseguito su blockchain. Questo comporterebbe una disponibilità continuativa sempre garantita e altre caratteristiche di affidabilità e sicurezza. Come punto negativo si evidenzia che una soluzione del genere implicherebbe l'uso di molte tecnologie non completamente mature e di linguaggi in molti casi incompleti. Inoltre questa scelta implicherebbe l'utilizzo della stessa blockchain presente nell'ITF. I vantaggi attribuibili a questo approccio non sono considerati fondamentali al fine di una buona implementazione del SP, di contro gli svantaggi risultano particolarmente pesanti. L'applicazione di un approccio di questo genere anche se possibile risulta sconsigliato. Uno sviluppo di questo tipo, almeno in reti di tipo Permissionless, comporta un cambio di stile di programmazione dovuto all'alto costo delle operazioni. Come descritto nello studio tecnologico Ethereum ciò comporta le seguenti diversità:

* alcuni pattern non risultano applicabili;
* complessità lineari sono difficilmente giustificabili;
* uso di pattern ad hoc.

Sviluppo tradizionale

La seconda opzione risulta essere una più tradizionale applicazione server che comunica tramite librerie alla blockchain. Si consiglia l'uso di linguaggi fortemente tipati quali:

* C++;
* C Sharp;
* Java.

Data l'alta diffusione di Javascript e NodeJS nella comunità Ethereum si consigliano pure questi. In base al linguaggio scelto e alla tecnologia blockchain scelta, si dovranno utilizzare differenti librerie per effettuare la comunicazione tra la rete e l'applicativo. Nel caso di Ethereum si propongono le seguenti librerie:

Tabella 3.2: Tabella comparivi linguaggio per sviluppo SP

Linguaggio	**Libreria**	**Note**
C++	cpp-ethereum	-
C Sharp	Nethereum	Questa soluzione si collega particolarmente bene alla scelta di Xamarin per
JS e NodeJS	Web3	-
Java	Web3j	-

Di seguito si procederà alla trattazione di alcuni degli strumenti che si ritengono più utili.

Nethereum :è un tool che permette una facile integrazione con il client in applicazioni .NET. Fornisce una suite di librerie open source che aiutano a prototipare applicazione .NET velocemente. E' disponibile anche nel sotto insieme Xamarin. La documentazione è presente è sembra ben strutturata e di ottima qualità. Inoltre potrebbe rappresentare una buona soluzione in caso di scelta di Microsoft Azure Blockchain. Il sito del progetto è il seguente www.nethereum.com.

Web3 : è una collezione di librerie che permettono di interagire con un nodo remoto o locale usando una connessione HTTP o IPC. Web3 è presente in npm, meteor, pure js. Per il suo funzionamento è necessario avere un client attivo nel proprio computer. Web3 supporta Mist e Metamask. Il sito del progetto è il seguente: web3js.readthedocs.io.

Web3J : si tratta di una libreria analoga a Web3 per Java.

Mist : è un browser sviluppato direttamente dal team Ethereum in grado di operare transazioni direttamente nella blockchain senza la necessità di possedere un intero nodo. È estremamente immaturo e non utilizzabile in produzione. Si riporta di seguito il sito del progetto: github.com/ethereum/mist.

Metamask : è uno plugin disponibile per i browser Chrome, Firefox e Opera che permette di interfacciarsi alla rete Ethereum senza la necessità di eseguire in intero nodo della rete. Il plugin include un wallet con cui l'utente può inserire il proprio account tramite la chiave privata. Una volta inserito l'account il plugin farà da tramite tra l'applicazione e la rete. In caso, invece, si opti per la scelta di Hyperledger la scelta risulterebbe molto più semplice in quanto la blockchain in questione fornisce un'API per la comunicazione REST.

Scelta del client Ethereum

Il client è un componente che implementa il protocollo di comunicazione di Ethereum. Ethereum diversamente da Hyperledger presenta una realtà molto varia e frattagliata. Solo dal punto di vista del client ci sono multiple implementazioni per differenti sistemi operativi ed in differenti linguaggi. Questa diversità viene vista dalla community come un indicatore di salute per l'intero ecosistema. Il protocollo in ogni caso è sempre lo stesso ed è definito nel così detto Yellow Paper in nota 5, in sostanza si basa sull'utilizzo di file Json. Fino al settembre 2016 erano presente le alternative esposte in tabella 3.3. I client appena citati sono accessibili tramite apposite librerie, un buon esempio

Tabella 3.3: Tabella comparivi client Ethereum

Client	**Linguaggio**	**Sviluppatore**
Go-ethereum	Go	Ethereum Foundation
Parity	Rust	Ethcore
Cpp-ethereum	C++	Ethereum Foundation
Pyethapp	Python	Ethereum Foundation
Ethereumjs-lib	Javascript	Ethereum Foundation
Ethereum(J)	Java	<ether.camp>
Ruby-ethereum	Ruby	Jan Xie
EthereumH	Haskell	BlockApps

potrebbe essere web3 per Javascript.

Scelta del client Hyperledger

In caso si dovesse optare per una scelta basata su Hyperledger quale base dell'ITF bisognerà ricadere sull'unica soluzione proposta dal team di sviluppo, cioè quella di utilizzare direttamente una comunicazione REST. Il team offre uno strumento detto Composer con il quale si potrà definire un'interfaccia per operare la comunicazione REST. Al fine di poter gestire efficientemente queste chiamate lato applicazione SP si consigliano le librerie esposte in tabella 3.4. Data l'amplissima diffusione delle API

Tabella 3.4: Tabella comparivi client Ethereum

Linguaggio	Librerie	Sito
.NET	WCF REST Starter Kit	www.asp.net/downloads/starter-kits/wcf-rest
.NET	OpenRasta	www.openrasta.org
.NET	Service Stack	www.servicestack.net
Java	Jersey	www.jersey.java.net
Java	RESREasy	www.jboss.org/resteasy
Java	Restlet	www.restlet.org
C++	linavajo	www.libnavajo.org
C++	C++ RESTful framework	www.github.com/corvusoft/restbed
C++	C++ REST SDK	www.github.com/Microsoft/cpprestsdk

REST e di queste librerie non si procede ad una trattazione analitica.

3.4.5 Conclusioni scelta sviluppo

Considerando quanto precedentemente detto lo sviluppo tradizionale sembrerebbe avrebbe meno incognite e un ampio repertorio di librerie utilizzabili. Si propone, quindi un'architettura del secondo tipo. Inoltre, si vuole fare notare come l'utilizzo delle soluzioni .NET potrebbero rivelarsi molto vantaggioso in quanto facilmente integrabili con il componente IW e Azure Blockchain.

3.4.6 Motivazioni

Aspetti positivi

A seguito dell'analisi sopra proposta sono stati individuati i seguenti aspetti positivi:

* lo sviluppo di un'applicazione server tradizionale comporta uno sviluppo molto semplice e immediato, grazie anche alla disponibilità di un ampio repertorio di librerie;
* la comunicazione con la blockchain risulta in ogni caso facilmente implementabile grazie all'utilizzo di apposite librerie.
* esiste un'ampia scelta di librerie front end per ogni possibile linguaggio di sviluppo.

Fattori di rischio

4. Inesperienza nello sviluppo C Sharp
Descrizione: Non si ha esperienza nello sviluppo di applicazioni C Sharp..
Soluzione: Rendere edotto il responsabili del progetto, il quale mettera a disposizione del personale per impartire supporto su C Sharp..

5. difficoltà di integrazione con Monokee
Descrizione: Il componete SP è il componente che deve essere integrato in Monokee.
Soluzione: Rendere edotto il responsabili del progetto, il quale mettera a disposizione del personale..

6. Prestazioni chiamate alla rete blockchain
Descrizione: Il componete SP deve interrogare la rete blockchain, questo potrebbe rappresentare un problema di performance.
Soluzione: Rendere edotto il responsabili del progetto, valutare soluzioni alternative..

3.4.7 Conclusioni

Dal presente studio emerge come la creazione di un SP sviluppato come applicativo server e non come applicazione distribuita possa essere un'ottima opzione implementativa. Lo studio non presenta particolari rischi. Si ritiene, quindi,che un approccio di questo tipo sia fattibile nei tempi dello stage.

3.5 Requisiti e obiettivi

3.6 Pianificazione

Capitolo 4

Analisi dei requisiti IW

Breve introduzione al capitolo
Questo capitolo ha lo scopo di fornire una definizione dei requisiti individua per la creazione del prodotto Identity Wallet (IW). Le metodologie usate sono tratte dal capitolo quattro di **som:swe** Più in particolare la presente capitolo si prefigge di:

* individuare le fonti per la deduzione dei requisiti;
* dedurre i requisiti dalle fonti;
* descrivere i requisiti individuati;
* catalogare i requisiti individuati;
* prioritizzare i requisiti individuati;

4.1 Specifiche in Linguaggio Naturale

Il linguaggio naturale ha un'enorme potenza espressiva ma, essendo inerentemente ambiguo, può portare ad incomprensioni. È quindi necessario limitarne l'utilizzo e standardizzarlo, in modo da ridurre al minimo le possibili ambiguità. È comunque fondamentale evitare di utilizzare espressioni e acronimi che possano essere fraintendibili dagli stakeholders, a tal proposito in fondo al documento è presente una lista degli acronimi utilizzato.

4.2 Specifiche in Linguaggio Strutturato

Il linguaggio strutturato mantiene gran parte dell'espressività del linguaggio naturale, fornendo però uno standard schematico che permette l'uniformità della descrizione dei vari requisiti. Sebbene l'utilizzo di un linguaggio strutturato permetta di organizzare i requisiti in modo più ordinato e comprensibile, talvolta la ridotta espressività rende difficile la definizione di requisiti complessi. A tal proposito è possibile integrare la specifica in linguaggio strutturato con una descrizione in linguaggio naturale.

4.3 Specifiche in Linguaggio UML Use Case

Per la definizione dei diagrammi UML dei casi d'uso, viene utilizzato lo standard UML 2.0[1]. Nei diagrammi dei casi d'uso vengono mostrati gli attori coinvolti in un'interazione con il sistema in modo schematico, indicando i nomi delle parti coinvolte. Eventuali informazioni aggiuntive possono essere espresse testualmente.

4.4 Casi d'uso

Per lo studio dei casi di utilizzo del prodotto sono stati creati dei diagrammi. I diagrammi dei casi d'uso (in inglese *Use Case Diagram*) sono diagrammi di tipo Unified Modeling Language (UML) dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso.

4.4.1 Descrizione Attori

I tipi di attori principali che andranno ad interagire direttamente con il sistema sono essenzialmente tre:

* utente;
* utente non registrato;
* utente autenticato.

Tra di essi è presente una relazione di generalizzazione che vede l'attore utente come generalizzazione degli attori utente non registrato e utente registrato. Questo tipo di generalizzazione viene rappresentata graficamente in figura 4.1. Sono stati individuati

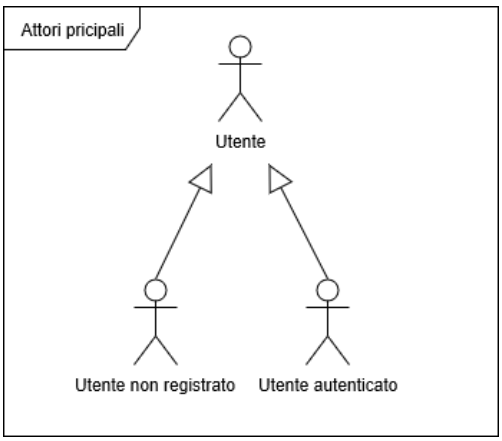


Figura 4.1: Gerarchia utenti user case

i seguenti attori secondari: ITF, MonoKee.

[1] **site:uml**.

Attori principali

* **Utente**: l'attore utente è un fruitore generico del sistema. Potrebbe avere o non avere effettuato l'accesso all'applicazione. Da lui derivano gli attori utente non registrato e utente autenticato.
* **Utente non registrato**: l'attore utente non registrato è una particolare specializzazione dell'attore utente. Unica sua caratteristica è quella di non essere riconosciuto come utente di MonoKee.
* **Utente autenticato**: l'attore utente autenticato è una particolare specializzazione dell'attore utente. Rappresenta un utente che ha effettuato l'accesso al sistema e che è stato riconosciuto all'interno del sistema MonoKee.

Attori secondari

* **ITF**: è il componente dell'estensione che ha il compito di conservare e convalidare tutte le informazioni provenienti dall'IW.
* **MonoKee**: è il componente centrare dell'attuale servizio MonoKee. Ha il compito di fornire le informazioni di accesso del servizio MonoKee.

4.4.2 Elenco casi d'uso

UC1: Azioni utente generico

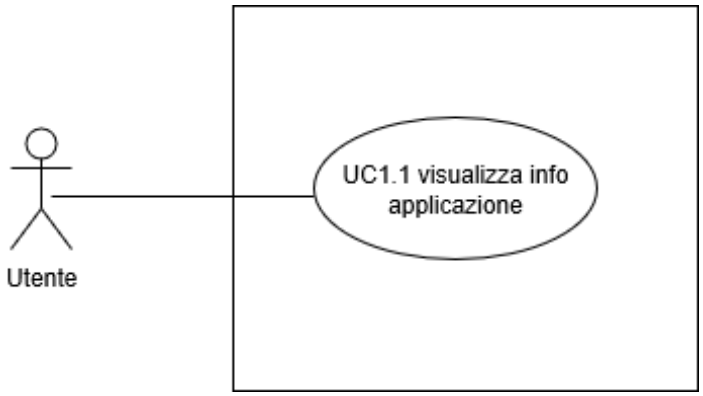


Figura 4.2: Use Case - UC1: Azioni utente generico

UC1.1 – Visualizza info applicazione

Tabella 4.1: Use Case - UC1: Azioni utente generico

Descrizione	L'utente può visualizzare le informazioni sull'applicazione
Attore primario	Utente
Attore secondario	Nessuno
Precondizioni	L'utente ha avviato l'applicazione
Postcondizioni	L'utente ha eseguito le azioni che desiderava compiere in relazione alle sue possibilità
Scenario principale	∗ UC1.1 Visualizza info applicazione
Scenari alternativi	Nessuno

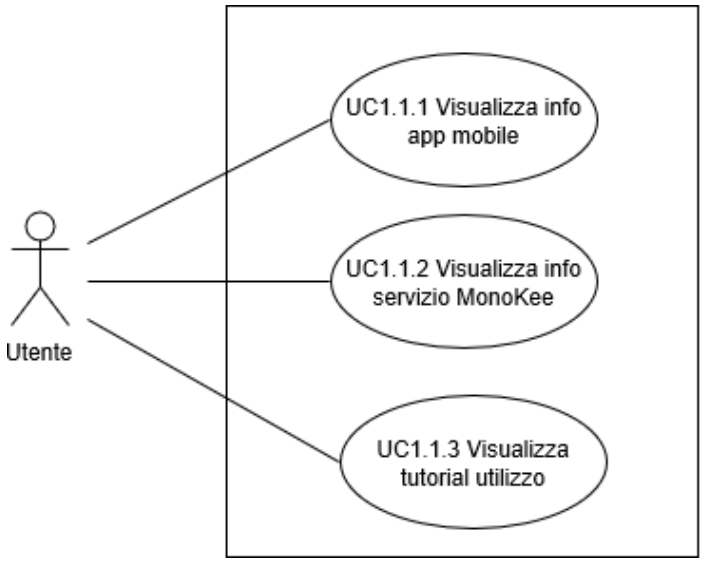


Figura 4.3: Use Case - UC1.1 – Visualizza info applicazione

Tabella 4.2: Use Case - UC1.1 – Visualizza info applicazione

Descrizione	Il sistema deve visualizzare le informazioni relative all'applicazione mobile e al servizio MonoKee
Attore primario	Utente
Attore secondario	Nessuno
Precondizioni	L'utente ha avviato l'applicazione
Postcondizioni	L'utente ha visualizzato le informazioni che desirava riguardo l'applicazione
Scenario principale	∗ UC1.1.1 Visualizza info applicazione ∗ UC1.1.2 Visualizza info servizio MonoKee ∗ UC1.1.3 Visualizza tutorial utilizzo
Scenari alternativi	Nessuno

4.5 Tracciamento dei requisiti

Da un'attenta analisi dei requisiti e degli use case effettuata sul progetto è stata stilata la tabella che traccia i requisiti in rapporto agli use case.
Sono stati individuati diversi tipi di requisiti e si è quindi fatto utilizzo di un codice identificativo per distinguerli.
Il codice dei requisiti è così strutturato R(F/Q/V)(N/D/O) dove:

R = requisito

F = funzionale

Q = qualitativo

V = di vincolo

N = obbligatorio (necessario)

D = desiderabile

Z = opzionale

Nelle tabelle 4.3, 4.4 e 4.5 sono riassunti i requisiti e il loro tracciamento con gli use case delineati in fase di analisi.

Tabella 4.3: Tabella del tracciamento dei requisti funzionali

Requisito	Descrizione	Use Case
RFN-1	L'interfaccia permette di configurare il tipo di sonde del test	UC1

Tabella 4.4: Tabella del tracciamento dei requisiti qualitativi

Requisito	Descrizione	Use Case
RQD-1	Le prestazioni del simulatore hardware deve garantire la giusta esecuzione dei test e non la generazione di falsi negativi	-

Tabella 4.5: Tabella del tracciamento dei requisiti di vincolo

Requisito	Descrizione	Use Case
RVO-1	La libreria per l'esecuzione dei test automatici deve essere riutilizzabile	-

Capitolo 5

Progettazione e codifica

Breve introduzione al capitolo

5.1 Tecnologie e strumenti

Di seguito viene data una panoramica delle tecnologie e strumenti utilizzati.

Tecnologia 1

Descrizione Tecnologia 1.

Tecnologia 2

Descrizione Tecnologia 2

5.2 Ciclo di vita del software

5.3 Progettazione

Namespace 1

Descrizione namespace 1.

Classe 1: Descrizione classe 1

Classe 2: Descrizione classe 2

5.4 Design Pattern utilizzati

5.5 Codifica

Capitolo 6

Verifica e validazione

Capitolo 7

Conclusioni

7.1 Consuntivo finale

7.2 Raggiungimento degli obiettivi

7.3 Conoscenze acquisite

7.4 Valutazione personale

Appendice A

Appendice A

Citazione

Autore della citazione

Bibliografia