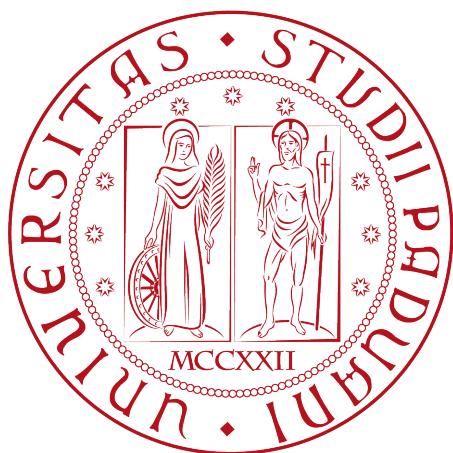


Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA “TULLIO LEVI-CIVITA”

CORSO DI LAUREA IN INFORMATICA



**VerifiedMovies: il cinema in piena sicurezza con l'uso della
blockchain**

Tesi di Laurea

Relatrice

Prof.ssa Ombretta Gaggi

Laureando

Gabriel Rovesti, Matricola 2009088

ANNO ACCADEMICO 2022–2023

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di tirocinio formativo, della durata di circa trecentoventi ore complessive, dal laureando Gabriel Rovesti presso l'azienda Sync Lab, sede di Padova. Durante il tirocinio era richiesto il raggiungimento di massima dei seguenti obiettivi.

- studio autonomo delle tecnologie *blockchain* e dei protocolli di comunicazione;
- lo sviluppo di un'applicazione decentralizzata per comunicare e leggere dati da un contratto intelligente;
- l'implementazione di un'interfaccia grafica per la visualizzazione degli eventi registrati;
- l'integrazione del progetto con una libreria esterna per l'autenticazione usando le tecnologie *Self Sovereign Identity* e *Zero Knowledge Proof*;
- la documentazione del lavoro svolto in un documento apposito dettagliante le tecnologie utilizzate, le scelte progettuali e le modalità di utilizzo del prodotto;
- l'utilizzo di *React* per la parte front-end e *Solidity* per la parte back-end di comunicazione con la *blockchain*;
- l'utilizzo di *TypeScript* per la parte di logica applicativa.

“But poetry, beauty, romance, love, these are what we stay alive for.”

— Dead Poets Society, Robin Williams

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine alla Prof.ssa Ombretta Gaggi, relatrice della mia tesi, per l'aiuto e il sostegno fornитоми durante la stesura del lavoro.

Desidero ringraziare con affetto mia mamma per il sostegno, il grande aiuto e per essermi stata vicina in ogni momento durante gli anni di studio.

Vorrei inoltre ringraziare i molti amici, di corso, di università e di vita, che mi hanno aiutato e sostenuto in questi anni e per le tante conoscenze che hanno spinto ulteriormente la mia voglia di fare e di conoscere.

Padova, Giugno 2023

Gabriel Rovesti, Matricola 2009088

Indice

1 Introduzione	1
1.1 L'azienda	1
1.2 Way of Working e strumenti	2
1.3 Organizzazione del testo	3
1.3.1 Struttura del documento	3
1.3.2 Convenzioni tipografiche	4
2 Tecnologie di interesse	5
2.1 Blockchain: concetti base	5
2.1.1 Introduzione	5
2.1.2 Blocco	5
2.1.3 Transazione	6
2.1.4 Wallet	6
2.1.5 Mining	7
2.1.6 Algoritmi di consenso	7
2.1.7 Tipi	9
2.2 Blockchain: concetti avanzati	9
2.2.1 Token	9
2.2.2 Tokenizzazione	10
2.2.3 Smart contract	11
2.2.4 Vulnerabilità e sicurezza	13
2.2.5 Scalabilità	14
2.3 Self-Sovereign Identity	15
2.3.1 Tipi e applicazioni	16
2.4 Zero Knowledge Proof	16
2.4.1 Tipi e applicazioni	17
3 Descrizione dello stage	19
3.1 Introduzione al progetto e idea dello stage	19
3.2 Analisi preventiva dei rischi	20
3.3 Obiettivi e requisiti	21
4 Analisi dei requisiti	23
4.1 Descrizione ed analisi del sistema	23
4.2 Casi d'uso	26
4.3 Tracciamento dei requisiti	39
5 Progettazione e codifica del progetto	43
5.1 Componenti principali del sistema	43
5.2 Tecnologie utilizzate	43
5.2.1 Codifica front-end	43

5.2.1.1	React	43
5.2.1.2	TypeScript	43
5.2.2	Codifica back-end	44
5.2.2.1	Solidity	44
5.2.3	Librerie di terze parti	44
5.2.3.1	Node.js	44
5.2.3.2	Web3.js	44
5.2.3.3	Hardhat	44
5.2.4	Versionamento	44
5.2.4.1	GitHub	44
5.2.5	Verifica	44
5.2.5.1	ESLint	44
5.2.5.2	Jest	45
5.3	Configurazione ambiente di sviluppo	45
5.3.1	Smart Contract	45
5.3.2	Front-end	45
5.4	Progettazione	46
5.4.1	Architettura e pattern front-end	46
5.4.2	Architettura e pattern back-end	47
5.5	Codifica	49
5.5.1	Codifica back-end	49
5.5.2	Codifica front-end	51
5.5.2.1	Pagine realizzate	51
5.5.2.1.1	Home Page	51
5.5.2.1.2	Registrazione	52
5.5.2.1.3	Login	53
5.5.2.1.4	Film	53
5.5.2.1.5	Prenotazione del film	56
5.5.2.1.6	Lista delle prenotazioni	57
5.5.2.1.7	Profilo	58
5.5.2.1.8	Errore 404	58
5.5.2.2	Componenti realizzati	59
5.5.2.2.1	AuthContext	59
5.5.2.2.2	Footer	59
5.5.2.2.3	NavBar	59
5.5.2.2.4	PrivateRoute	59
5.5.2.2.5	ScreenReaderHelp	59
5.5.2.2.6	SearchBox	59
5.5.2.2.7	Notification	59
6	Verifica e validazione	61
6.1	Accessibilità	61
6.2	Test di unità	62
6.3	Test di integrazione	63
6.4	Test di regressione	66
6.5	Accettazione e collaudo	67
7	Conclusioni	69
7.1	Obiettivi raggiunti e consuntivo finale	69

7.2 Conoscenze acquisite e analisi del lavoro svolto	69
7.3 Possibili sviluppi futuri e scenari di applicabilità	71
7.4 Valutazione personale	71
Glossario	73
Bibliografia	77

Elenco delle figure

1.1	Logo azienda Sync Lab	1
1.2	Esempio di utilizzo di Trello	2
2.1	Blockchain vista come blocchi a catena	6
2.2	Processo di conferma in Proof of Work	8
2.3	Funzionamento di uno Smart Contract	12
2.4	Processo di riconoscimento di una credenziale SSI	15
2.5	Processo di verifica di una ZKP	17
4.1	Scenario principale	27
4.2	UC1: Registrazione	27
4.3	UC4: Login	29
4.4	UC6: Visualizzazione lista film	30
4.5	UC8: Prenotazione film	32
4.6	UC9: Visualizzazione lista prenotazioni	34
4.7	UC9.1: Visualizzazione singola prenotazione in lista	35
4.8	UC12: Modifica informazioni profilo	38
5.1	Schema logico applicazione React	47
5.2	Architettura Ethereum	48
5.3	Architettura applicazione	49
5.4	Pagina principale	51
5.5	Pagina principale vista da cellulare	51
5.6	Pagina principale	52
5.7	Meccanismo di conferma identità dell'utente	52
5.8	Pagina di login	53
5.9	Pagina dei film dell'applicazione	53
5.10	Finestra di verifica dell'età	54
5.11	Esempio di caricamento durante la verifica dell'età	56
5.12	Recensione per un film	56
5.13	Condivisione di un film	56
5.14	Pagina di prenotazione del film selezionato	57
5.15	Finestra di conferma della prenotazione del film selezionato	57
5.16	Pagina della lista delle prenotazioni effettuate	57
5.17	Pagina di gestione dei dati dell'utente	58
5.18	Pagina di errore 404	58

Elenco delle tabelle

4.1	Tabella del tracciamento dei requisiti funzionali	40
4.2	Tabella del tracciamento dei requisiti di vincolo	40
4.3	Tabella del tracciamento dei requisiti qualitativi	41
6.1	Tabella di tracciamento dei test di unità	62
6.4	Tabella del tracciamento dei test di integrazione	63
6.8	Tabella del tracciamento dei test di regressione	66

Capitolo 1

Introduzione

In questo capitolo verrà descritta l'azienda sede del tirocinio, l'organizzazione del testo e delle convenzioni tipografiche impostate.

1.1 L'azienda

Sync Lab (logo in figura 1.1) è una *software house* italiana nata a Napoli nel 2002 che, grazie ad una progressiva maturazione delle sue competenze in ambito applicativo e tecnologico, è riuscita a diventare un punto di riferimento per le aziende che intendono innovare i propri processi di business, trasformandosi in un *System Integrator*. L'azienda si è fatta notare sul mercato grazie alla proposta di vari prodotti software, sviluppati all'interno del suo laboratorio di ricerca e sviluppo, conquistando importanti fette di mercato in ambito nazionale nei settori mobile, di sicurezza e videosorveglianza. Ad oggi, Sync Lab conta più di 150 clienti e oltre 200 dipendenti dislocati nelle proprie sedi. Queste sono situate a Napoli, Milano, Verona, Roma e Padova. L'obiettivo principale dell'azienda è quello di fornire soluzioni tecnologiche innovative e di qualità che possano soddisfare le esigenze dei propri clienti, garantendo un servizio di consulenza e assistenza continuo e di alto livello.



Figura 1.1: Logo azienda Sync Lab

1.2 Way of Working e strumenti

L'azienda Sync Lab adotta un modello di sviluppo ^g[Agile](#), con l'obiettivo di monitorare e controllare lo sviluppo del progetto in modo flessibile e continuo, suddividendo le attività in piccoli incrementi e con una collaborazione asincrona e distribuita. In particolare, il modello di sviluppo adottato è ^g[Scrum](#), che prevede la suddivisione del progetto in sprint, ovvero periodi di tempo di durata fissa, in cui vengono pianificate le attività da svolgere e i relativi obiettivi da raggiungere. Al termine di ogni sprint, viene effettuato su base settimanale un incontro interno approfondito con il tutor aziendale, per discutere lo stato di avanzamento del progetto e le attività da svolgere per il successivo sprint ([15]).

L'obiettivo del modello è dare maggiore importanza al ciclo di vita del ^g[software](#) e dei processi correlati, piuttosto che al prodotto finale, con l'obiettivo di migliorare la qualità del prodotto stesso. Inoltre, grazie alla collaborazione con altri stagisti con progetti basati sugli stessi concetti e in generale con il team di sviluppo, è stato possibile condividere conoscenze e competenze, discutere problemi e trovare soluzioni comuni, con un approccio incrementale e iterativo.

Gli strumenti utilizzati per lo sviluppo del progetto sono stati i seguenti:

- **Trello**, per la gestione delle attività e dei task da svolgere (esempio in figura 1.2), condiviso con il tutor aziendale e verificato dallo stesso ad ogni incremento. All'interno di questo strumento è possibile monitorare le attività attraverso delle schede, simili a dei *post-it*, differenziando le attività per specifiche colonne di avanzamento. Le diciture riportate dalle colonne sono generalmente come segue:
 - **Backlog**: che contiene attività in corso di svolgimento, comprendenti schede con obiettivi di massima dello stage e delle sue singole parti e periodi;
 - **In corso**, ossia attività in esecuzione e da realizzare entro la fine dello *sprint*;
 - **In verifica**, ossia attività in attesa di verifica e considerate concluse, in attesa di essere spostate nella colonna *Terminati* a seguito della *sprint review*;
 - **Terminati**, cioè attività completate e verificate da parte del tutor aziendale.

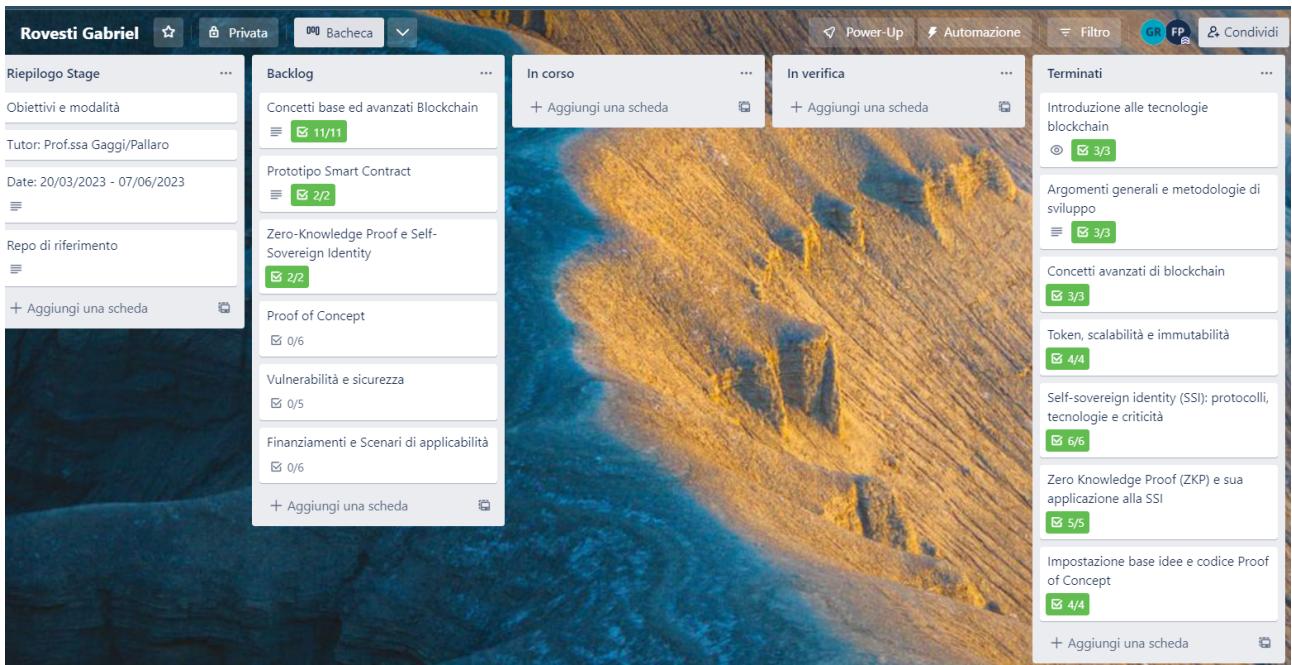


Figura 1.2: Esempio di utilizzo di Trello

- **Google Calendar**, per la pianificazione delle attività e degli incontri con il tutor aziendale, condiviso con il tutor stesso e con gli altri stagisti e dipendenti di Sync Lab, capendo così e in anticipo le attività da svolgere e gli incontri da effettuare;
- **Visual Studio Code**, ^gIDE utilizzato per la scrittura del codice sorgente e dei file di configurazione, con l'aggiunta di alcuni *plugin* per la formattazione del codice e per la gestione dei file di configurazione del progetto. All'interno di questo, è stato realizzato lo studio di esempi di codice e la realizzazione dell'intero progetto, con l'aggiunta di un sistema di *linting* per la formattazione del codice e di un sistema di *debugging* per il controllo del flusso di esecuzione del codice. Sempre tramite questo strumento, è stata scritta la tesi, attraverso l'utilizzo di un *plugin* per la scrittura in L^AT_EX;
- **Diagrams.net**, per la realizzazione e la creazione delle immagini del documento, utilizzati per la documentazione e la strutturazione delle sezioni;
- **StarUML**, per la realizzazione dei diagrammi ^gUML del documento, quindi per i diagrammi dei casi d'uso e dei diagrammi delle classi;
- **GitHub**, utilizzato internamente per la gestione del codice sorgente e dei file di configurazione, condiviso con il tutor aziendale e verificato dallo stesso ad ogni incremento. In particolare, è stato utilizzato il sistema di controllo versione ^gGit, che permette di tenere traccia delle modifiche effettuate ai file e di gestire le varie versioni del codice e degli appunti interni da me realizzati, per una migliore organizzazione e condivisione delle informazioni. Inoltre, è stato possibile condividere il codice sorgente con gli altri stagisti e dipendenti di Sync Lab, per garantire il versionamento e la condivisione delle modifiche effettuate.
- **Discord, social network** utilizzato per la comunicazione interna tra i dipendenti di Sync Lab, tramite la gestione di vari canali vocali e testuali divisi per argomento e per sedi interne aziendali. La suddivisione in gruppi e canali ha permesso di comunicare in modo efficace e veloce, condividendo informazioni e risorse utili per lo svolgimento del progetto, sia a livello di risorse didattiche che per la risoluzione di eventuali dubbi e problemi. Inoltre, è stato possibile comunicare con gli altri stagisti in modo semplice e veloce, condividendo conoscenze e competenze e discutendo problemi e soluzioni comuni.

1.3 Organizzazione del testo

1.3.1 Struttura del documento

Il secondo capitolo descrive le tecnologie studiate e utilizzate per lo sviluppo del progetto.

Il terzo capitolo approfondisce il percorso di tirocinio formativo, descrivendo precisamente gli obiettivi e le attività svolte durante il percorso, con relativa analisi dei rischi.

Il quarto capitolo approfondisce il progetto da un punto di vista tecnico, descrivendo i requisiti e i casi d'uso del progetto.

Il quinto capitolo approfondisce le tecnologie utilizzate per la realizzazione del progetto, descrivendo le scelte progettuali e le soluzioni implementate, a livello architettonico e di codifica.

Il sesto capitolo approfondisce le attività di verifica e validazione del progetto, descrivendo le tipologie di test effettuate e i risultati ottenuti.

Nel settimo capitolo descrive lo stato degli obiettivi raggiunti e la valutazione personale del percorso, con un'analisi retrospettiva del progetto e del suo utilizzo rispetto a sviluppi futuri sulle stesse tematiche.

1.3.2 Convenzioni tipografiche

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;
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*;
- tutti i termini appartenenti al glossario, sono evidenziati e presentano una lettera G ad apice del termine presente. Tale dicitura è presente per la prima apparizione del termine nel capitolo in oggetto.

Capitolo 2

Tecnologie di interesse

In questa sezione viene presentata una panoramica di base delle tecnologie oggetto del mio tirocinio, al fine di descrivere in modo chiaro e conciso i concetti di base e le caratteristiche principali delle tecnologie utilizzate nel progetto di stage e oggetto di studio autonomo e autodidatta.

2.1 Blockchain: concetti base

2.1.1 Introduzione

La [blockchain](#) è una tecnologia che permette di memorizzare dati in maniera decentralizzata e distribuita. Essa è una struttura dati che si comporta come un registro distribuito, salvando le informazioni in modo sicuro ed immutabile. La struttura è stata introdotta nel 2008 da Satoshi Nakamoto, che ha pubblicato il suo white paper *Bitcoin: A Peer-to-Peer Electronic Cash System*. Nel 2009 è stato pubblicato il primo *software open source* per la *blockchain*, Bitcoin, che ha permesso di creare una moneta digitale decentralizzata.

A tal fine, non si devono confondere le cosiddette criptovalute con la *blockchain*. Di fatto, quest'ultima è solo la struttura che permette lo scambio di beni di qualsiasi tipo, in modo sicuro, registrato ed immutabile. Una criptovaluta è invece una moneta digitale, che può essere scambiata con altre monete digitali o con beni fisici. Essendo lo standard *blockchain open source*, è possibile crearne di nuove con molta facilità.

Normalmente, viene utilizzata per memorizzare transazioni finanziarie, ma può essere utilizzata per memorizzare qualsiasi tipo di informazione. Un altro suo nome è *distributed ledger technology* (DLT), che indica che le sue informazioni sono registrate come su un libro mastro, in cui le singole componenti della rete, definite nodi, possono accedere e modificare i dati, stabilendo se questi sono validi o meno.

2.1.2 Blocco

I dati delle *blockchain* sono strutturati in singoli blocchi, ciascuno contenente uno specifico set di informazioni. Ogni blocco è collegato al precedente tramite un hash, che ne garantisce l'immutabilità. I blocchi sono collegati in una catena, che viene aggiornata ogni volta che viene aggiunto un nuovo blocco. Nello specifico, possiamo dettagliare una struttura formata dai seguenti (come descritto in figura 2.1):

- blocchi di dati;
- nonce, un numero generato casualmente alla creazione del blocco;
- l'hash del blocco precedente;
- il numero della transazione;
- *timestamp* di generazione del blocco (data e ora).

Per verificare l'integrità dei dati memorizzati, vengono usate delle strutture dati chiamati *Merkle trees*, in cui ogni foglia rappresenta l'hash della transazione. Le foglie vengono poi raggruppate in coppie, e l'hash di ogni

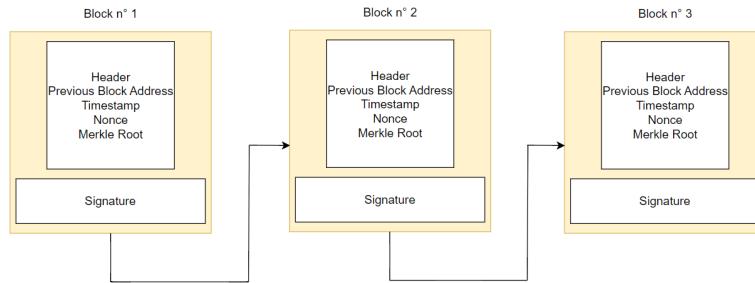


Figura 2.1: Blockchain vista come blocchi a catena

coppia viene calcolato e memorizzato in un nodo superiore. Il processo si ripete, fino a raggiungere la radice dell'albero, che rappresenta l'hash di tutte le transazioni contenute nel blocco.

2.1.3 Transazione

Una transazione all'interno di una *blockchain* comporta il trasferimento di beni digitali, che possono essere valute, token, o qualsiasi altro tipo di informazione. In questo senso, possiamo individuare vari componenti del processo di transazione:

- gli utenti, che avviano le transazioni firmandole digitalmente con la propria chiave privata;
- i *miners*, che attraverso un processo specifico definito come *mining*, verificano la validità delle transazioni e le includono nel blocco successivo.
- i nodi, che convalidano i blocchi di transazioni inviati dai miners prima che vengano aggiunti alle *blockchain*.

Nello specifico, possiamo descrivere una transazione in questo modo:

1. l'utente avvia la transazione creando una firma digitale utilizzando la propria chiave privata. La firma dimostra che l'utente ha il diritto di inviare i beni;
2. la transazione viene trasmessa alla rete di nodi o computer che eseguono il *software* della *blockchain*. Ogni nodo riceve la transazione e la aggiunge a un pool di transazioni non confermate;
3. i nodi della rete convalidano la transazione per assicurarsi che il mittente abbia fondi sufficienti per completare la transazione e che questa sia conforme alle regole del protocollo *blockchain*;
4. una volta che un numero sufficiente di nodi ha convalidato la transazione, questa viene aggiunta a un nuovo blocco di transazioni, insieme ad altre transazioni convalidate di recente;
5. il blocco di transazioni viene aggiunto alla *blockchain* in un processo chiamato *mining*. L'estrazione comporta la risoluzione di complesse equazioni matematiche per creare un nuovo blocco, il che richiede una grande potenza di calcolo;
6. una volta aggiunto il nuovo blocco alla *blockchain*, la transazione viene considerata confermata e i beni vengono trasferiti dall'indirizzo del mittente a quello del destinatario. La transazione è ora registrata in modo permanente sul libro mastro della *blockchain*, che può essere visualizzato e verificato da chiunque abbia accesso alla rete;

2.1.4 Wallet

Un *wallet*, detto anche *portafoglio*, è un software che permette di memorizzare e gestire le chiavi private e pubbliche, e di inviare e ricevere transazioni. In particolare, possiamo più propriamente definirli portachiavi, in quanto non contengono realmente i beni digitali, ma le chiavi utilizzate per accedervi. L'utente dispone in ogni momento di:

- una chiave pubblica, usata per inviare messaggi e ricevere pagamenti. È un codice univoco che identifica l'utente;
- una chiave privata, usata per firmare i messaggi e per accedere ai propri beni digitali. È un codice segreto che deve essere conservato in modo sicuro.

Ogni *wallet* dispone di una frase segreta, che contiene tutte le informazioni necessarie per recuperare ed accedere ai fondi del proprio portachiavi.

Inoltre, dispone di un proprio indirizzo, matematicamente derivato dalla stessa chiave pubblica mediante l'operazione di *hashing*, con una lunghezza di 160 bit. Ciascuno è *pseudonimo*, in quanto non appartiene nello specifico ad una persona, ma non è completamente anonimo.

È importante tenere la chiave privata in un luogo sicuro e non condividerla con nessuno, in quanto è l'unica cosa che garantisce l'accesso ai propri fondi. Distinguiamo due tipi di *wallet*:

- *hot wallet*, che sono i portafogli online, dunque più vulnerabili al rischio di *hacking*;
- *cold wallet*, che sono i portafogli offline, quindi considerati più sicuri, in quanto si collegano ad Internet principalmente per effettuare le transazioni.

2.1.5 Mining

Il processo di *mining* consente di creare nuovi blocchi sulla catena, al fine di convalidare le transazioni e ottenere nuove criptovalute come ricompensa per il proprio ‘sforzo’. Questo obiettivo viene raggiunto attraverso un processo chiamato ‘consenso’, che prevede la risoluzione di complessi puzzle matematici utilizzando la potenza di calcolo. Il miner che riesce a risolvere il puzzle prima degli altri, vince il diritto di aggiungere il blocco alla *blockchain*.

Questo processo è composto da due fasi:

- *hashing*, che consiste nella risoluzione di un *puzzle* matematico, che consiste nel trovare un numero che, una volta applicata una funzione di hash, abbia un valore inferiore ad un valore prefissato. Il valore di questo numero viene chiamato *nonce*;
- *ricerca del consenso*, che consiste nella verifica della validità del blocco, che viene effettuata da tutti i nodi della rete. Se il blocco è valido, viene aggiunto alla *blockchain*.

I minatori (miners) utilizzano un software speciale per risolvere il problema matematico incredibilmente complesso di trovare un *nonce* che generi un hash accettato. Poiché il *nonce* è di soli 32 bit e l'hash di 256, ci sono circa quattro miliardi di possibili combinazioni *nonce*-hash che devono essere estratte prima di trovare quella giusta.

Quando un blocco viene estratto con successo, la modifica viene accettata da tutti i nodi della rete e il *miner* viene ricompensato finanziariamente. Il primo minatore che risolve il puzzle e aggiunge un nuovo blocco alla *blockchain* viene ricompensato con un blocco di criptovaluta di nuovo conio. Il processo richiede un software specializzato e una grande potenza di calcolo, che può essere ottenuta utilizzando un computer o un gruppo di computer con grosso dispendio di energia e risorse.

2.1.6 Algoritmi di consenso

Il meccanismo di ricerca di consenso prevede numerose varianti, che differiscono per il modo in cui i nodi della rete si accordano per aggiungere un nuovo blocco alla *blockchain*.

Possiamo principalmente distinguere:

1. *Proof of Work (PoW)*, basato nella ricerca di un hash (una stringa di numeri e lettere) che soddisfa una determinata condizione, detta target. Tale condizione richiede che l'hash del nuovo blocco sia inferiore

a un determinato valore. Questo valore viene stabilito in base alla difficoltà della *blockchain*, che viene regolata automaticamente per mantenere il tempo medio di creazione di un nuovo blocco costante. Questo processo è descritto visivamente dalla figura 2.2.

Il primo miner che trova correttamente l'hash del blocco viene ricompensato in criptovaluta per il lavoro svolto. Questo meccanismo garantisce sicurezza nella rete, perché rende difficile ad un attaccante malintenzionato compromettere la sicurezza della rete non avendo la stessa potenza di calcolo.

Questa è attualmente utilizzata in *Bitcoin*, in cui il tempo medio della formazione di blocchi è di 10 minuti circa, oppure la piattaforma *Ethereum*, altro importante esempio di blockchain. Si consideri che ne esistono varie tipologie, in cui ciascuna cerca di ridurre il consumo energetico dando prova del lavoro svolto (*Meaningful*) oppure assicurandosi che i nodi abbiano calcolato correttamente dopo un certo tempo (*Delayed*). Il principale svantaggio del PoW è che richiede un elevato consumo energetico, che può essere risolto con l'utilizzo di algoritmi di consenso alternativi.

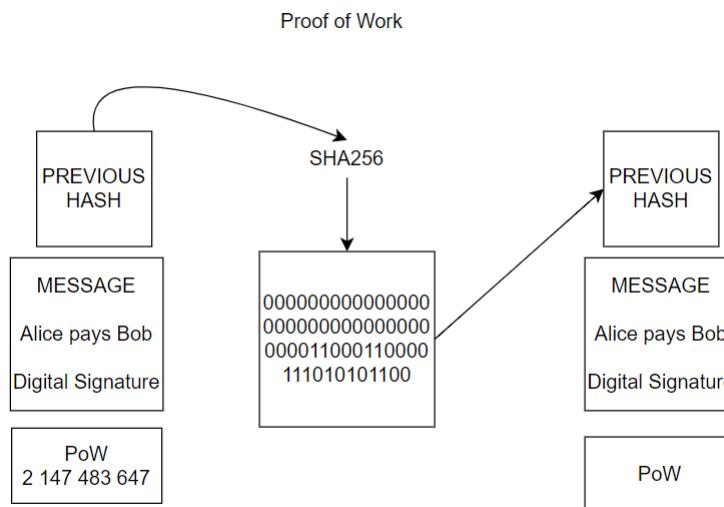


Figura 2.2: Processo di conferma in Proof of Work

2. *Proof of Stake (PoS)*, che si basa sulla detenzione di una certa quantità di criptovaluta come garanzia per la validazione delle transazioni. L'alto consumo di energia dell'algoritmo precedente ha portato ad algoritmi come il *Proof of Stake*, i nodi della rete bloccano una certa quantità di criptovaluta come 'punteggio' per dimostrare che hanno un interesse nella corretta validazione delle transazioni. Questo punteggio viene utilizzato come base per la selezione del nodo che convalida la transazione successiva.

A differenza del *PoW*, non ci sono *miner* coinvolti nel processo. Al loro posto, i partecipanti alla rete che vogliono essere coinvolti nella verifica della validità delle transazioni e nella creazione di blocchi nella rete devono detenere una certa quota nella rete, per esempio mettendo una certa quantità di moneta della rete in un portafoglio collegato alla sua *blockchain*. Questo processo è noto come 'placing a stake' o 'staking', che può essere tradotto come il fatto di mettere i propri interessi in gioco.

Pertanto, le reti PoS sono basate su algoritmi deterministici, il che significa che i validatori dei blocchi sono eletti a seconda della natura della posta in gioco. Pur risultando meno intensiva in termini di energia rispetto alla *PoW*, il sistema risulta essere sbilanciato a favore dei produttori di blocchi oppure a favore degli utenti ricchi, poiché i validatori con più moneta hanno più possibilità di essere eletti. Alcune varianti di questo algoritmo prevedono infatti di scegliere dei nodi delegati per convalidare le transazioni e avere più controllo (*Delegated*), oppure di scegliere i validatori in base alla quantità di moneta che hanno investito (*Proof of Burn*).

2.1.7 Tipi

Una prima categorizzazione delle *blockchain* avviene distinguendole in base ai permessi. Nello specifico:

- *permissioned*, in cui i nodi della rete sono controllati da un ente centrale, che può essere un'azienda, un'organizzazione o un'istituzione. Esse limitano l'accesso alla rete a determinati nodi e possono anche limitare i diritti di tali nodi su tale rete. Le identità degli utenti di una blockchain autorizzata sono note agli altri utenti della blockchain autorizzata. Queste sono spesso utilizzate da aziende per la gestione di dati sensibili o convalida di transazioni;
- *permissionless*, che non richiedono alcun permesso per partecipare e permettono agli utenti di essere pseudoanonimi (usando uno pseudonimo non si rivela alcun dettaglio relativo all'identità personale) e non restringono i permessi dei nodi. Queste ultime tendono ad essere più sicure delle precedenti, avendo vari nodi che convalidano le singole transazioni. Di fatto, sono usate dalle varie *blockchain*, come ad esempio quella di Bitcoin.

Fatta tale premessa, possiamo distinguere:

- *public*, in cui chiunque abbia accesso alla rete può partecipare e diventare un nodo autorizzato nella *blockchain*. Di fatto sono sicure, ma non anonime e molto meno scalabili. Una loro futura implementazione sarà all'interno dei sistemi di votazioni elettroniche.
- *private*, in cui i singoli nodi sono controllati da un ente centrale, che opera in una rete chiusa e, come tali, restringono la natura di blockchain, dato che richiede controllo centralizzato dei singoli nodi per poter funzionare. Queste sono utilizzate da parte di alcune banche per la gestione dei loro sistemi di pagamento.
- *hybrid*, in cui alcuni nodi sono pubblici e altri sono privati, normalmente utilizzata per eseguire con valide delle transazioni presenti al suo interno. Microsoft utilizza una blockchain di identità digitale basata sul meccanismo ibrido.
- *consortium*, all'interno della quale i nodi sono controllati da un ente centrale, ma la rete è aperta a tutti i membri del consorzio, portando anche qui ad una privatizzazione e centralizzazione della rete. IBM sta utilizzando una piattaforma basata sulla gestione dei pagamenti per banche ed aziende partner.

2.2 Blockchain: concetti avanzati

2.2.1 Token

Possiamo definire i *token* come unità di valore accettate da una comunità e costruite su una *blockchain* preesistente (dunque, non possono essere minati). Possono essere usati per rappresentare asset fisici come l'oro o l'immobiliare, oppure per rappresentare beni digitali come i dati o i diritti di accesso a servizi. Essendo registrati su una *blockchain*, sono immutabili e possono essere trasferiti in modo sicuro e trasparente da un proprietario all'altro.

Per utilizzare i token, gli utenti devono prima avere un portafoglio digitale, ovvero un'interfaccia che consente loro di accedere alla *blockchain* e interagire con essa. Una volta che un utente ha un portafoglio, può ricevere e inviare token. Per ricevere i token, l'utente deve fornire il proprio indirizzo del portafoglio al mittente, che poi invia i *token* all'indirizzo fornito. Per inviare i token, l'utente deve avere abbastanza *token* nel proprio portafoglio e deve conoscere l'indirizzo del destinatario a cui inviare i token.

Ogni bene viene considerato *asset*, in quanto non riproducibile e non falsificabile, esistente solo in forma digitale e quindi non fisica. Un *token* può essere lanciato sul mercato tramite un *initial coin offering* (ICO), che è un'offerta pubblica iniziale di *token*, al fine di finanziare nuovi progetti e comprenderne il nuovo valore. Come per le *blockchain*, gli asset non sono regolamentati da un organo centrale, pertanto è possibile raccogliere fondi senza nuovi intermediari e così creare nuovi progetti, adeguatamente documentati tramite un *white paper*.

che descrive il progetto e il suo funzionamento. Il loro processo di creazione dei *token* nelle *blockchain* viene definito come *minting*, in cui un creatore mette a disposizione un certo numero di token, che possono essere acquistati dai partecipanti.

Principalmente, i *token* possono essere classificati in base al loro scopo:

- *Utility token*, che consentono di acquistare un determinato bene o servizio e conferisce al titolare un diritto di opzione per l'acquisto o somministrazione di cose o per la fornitura di servizi (attuali o futuri).
- *Security token*, che sono *token* che rappresentano un diritto di proprietà su un bene fisico o digitale.
- *Payment token*, più comunemente noti come ‘crypto’. Come si può intuire, questi *token* sono utilizzati per acquistare e vendere beni e pagare le commissioni delle transazioni basate sulla blockchain senza la necessità di un intermediario.

I token, principalmente, sono caratterizzati dal fatto di essere spendibili e scambiabili con altri beni dal valore equivalente, ossia ‘fungibili’. In questo senso, è possibile citare:

- *token fungibili (fungible tokens)*, che hanno la caratteristica di essere non uniche, divisibili e avere un chiaro valore di mercato, quindi scambiabili sempre con altri beni dello stesso valore;
- *token non fungibili (non fungible token/NFT)*, beni fisici o digitali unici e irripetibili. Questi vengono definiti come tali in quanto non possono essere scambiati con altri beni dello stesso valore, e per natura delle stesse *blockchain*, sono considerati immutabili. Infatti, l'utente, disponendo di un *wallet*, può creare in qualsiasi momento un bene considerabile come unico, pagando una commissione al momento dell'acquisto del bene. Questo particolare tipo di *token* è molto utilizzato per la vendita di beni come opere d'arte, musica, videogiochi, per loro natura non contraffabili dato lo scambio tramite blockchain.

La loro creazione è vincolata da alcuni standard, stabilità della comunità della *blockchain Ethereum*, che ne vincola la creazione tramite *smart contract*, che sono dei contratti che vengono eseguiti sulla *blockchain* e che possono essere scritti in vari linguaggi di programmazione.

Di seguito i principali standard di riferimento che stabiliscono precisamente i parametri della loro creazione:

- *ERC-20*, che è il più diffuso e utilizzato standard per i token, che consente di creare *token* che possono essere trasferiti tra gli utenti. Questo stabilisce che un contratto esponga un saldo, un'opzione di trasferimento, un'opzione di approvazione e un evento di trasferimento;
- *ERC-721*, che è uno standard per i *token* non divisibili (NFT), che rappresentano un bene unico non scambiabile in altri modi. Di base utilizza gli stessi parametri del precedente, ma possiede un campo specifico che indica chi è il proprietario;
- *ERC-1155*, che è uno standard per i *token* divisibili e non divisibili e riunisce a livello di dati un supporto comune con funzione di trasferimento e di approvazione anche in blocco.

2.2.2 Tokenizzazione

La tokenizzazione è il processo di rappresentazione di un bene o di un'attività in forma digitale attraverso l'emissione di un *token* su una *blockchain*. In altre parole, si tratta di convertire un bene fisico o immateriale in un asset digitale che può essere negoziato e scambiato in modo decentralizzato. La combinazione con la *blockchain* potrebbe aprire nuove prospettive per l'ottimizzazione dei processi aziendali, che includono più partner, e l'introduzione di nuovi modelli di *business*. Poiché la tokenizzazione è un processo decentralizzato, non è necessario un intermediario per la creazione e la gestione di un *token* e questo facilita la creazione di nuovi beni, sapendo che la verifica viene effettuata da tutti i partecipanti della rete ed ogni bene è considerato unico.

Esistono diversi tipi di asset da considerare, i quali saranno poi successivamente convertiti in token:

- *Asset finanziari*, che rappresentano un diritto di proprietà su un bene fisico o digitale. Il concetto della finanza decentralizzata sta emergendo come *DeFi (Decentralized Finance)*, in cui i beni vengono spostati

su *blockchain* a seconda della loro natura. Infatti, come per i *token* i beni acquistano una fungibilità e possono essere scambiati tra utenti;

- *Asset immobiliari*, che rappresentano un bene fisico, come un immobile, che può essere tokenizzato e quindi diventare un bene digitale. Questi beni sono fungibili e come tali hanno un valore tendenzialmente fisso, che può essere soggetto a variazioni nel tempo;
- *beni intangibili (intangibles)*, come diritti di proprietà intellettuale o i citati beni collezionabili, e quindi acquisire grazie alla natura di *blockchain* un valore unico e verificato.

Si può quindi comprendere che la tokenizzazione può potenzialmente trasformare tutto in un bene con un valore, permettendo di avere dei prezzi equi stabiliti dalle vere esigenze del mercato, riducendo i costi di gestione essendo un sistema scalabile e sicuro che lo gestisce, permettendo l'aumento della liquidità, avendo sempre degli investitori pronti a scambiare i token, in modo trasparente e senza intermediari, riducendo notevolmente i tempi di scambio rispetto ai beni tradizionali.

2.2.3 Smart contract

Gli *smart contract* (o contratti intelligenti) sono programmi che automatizzano le azioni richieste in un accordo o contratto, considerate tracciate e irreversibili. Essi sono programmi informatici auto-eseguibili che vengono eseguiti su una *blockchain*, che automatizzano ed eseguono le clausole contrattuali in modo sicuro, trasparente e immutabile, senza la necessità di intermediari. Il loro funzionamento è regolato dal libro mastro centrale di *blockchain*, in cui i partecipanti devono firmare crittograficamente la loro partecipazione, fornendo precisamente i loro indirizzi di riferimento e codificando lo scambio delle informazioni secondo gli standard definiti nella sezione precedente.

Gli *smart contract* sono stati sviluppati per risolvere i problemi di affidabilità e sicurezza dei contratti tradizionali, che sono soggetti a errori umani, mancanza di trasparenza e vulnerabilità. Il loro funzionamento segue generalmente questo schema:

1. un utente avvia una transazione dal proprio portafoglio *blockchain*;
2. la transazione arriva al database distribuito, dove viene confermata l'identità del portafoglio dell'utente;
3. la transazione, che può essere un trasferimento di fondi e comprende il codice che definisce il tipo di transazione da eseguire, viene approvata;
4. questa viene eseguita come blocco all'interno della *blockchain*.

Nella figura 2.3 è possibile vedere un esempio di funzionamento.

Principalmente, questi vengono utilizzati nella piattaforma ⁶[Ethereum](#), dove i contratti vengono scritti in linguaggio di programmazione *Solidity*, che è un linguaggio di programmazione orientato agli oggetti basato su *C++*, eseguiti sulla piattaforma *Ethereum Virtual Machine (EVM)*, che ne consente l'esecuzione in modo scalabile regolata dagli standard precedenti. Ogniqualvolta venga eseguita una transazione, si ha un costo pagato in *gas*, costo in termini di energia necessaria per eseguire la transazione, che viene calcolato in base al numero di operazioni eseguite e pone un limite al numero di transazioni che possono essere eseguite in un dato periodo di tempo.



Figura 2.3: Funzionamento di uno Smart Contract

Possiamo citare alcuni esempi di applicazione di *smart contract*:

- *smart legal contracts*, legalmente applicabili e richiedono alle parti di soddisfare i loro obblighi contrattuali. Per creare alcuni contratti legali intelligenti, le parti coinvolte lavorano sul codice del contratto intelligente (o lo fanno i loro sviluppatori di software) finché non si accordano sui termini e sulle condizioni dell'accordo;
- *decentralized autonomous organizations (DAO)*, che sono organizzazioni che funzionano in modo decentralizzato e autonomo, senza un leader centrale e *open-source*, ma regolate da un contratto intelligente. All'interno delle *blockchain* vengono organizzate delle votazioni, che possono essere eseguite da tutti i partecipanti, che possono votare per o contro una proposta, oppure garantire il corretto scambio dei beni digitali all'interno della piattaforma, prevenendo possibili vulnerabilità;
- *crowdfunding*, che è un metodo di finanziamento di progetti e aziende, in cui le persone possono contribuire con denaro o beni, in cambio di un premio o di un prodotto finale;
- *logical applicative contracts*, che sono contratti che possono essere utilizzati per eseguire operazioni logiche, come la verifica di un dato e l'esecuzione di un'altra operazione, oppure gestire un sistema di pagamento o beni di scambio, garantendo la tracciabilità e la trasparenza di ogni bene alla consegna.

I contratti rimangono aggiornati grazie all'uso di *oracoli*, sistemi che permettono di ottenere informazioni esterne alla *blockchain*, come il valore di un bene o il valore di un'azione, attraverso l'uso di ^g[Application Program Interface](#) o di altri sistemi di comunicazione. In questo modo, il contratto mantiene la sua funzionalità interagendo in maniera ibrida con il mondo esterno; questi vengono definiti come *hybrid smart contracts*, combinando un'infrastruttura *on-chain* con un'infrastruttura *off-chain*. Il loro utilizzo, per quanto utile, deve essere attentamente bilanciato, sia in termini di veridicità dei dati che di sicurezza, in quanto possono essere soggetti a compromissione o di scalabilità, per cui è necessario un'attenta valutazione dei rischi.

2.2.4 Vulnerabilità e sicurezza

Le *blockchain* sono soggette a vulnerabilità, che possono essere sfruttate per ottenere informazioni o per compromettere il sistema. Il linguaggio *Solidity*, infatti, è un linguaggio che richiede particolare attenzione nella scrittura del codice, in quanto è possibile sfruttare alcune vulnerabilità date dal mancato controllo delle operazioni fatte, ciascuna delle quali comporta una transazione e scambio di criptovaluta, pubblicamente visibile e come tale tracciabile ed attaccabile in ogni momento. Ogni dato deve essere opportunamente criptato e protetto, in quanto è possibile ottenere informazioni sensibili come *password* oppure *chiavi private* di un portafoglio, che permettono di accedere ai fondi e di eseguire transazioni.

Di seguito verrà esaminato un insieme di problemi comuni che si possono verificare all'interno di un contratto intelligente, dovuti principalmente al mancato controllo e convalida dei dati in ingresso e in uscita, portando allo sfruttamento di vulnerabilità nell'ordine in cui vengono eseguite le operazioni.

- **re-entrancy:** è un problema importante che può verificarsi ogni volta si chiama una funzione di un contratto esterno; se il contratto esterno è malevolo, può essere sfruttato per eseguire delle chiamate ricorsive. Ogni volta che il contratto esterno ‘rientrerà’ nella funzione, riceverà la stessa quantità di fondi. Al termine dell'esecuzione, l'importo inviato sarà probabilmente più volte superiore a quello previsto. Questo problema può essere risolto utilizzando la funzione *transfer* o *send* per inviare fondi, in quanto queste funzioni eseguono un'operazione *throw* in caso di errore, interrompendo l'esecuzione del contratto e lanciando immediatamente un'eccezione che interrompe l'esecuzione del contratto. Un altro modo è di controllare l'ordine delle chiamate delle funzioni, introducendo l'uso dei cosiddetti *modifier*, che permettono di controllare l'ordine di esecuzione delle funzioni oppure attraverso l'uso delle istruzioni *assert* e *require*, che fermano l'esecuzione del codice qualora determinate condizioni non vengano rispettate (ad esempio, se il valore di un dato è negativo oppure per controllare gli indirizzi che eseguono le stesse transazioni);
- **integer overflow/underflow:** è un problema che si verifica quando si tenta di eseguire un'operazione aritmetica che produce un risultato troppo grande o troppo piccolo per essere rappresentato. Questo problema può essere sfruttato per ottenere un risultato diverso da quello atteso, ad esempio, se si tenta di sottrarre un numero più grande da uno più piccolo, il risultato sarà un numero molto grande, che potrebbe essere utilizzato per realizzare una transazione malevola ed ottenere un guadagno. Il problema può essere risolto sfruttando le ultime versioni del linguaggio *Solidity*;
- **Denial of Service (DoS):** è un attacco che mira a rendere un servizio non disponibile per gli utenti legittimi, attraverso l'invio di un numero elevato di richieste, che possono essere eseguite in maniera automatica. Il problema si verifica qualora esistano delle chiamate a contratti esterni non controllati oppure se si utilizzano cicli *for* o *while* con un numero di iterazioni elevato. Questo problema può essere risolto attraverso l'uso di *gas limit* e *gas price*, che permettono di limitare il numero di operazioni eseguibili e di fissare un prezzo limite per ogni operazione;
- **timestamp dependency:** è un problema che si verifica quando si utilizza il *timestamp* per eseguire operazioni, ad esempio, per controllare se un contratto è scaduto oppure per controllare se un utente ha eseguito un'operazione in un determinato intervallo di tempo. Questo problema può essere sfruttato per eseguire un'operazione in un momento diverso da quello previsto, ad esempio, se si utilizza il timestamp per controllare se un contratto è scaduto, un utente potrebbe eseguire un'operazione prima del tempo previsto. Questo problema può essere risolto utilizzando il *block number*, che permette di controllare il numero di blocchi eseguiti, oppure utilizzando il *block hash*, che permette di controllare l'hash del blocco corrente e verificare se è stato modificato;
- **transaction-ordering dependency (front running):** è un problema che si verifica quando un utente esegue un'operazione prima di un altro utente, ad esempio, se un utente esegue un'operazione di acquisto di un bene ad un prezzo più basso rispetto ad un altro utente, l'utente che esegue l'operazione per primo otterrà il bene ad un prezzo più basso. Questo problema può essere risolto utilizzando il *block number* oppure il *block hash* come visto in precedenza;

Altri accorgimenti che si possono adottare sono il controllo della visibilità fornita alle funzioni, che possono essere *public*, *private*, *internal* o *external*, intendendo rispettivamente che la funzione può essere chiamata da chiunque, solo all'interno del contratto, solo all'interno del contratto e dai contratti che ereditano il contratto oppure solo dai contratti che ereditano il contratto. Anche i valori di ritorno delle funzioni, se non controllati, possono scatenare dei problemi importanti di sicurezza, ad esempio, se una funzione restituisce un valore booleano, è possibile che un utente malevolo possa sfruttare questo valore per eseguire un'operazione non prevista, iniettando del codice indesiderato oppure eseguendo transazioni nulle, volte a consumare risorse della rete ed ottenere un guadagno esterno. I contratti sono, per loro natura, deterministici e le operazioni devono essere eseguite precisamente nello stesso ordine da tutti i nodi della rete, in modo da ottenere lo stesso risultato. Risulta quindi fondamentale considerare e controllare ciascuna di queste importanti problematiche, evitando che si generino situazioni in cui un nodo della rete può prendere il controllo di numerose operazioni, idealmente prendendo il controllo della maggioranza della rete (problema noto come *51% attack*) oppure prendendo il controllo di numerosi nodi creando molteplici identità fasulle (problema noto come *Sybil attack*).

2.2.5 Scalabilità

Le blockchain sono preziose per il fatto di essere un sistema deterministico e con un grado di affidabilità molto elevato, generalmente neutro e verificabile da parte dell'utente finale. In questo contesto, è fondamentale parlare del cosiddetto *blockchain trilemma*, che è un problema di scelta tra tre caratteristiche fondamentali di una blockchain: decentralizzazione, sicurezza e scalabilità. Infatti, decentralizzare significa mantenere un vasto numero di nodi sulla rete, senza però alcun controllo da parte di un ente centrale, mantenendo allo stesso modo un alto numero di transazioni in modo scalabile e una robustezza ai possibili partecipanti della rete.

In questo senso, si cerca di trovare un compromesso tra le tre caratteristiche, che è possibile ottenere laggiunta di ridondanza dei dati anche al di fuori della blockchain principale, cercando di bilanciare quanto possibile il consenso sui nodi distribuiti non intaccando il livello di libertà offerto ma anche aumentare il livello di dati trasmessi. Per questo, il nucleo scalabilità permette di introdurre numerose soluzioni al fine di migliorare le prestazioni della *blockchain*, dividendo però la *blockchain* in vari strati, definiti come *layer*.

- *Layer 0*, che è la *blockchain* principale, che contiene i dati e le transazioni;
- *Layer 1*, che è il livello responsabile della sicurezza e della scalabilità, che contiene i nodi che eseguono le transazioni e che sono responsabili della validazione;
- *Layer 2*, noto anche come livello di esecuzione, in cui si hanno i contratti intelligenti e le applicazioni che vengono eseguite sulla *blockchain*;
- *Layer 3*, noto anche come livello applicativo, che contiene le applicazioni decentralizzate (dApps), eseguite sulle *blockchain* e che interagiscono con gli utenti.

Comunemente, le soluzioni di scalabilità cercano di intervenire in vari modi:

- *Sharding*, che è una tecnica di scalabilità che consente di dividere la *blockchain* in più parti, chiamate *shards*, che possono essere gestite da nodi diversi e aumentando la dimensione dei singoli blocchi e la loro frequenza di creazione. Questa si effettua comunemente per il *Layer 1*.
- *Sidechains*, che sono *blockchain* secondarie che vengono utilizzate per eseguire transazioni parallele, che vengono poi sincronizzate con la *blockchain* principale. In questo ambito, soluzioni comuni includono i *canali di stato (state channels)*, che sono dei contratti intelligenti che permettono di eseguire transazioni tra due parti, senza che queste siano visibili sulla blockchain principale, e i *canali di pagamento (payment channels)*, che migliorano la comunicazione bidirezionale delle transazioni eseguendole sulla *blockchain* secondaria e dandone traccia sulla principale.

2.3 Self-Sovereign Identity

La ^o[Self Sovereign Identity](#) è un approccio all'identità digitale che dà agli individui il controllo sulle informazioni che usano per dimostrare chi sono a siti web, servizi e applicazioni in tutto il web. Senza l'*SSI*, gli individui con *account* (identità) persistenti su Internet devono affidarsi a una serie di fornitori terzi, come Facebook, Google e altri, che hanno il controllo delle informazioni associate alla loro identità.

Esistono molti modi per implementare l'*SSI* utilizzando le chiavi crittografiche e ne analizzeremo due.

- l'utilizzo di firme digitali, attraverso un processo di *firma digitale*, che permette di firmare un documento con una chiave privata, e di verificare la firma con la chiave pubblica;
- ^o[Decentralized Identifier](#), che è un identificatore univoco alfanumerico per un soggetto, che può essere utilizzato per identificare una persona, un'organizzazione, un dispositivo, un servizio, un documento, ecc.

Le parti coinvolte in questo processo sono principalmente tre:

1. *emittente*, detto anche *holder*, ossia l'entità che emette una credenziale, ad esempio un documento d'identità governativo;
2. *titolare*, detto anche *issuer*, ossia il proprietario della credenziale, cioè l'entità su cui l'emittente genera la credenziale;
3. *verificatore*, detto anche *verifier*, cioè l'entità che controlla la validità e l'autenticità della credenziale presentata dal titolare.

Il processo di riconoscimento è descritto dalla figura 2.4.

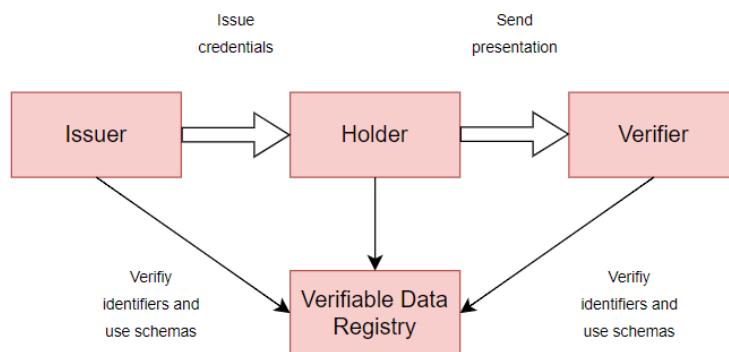


Figura 2.4: Processo di riconoscimento di una credenziale SSI

Lo scopo principale di questa tecnologia è consentire agli utenti un'esistenza indipendente da provider terzi, permettendo loro di controllare la propria identità in modo sicuro e accedendovi senza dover affidarsi a terzi. Inoltre, i sistemi e gli algoritmi che la supportano devono essere trasparenti, mantenendo le informazioni in modo trasparente e permanente, rendendo però semplice la portabilità e l'interoperabilità di queste all'interno delle varie piattaforme.

La tecnologia blockchain, per mezzo della sua natura decentralizzata, permette di creare un sistema di identità digitale che soddisfi questi requisiti. In particolare:

- il titolare della credenziale possiede il suo *Decentralized Identifier* firmato dalla propria coppia di chiavi che certifica la sua identità;
- l'emittente fornisce delle ^o[Verifiable Credentials](#), che certificano in modo digitale e crittograficamente protetto la validità del proprio ruolo;

- il verificatore controlla che, tramite ciascun blocco, sia stata rilasciata una VC valida e che il titolare sia il legittimo possessore di quella VC.

L'obiettivo principale è quello di fornire un utilizzo delle tecnologie *blockchain* tali da permettere agli utenti di selezionare quali credenziali mostrare (*selective disclosure o divulgazione selettiva*), secondo opportuni standard definiti gradualmente dall'organizzazione internazionale ^g[W3C](#), tra cui il citato *VC*.

2.3.1 Tipi e applicazioni

Il concetto di *SSI* è stato introdotto nel 2015 da *Sovrin Foundation*, che ha sviluppato il protocollo *Sovrin*, organizzazione privata senza scopo di lucro che ha creato la prima rete di identità auto-sovrana, diventato poi standard *W3C*. Esso si basa sul protocollo *Hyperledger Indy*, che è un ^g[framework open source](#) per la creazione di *SSI* basato su *blockchain* che fornisce strumenti, librerie e componenti riutilizzabili per fornire identità digitali legate al mondo *blockchain*, come ad esempio la firma digitale e la crittografia.

Questo è il principale, ma possiamo definire un insieme di standard utilizzati al fine di garantire agli utenti di possedere e controllare in modo indipendente la propria identità digitale, gestendo come vengono condivise le proprie informazioni. La particolarità di questi protocolli è di comunicare gli uni con gli altri attraverso messaggi crittografati:

- *DID (Decentralized Identifier)*, che consente di creare identificativi digitali decentralizzati come citato supportano e permettono di identificare i soggetti in modo resistente alla falsificazione, autenticando le informazioni in modo sicuro;
- *VC (Verifiable Credentials)*, che consente di gestire attestazioni verificati, quindi insiemi di informazioni che un individuo possiede o controlla, dimostrando che l'attestazione è stata rilasciata da un emittente autorizzato. Questo all'interno delle *blockchain* garantisce che ciascun nodo e parte in gioco sia chi dice di essere senza ledere la decentralizzazione alla base;
- ^g[Zero Knowledge Proof](#), che consente di dimostrare la conoscenza di una proprietà di un dato, senza rivelare alcun dettaglio relativo. Questa tecnologia è considerata a sé stante un'innovazione tecnologica, che permette attraverso l'individuazione di parti precise, la certezza che solo chi conosce l'informazione può correttamente rispondere alle domande poste senza rivelare i propri dati.

Attualmente, la tecnologia *SSI* prevede una graduale implementazione in sistemi di autenticazione e in generale di accesso ai servizi pubblici, permettendo all'utente un ulteriore controllo sui dati trasmessi rivelandoli solo a parti autorizzate. Questo risulta particolarmente utile nel settore sanitario, legislativo (garantendo certezza di identità e di voto) e finanziario, permettendo di ridurre i costi di gestione e di mantenimento dei dati, oltre che di ridurre i tempi di accesso ai servizi. Il principale organo promotore è la *Decentralized Identity Foundation (DIF)*, che ha come obiettivo quello di creare un ecosistema di identità digitali decentralizzate, promuovendo integrazioni, progetti e nuove idee in questo ambito.

2.4 Zero Knowledge Proof

Definita anche come prova a conoscenza zero, è una tecnologia che permette di dimostrare la conoscenza di una proprietà di un dato, senza rivelare alcun dettaglio relativo. Occorre definire le parti coinvolte:

- *Prover*, ossia l'entità che prova la conoscenza di alcune informazioni riservate. L'informazione segreta è il 'testimone' (*witness*) della prova e la presunta conoscenza del testimone da parte del prover stabilisce un insieme di domande a cui può rispondere solo chi conosce l'informazione.
- *Verifier*, ossia l'entità che verifica la correttezza della prova. Lui si occupa di scegliere a caso la sfida (*challenge*) e di verificare la risposta del prover. La risposta del prover permette al verificatore di controllare se il primo ha davvero accesso al testimone. La verifica segue visivamente dalla figura 2.5. Per assicurarsi

che il prover non stia indovinando alla cieca e non ottenga le risposte corrette per caso, il verificatore sceglie altre domande da porre.

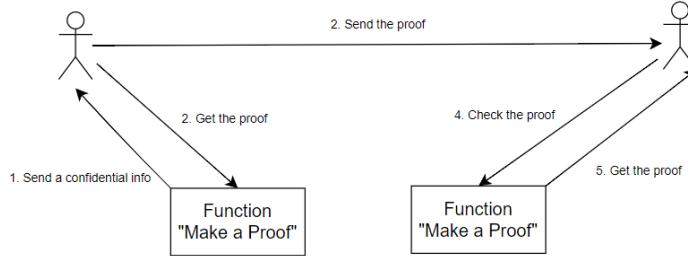


Figura 2.5: Processo di verifica di una ZKP

A livello di transazioni, la *ZKP* è utilizzata per garantire la sicurezza con certezza che le transazioni siano valide (*validity proofs*), tramite la presenza del testimone tipicamente basato su calcoli polinomiali o prossimità ad un insieme di valori, come ad esempio la distanza di Hamming. Le transazioni possono essere invalidate in ogni momento tramite le *fraud proofs*, che dimostrano che la transazione è stata effettuata in modo illegittimo. Queste ultime confrontano i *Merkle trees* delle transazioni e verificano che l'*hash* corrisponda a quella originale.

Di fatto, ogni prova deve dimostrare di essere completa, pertanto è necessario che il verificatore sia in grado di verificare che la risposta del *prover* sia corretta, dimostrando questa affermazione con solidità, in modo da non lasciare spazio a dubbi e a falsi positivi, senza trasmettere alcun dato (a conoscenza zero). L'idea principale della *ZKP* è presupporre di non fidarsi di nessuno (*zero trust*), al fine di garantire la sicurezza delle informazioni. In questo modo, ciascun utente viene monitorato a livello di privilegi, accessi e dati, senza che nessuno possa accedere senza dimostrare di averne il permesso.

2.4.1 Tipi e applicazioni

Esistono diverse implementazioni di *ZKP*, ognuna delle quali presenta un proprio compromesso in termini di dimensione della prova, tempo del *prover*, tempo di verifica e altro ancora. I principali tipi sono:

- *zk-SNARK*, acronimo di *Zero Knowledge Succinct Non-interactive Argument of Knowledge*, prova di dimensioni ridotte e facile da verificare. Essa è stata sviluppata dalle piattaforme *Zcash* e *Ethereum* utilizzando le curve ellittiche, che presuppongono che sia impossibile trovare il logaritmo discreto di un elemento casuale della curva ellittica a partire da un punto base pubblicamente noto. Il calcolo delle curve ellittiche è meno dispendioso dal punto di vista computazionale rispetto al calcolo delle funzioni di hashing.
- *zk-STARK*, acronimo di *Zero Knowledge Succinct Transparent Argument of Knowledge*, tipo di prova crittografica che richiede un'interazione minima o nulla tra il *prover* e il verificatore. I vantaggi principali delle STARK rispetto alle SNARK sono che hanno tempi di prover rapidi e sono più facili da scalare in quanto offrono una maggiore potenza di calcolo. Essa è stata sviluppata dalla piattaforma *Horizen* utilizzando le *curve ellittiche*, che presuppongono che sia impossibile trovare il logaritmo discreto di un elemento casuale della curva di *Weierstrass* a partire da un punto base pubblicamente noto. Il calcolo delle curve di *Weierstrass* è più dispendioso dal punto di vista computazionale rispetto al calcolo delle curve ellittiche.

All'interno delle blockchain, vengono utilizzati all'interno delle *sidechain* secondo il sistema delle *zk-Rollup*, che permettono di eseguire transazioni su una *blockchain* laterale (*sidechain*) senza doverle scrivere su questa. In altre parole, con uno *zk-rollup*, le transazioni vengono ‘confezionate’ in un'unica transazione che viene elaborata sulla *blockchain*, riducendo il carico di lavoro richiesto alla rete. Inoltre, grazie all'utilizzo delle *Zero Knowledge*

Proof, le transazioni vengono verificate in modo sicuro, senza rivelare alcuna informazione sulle transazioni stesse.

Principalmente, queste vengono utilizzate in sistemi di scalabilità *layer-2*, al fine di memorizzare solo il minimo numero previsto di transazioni (*rollup*), oppure memorizzando solamente un hash sulla catena per proteggere i dati del libro mastro (validium), oppure scegliere come salvarli per ogni transazione (*volition*). Alcuni casi d'uso possibili di applicazione di questa tecnologia possono essere:

- pagamenti anonimi
- protezione dell'identità
- autenticazione
- archiviazione di dati sensibili

L'utilizzo combinato con la *Self-Sovereign Identity* è particolarmente interessante perché consente di garantire la *privacy* e la sicurezza delle informazioni personali degli utenti. In particolare, è possibile garantire di conoscere le parti in gioco nella rete poiché verificate in modo sicuro e privato tramite credenziali certificate, scegliendo anzi quali informazioni rivelare e senza dover rivelare alcuno dei propri dati per dimostrare di possederli.

Capitolo 3

Descrizione dello stage

In questa sezione viene presentata un'analisi del percorso di stage, una descrizione dell'idea e delle tecnologie, individuando possibili rischi e problematiche che potrebbero presentarsi durante lo svolgimento dello stesso. Inoltre, viene fornita una panoramica degli obiettivi da raggiungere e della pianificazione delle ore di lavoro.

3.1 Introduzione al progetto e idea dello stage

Oggi più che mai è importante garantire la sicurezza dei propri dati e delle proprie informazioni, permettendo uno scambio sicuro e protetto di dati sensibili tra due o più entità. Inoltre, è necessario garantire che le informazioni siano autentiche e non modificate, per evitare che un utente malintenzionato possa alterare i dati e compromettere la sicurezza del sistema. Sulla base di questa premessa, in accordo con l'azienda, si è deciso di sviluppare un progetto che permetta di esplorare le possibili applicazioni della tecnologia ⁶blockchain all'interno di un contesto reale e possibilmente applicabile in ambito aziendale.

In particolare, la piattaforma oggetto dello sviluppo durante il tirocinio ha come obiettivo principale il controllo dell'accesso ai contenuti riservati, nel contesto dei film vietati ai minori. Grazie all'utilizzo della tecnologia ⁶blockchain e dello standard ⁶W3C ⁶Self Sovereign Identity, permettendo quindi di riconoscere i propri utenti evitando condivisione di informazioni, ma criptando i propri dati personali e verificando l'autenticità dei dati trasmessi. Il progetto si pone quindi di risolvere il problema della condivisione di contenuti riservati, tramite l'utilizzo di tecnologie innovative e la collaborazione con lo stagista Alessio De Biasi, utilizzando il codice di un progetto di un laureando magistrale presso Computer Science dell'Università Ca' Foscari richiamato come libreria e di altri stagisti di laurea triennale con progetti simili basati sull'applicazione di tecnologie immutabili e decentralizzate.

Il suo utilizzo è molto simile ad altre piattaforme basate sulla condivisione e prenotazione di contenuti multi-mediali legati ai film: l'utente si registra presso la piattaforma, fornendo i propri dati personali e la propria età. In questa fase, si ha il riconoscimento dell'utente tramite l'utilizzo di un ⁶Decentralized Identifier e la creazione di un ⁶Verifiable Credentials contenente i dati personali dell'utente, che verrà poi utilizzato per l'autenticazione e la verifica dei dati. Una volta effettuata la registrazione, l'utente può accedere alla piattaforma e visualizzare i contenuti disponibili, permettendo l'accesso anche a contenuti limitati qualora l'utente sia maggiorenne. Riconosciuta l'età dell'utente e verificata dalla piattaforma *blockchain* la sua autenticità, l'utente può visualizzare il contenuto selezionato dimostrando tramite ⁶Zero Knowledge Proof di essere maggiorenne, senza dover fornire ulteriori informazioni personali, completando la prenotazione.

Nello specifico, il progetto è sviluppato da autodidatta, chiedendo consigli e supporto allo stagista De Biasi per confronti e chiarimenti sull'implementazione degli standard in oggetto e per l'utilizzo del codice dello *smart contract* da lui fornito, usato come libreria per l'implementazione della piattaforma in oggetto. Inoltre, consigli e supporto sono stati forniti anche in modo reciproco con lo stagista Raffaele Bussolotto, studente presso Inge-

gneria Informatica di Padova, con un percorso simile al mio a livello di argomenti, basato sempre sulle tecnologie in oggetto ma con diverso scopo. Il loro supporto è stato puramente di confronto, in quanto l'implementazione dei rispettivi progetti è stata fatta in modo indipendente e senza collaborazione diretta.

Sulla base di queste premesse nasce il progetto **VerifiedMovies**.

3.2 Analisi preventiva dei rischi

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. Inesperienza tecnologica e metodologica

Descrizione: il progetto prevede l'utilizzo di tecnologie e metodologie di cui non si ha piena esperienza e conoscenza, rendendo più difficoltosa la comprensione e l'applicazione delle stesse in fase di implementazione.

Soluzione: è stato previsto un periodo di formazione iniziale per studiare le tecnologie e le metodologie da utilizzare e l'aiuto/supporto del tutor aziendale e di altri stagisti nella risoluzione di problemi e nella discussione di possibili soluzioni.

2. Difficoltà di integrazione con lo *smart contract* da usare come libreria

Descrizione: il sistema potrebbe incontrare difficoltà nell'integrazione con lo *smart contract* da richiamare per la gestione delle identità digitali, a causa di errori di programmazione o di problemi di comunicazione tra le parti coinvolte.

Soluzione: è stato previsto un periodo di test e debug per verificare il corretto funzionamento del sistema e per risolvere eventuali problemi riscontrati, approfondendo il dialogo con il tutor aziendale e con lo stagista che ha sviluppato lo *smart contract* che dovrà essere richiamato in fase di implementazione.

3. Ritardi nello sviluppo

Descrizione: potrebbero verificarsi ritardi nello sviluppo del sistema a causa di problemi tecnici o imprevisti, come la mancanza di risorse necessarie o la complessità del progetto, rimodulando adeguatamente attività e tempistiche.

Soluzione: è stato previsto un periodo di test e debug per verificare il corretto funzionamento del sistema e per risolvere eventuali problemi riscontrati, intensificando il dialogo interno con il proponente e gli stagisti per discutere di possibili soluzioni e rimodulando le attività e le tempistiche in base alle necessità.

4. Problemi di sicurezza

Descrizione: il sistema potrebbe riscontrare problemi di sicurezza e vulnerabilità a attacchi informatici o la mancanza di protezione dei dati sensibili.

Soluzione: fin dall'inizio dell'attività di sviluppo, si cerca di garantire un'implementazione al passo con le *best practices* previste dai linguaggi di programmazione utilizzate e lato web, come l'utilizzo di librerie e framework aggiornati e sicuri, la validazione dei dati in input e la protezione da attacchi di tipo *SQL injection* e *XSS*. Inoltre, è possibile prevedere un'attività di testing approfondito, ad esempio utilizzando strumenti di analisi statica del codice e di penetration testing, per verificare la corretta implementazione delle misure di sicurezza e la presenza di eventuali vulnerabilità.

5. Cambiamenti dei requisiti durante lo sviluppo

Descrizione: i requisiti del sistema potrebbero cambiare durante l'attività di implementazione, ad esempio a causa di una modifica delle esigenze del progetto o di un errore di analisi iniziale.

Soluzione: è possibile prevedere un'attività di pianificazione flessibile, ad esempio utilizzando metodologie agili, come ^gScrum, che prevedono una pianificazione adattiva ai cambiamenti dei requisiti. Inoltre, potrebbe essere utile prevedere una comunicazione costante con il tutor aziendale e gli altri stagisti in fase di validazione dei requisiti aggiornati, procedendo con lo sviluppo in modo coerente ed organizzato.

3.3 Obiettivi e requisiti

Il tirocinio prevede lo svolgimento dei seguenti obiettivi, riportando questa notazione, come dal documento *Piano di Lavoro*:

- O per i requisiti obbligatori, vincolanti in quanto obiettivo primario richiesto dal committente;
- D per i requisiti desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
- F per i requisiti facoltativi, rappresentanti valore aggiunto non strettamente competitivo.

Le sigle precedentemente indicate saranno seguite da una coppia sequenziale di numeri, identificativo del requisito.

- Obbligatori:
 - *O01*: descrivere i concetti di base della ^gblockchain, tra cui la sua architettura, i nodi della rete, la crittografia e il consenso distribuito;
 - *O02*: analizzare il concetto di ^gsmart contract e il linguaggio ^gSolidity, con particolare attenzione alle vulnerabilità principali e alle tecniche per evitare errori di programmazione;
 - *O03*: approfondire il funzionamento della firma asimmetrica delle transazioni su catena e la validazione dei blocchi, studiando le tipologie di consenso e le catene più conosciute;
 - *O04*: studiare le tecniche di crittografia utilizzate per garantire la sicurezza e la *privacy* delle informazioni personali nell'ambito della ^gSelf Sovereign Identity e dei protocolli per la gestione delle identità digitali;
 - *O05*: individuare casi d'uso reali per la *Self Sovereign Identity*, analizzando i consorzi internazionali coinvolti in ambito di ricerca e i finanziamenti europei;
 - *O06*: discutere le sfide e i problemi legati alla *Self Sovereign Identity*, studiando un possibile scenario futuro di applicabilità e basato su ^gZero Knowledge Proof.
- Desiderabili:
 - *D01*: implementare una ^gDecentralized Application (o dApp) tramite le librerie ^gEthers.js oppure ^gWeb3.js, utilizzando uno *smart contract* di esempio;
 - *D02*: realizzare una UI (interfaccia utente) per la dApp utilizzando *HTML*, *CSS* e *JavaScript*;
 - *D03*: utilizzare la *Self Sovereign Identity* e i protocolli studiati per implementare funzionalità di autenticazione utente nella *dApp*;
 - *D04*: testare e debuggare l'applicazione implementata;
 - *D05*: discutere problemi e sfide relativi all'implementazione di applicazioni decentralizzate su *blockchain* ^gEthereum, con particolare attenzione alla scalabilità e alla sicurezza.
- Facoltativi:
 - *F01*: approfondire l'utilizzo di altri linguaggi di programmazione per gli *smart contract*, come Vyper;
 - *F02*: analizzare l'utilizzo di altre tecnologie *blockchain*, come *Polkadot* o *Cardano*, per implementare la *Self Sovereign Identity*;

- F03: esplorare altre funzionalità delle librerie *Ethers.js* e/o *Web3.js*, come l’invio di transazioni;
- F04: investigare i limiti e le sfide legate all’utilizzo della tecnologia *blockchain*.

Capitolo 4

Analisi dei requisiti

In questa sezione vengono analizzati i requisiti del progetto e ne viene data un'analisi ad alto livello, combinando una visione concettuale con una visione pratica ed implementativa. Vengono inoltre descritti i casi d'uso e i requisiti individuati, con l'obiettivo di fornire una visione generale del sistema e delle sue funzionalità, in modo semplice e comprensibile.

4.1 Descrizione ed analisi del sistema

L'obiettivo del progetto è la creazione di una piattaforma per la visualizzazione di contenuti multimediali, in questo caso dei film, tramite l'utilizzo degli standard [«W3C Decentralized Identifier»](#) e [«Self Sovereign Identity»](#) per la verifica sicura dell'identità dell'utente e per la visualizzazione di tutti i contenuti della piattaforma, certificando in modo sicuro la propria identità senza diffondere dati personali tramite l'utilizzo di un [«Zero Knowledge Proof»](#).

Più precisamente, l'implementazione avviene usando lo [«smart contract»](#) che implementa ad alto livello lo standard [«Self Sovereign Identity»](#), scritto nel linguaggio di programmazione [«Solidity»](#) da parte dello studente magistrale Alessio De Biasi. Questo contratto permette la gestione dell'identità digitale tramite la creazione di un documento di identità digitale, che contiene un identificatore univoco chiamato [«Decentralized Identifier»](#), composto da una stringa alfanumerica che identifica l'utente, e un *DID Document*, che contiene le informazioni dell'utente a lui associato. Normalmente, la sua creazione avviene tramite un file [«JSON»](#), composto dallo standard di riferimento, l'identificativo ed un campo *controller*, che ha la capacità di fare dei cambiamenti al documento stesso e certifica la firma digitale del documento, sulla base del metodo usato per la sua creazione. Il formato del *DID Document* è come il seguente (codice proveniente da [5]):

```
1  {
2      "@context": [
3          "https://www.w3.org/ns/did/v1",
4          "https://w3id.org/security/suites/ed25519-2020/v1"
5      ]
6      "id": "did:example:123456789abcdefghi",
7      "authentication": [
8          {
9              "id": "did:example:123456789abcdefghi#keys-1",
10             "type": "Ed25519VerificationKey2020",
11             "controller": "did:example:123456789abcdefghi",
12             "publicKeyMultibase": "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
13         }
14     ]
15 }
```

Nel codice che dovrà essere utilizzato come libreria, ogni documento è caratterizzato da una *capability delegation*, che permette di stabilire un'entità autorizzata come *issuer* del dato trasmesso ed un metodo di autenticazione, in cui viene stabilito il modo in cui implementare il meccanismo di riconoscimento da parte dell'utente (dimostrando di fatto che sia chi dice di essere) e un genitore, composto da un identificativo ed una firma digitale.

L'idea del codice è di creare una cosiddetta ‘catena di fiducia’, la cui idea è associare ad un utente un *Decentralized Identifier (DID)* e dimostrare, attraverso un meccanismo di autenticazione definito a priori nel documento tramite firme digitali, la risoluzione dei dati trasmessi, il riconoscimento univoco della sua identità e l'accesso ad un determinato contenuto. Infatti, l'utente viene riconosciuto perché i suoi dati sono stati certificati da un ente con capacità di firma digitale che, a sua volta, si fida di altri enti anch'essi riconosciuti, fino ad arrivare ad un soggetto radice. Se questa catena di fiducia viene risolta correttamente, l'utente viene riconosciuto univocamente e può accedere ai contenuti della piattaforma.

La struttura dati *blockchain*, attraverso l'utilizzo di un *account* associato ad un *Decentralized Identifier*, permette di certificare l'identità dell'utente tramite la firma digitale dei dati trasmessi con la propria coppia di chiavi pubblica/privata, certificando in modo immutabile la propria identità e permette di realizzare un sistema di autenticazione sicuro e decentralizzato, che non richiede l'utilizzo di un ente terzo per la verifica delle informazioni.

L'utilizzo della piattaforma e delle sue parti prescinde dalla dimostrazione, attraverso una chiamata al contratto libreria fornito, della risoluzione attraverso firma digitale del *Decentralized Identifier (DID)* associato all'utente, che dimostra l'utilizzo di un *account* associato alla rete *blockchain*, la sua identità. Se l'indirizzo che ha firmato digitalmente il documento è associato ad un profilo creatore di quel *Decentralized Identifier (DID)* così firmato, allora l'utente può accedere ai contenuti della piattaforma. L'implementazione di questo passaggio prevede che l'utente dimostri precisamente la propria identità secondo un meccanismo *challenge-response*: viene generato un numero casuale e viene chiesto all'utente di firmarlo digitalmente, in modo da dimostrare che è in possesso della chiave privata associata all'indirizzo che ha firmato il documento.

Se il metodo di autenticazione conteneva la sua firma e come dato il numero casuale trasmesso, grazie alla generazione dell'identificatore e la chiamata al contratto in grado di risalire precisamente all'*account blockchain* che l'ha emessa, si certifica l'utente in modo univoco. Tale meccanismo viene implementato in fase di registrazione, in cui l'utente possiede un proprio ^g*Decentralized Identifier* e registra i propri dati anagrafici e in fase di accesso, quando l'utente inserita esclusivamente come unico dato per effettuare il *login* il proprio identificatore, firmato digitalmente nella fase precedente e appartenente in modo univoco a lui.

In questa sezione, l'utente viene presentato ad un insieme di film e può scegliere di prenotarne uno. Ciascun film possiede una sua valutazione; per potervi accedere e prenotarlo, l'utente dovrà ogni volta certificare la propria età; questo passaggio avviene attraverso la creazione di una ^g*Verifiable Presentation*, contenente molteplici ^g*Verifiable Credentials*, con una associata all'utente (definito *holder* delle credenziali) e le altre associate agli enti verificatori (definiti *issuer*, che sono gli enti che hanno rilasciato le credenziali). Questo viene realizzato tramite l'utilizzo del ^g*Decentralized Identifier* utilizzato in fase di registrazione assieme ad altri dati personali di massima (*username*, *email*, data di nascita) e successivo *login* (utilizzando il solo ^g*Decentralized Identifier*), verificando che i suoi dati siano corretti ed innescando il meccanismo di verifica correttamente.

Normalmente, una *Verifiable Presentation* è infatti composta da:

- un insieme di metadati, che descrivono la presentazione;
- un insieme di *Verifiable Credentials*, che sono le credenziali associate alla presentazione;
- un insieme di *proof*, che sono le prove di autenticità delle credenziali.

Invece, una *Verifiable Credential* è composta da:

- un contesto di riferimento, che è un [URI](#) che definisce il contesto di riferimento delle credenziali;
- un identificativo, che specifica il sito di riferimento;
- un tipo, che specifica se la credenziale è appropriata per la presentazione;
- un soggetto, che è l'identificativo dell'utente a cui sono associate le credenziali;
- un *issuer*, che è l'identificativo dell'ente che ha rilasciato le credenziali;
- un *issuanceDate*, che è la data di rilascio delle credenziali;
- una *proof*, che è la prova di autenticità delle credenziali.

Il seguente è un esempio di una *Verifiable Presentation* con al suo interno un insieme di *Verifiable Credentials* (codice proveniente da [20]):

```

1   {
2     "@context": [
3       "https://www.w3.org/2018/credentials/v1",
4       "https://www.w3.org/2018/credentials/examples/v1"
5     ],
6     "type": "VerifiablePresentation",
7     "verifiableCredential": [
8       {
9         "@context": [
10          "https://www.w3.org/2018/credentials/v1",
11          "https://www.w3.org/2018/credentials/examples/v1"
12        ],
13        "type": ["VerifiableCredential", "UniversityDegreeCredential"],
14        "credentialSchema": {
15          "id": "did:example:cdf:35LB7w9ueWbagPL94T9bMLtyXDj9pX5o",
16          "type": "did:example:schema:22KpkXgecryx9k7N6XN1QoN3gXwBkSU8SfyyYQG"
17        },
18        "issuer": "did:example:Wz4eUg7SetGfaUVCn8U9d62oDYrUJLuUtcy619",
19        "credentialSubject": {
20          "degreeType": "BachelorDegree",
21          "degreeSchool": "College of Engineering"
22        },
23        "proof": {
24          "type": "AnonCredDerivedCredentialv1",
25          "primaryProof": "cg7wLNSi48K5qNyAVMwdYqVHSMv1Ur8i...",
26          "nonRevocationProof": "mu6fg24MfJPU1HvSXsf3ybzMARib4WxG...",
27          "RSce53M6UwQCxYshCuS3d2h"
28        }
29      ],
30      "proof": {
31        "type": "AnonCredPresentationProofv1",
32        "proofValue": "DgYdYMUYHURJLD7xdnWRinqWCEY5u5fK...",
33        "j915Lt3hMzLHoPiPQ9sSVfRrs1D"
34      }
35    }
36  }
```

Al suo interno, come si vede, sono presenti un insieme di credenziali, contenente un campo di dimostrazione di appartenenza ad uno schema comune, a cui sarà associata la catena di fiducia, e degli insiemi di metodi di firma digitale, che dimostrano la validità delle credenziali presenti, avendo ciascuna un insieme di *issuer* riconosciuti. Ciò dipende dall'utilizzo del campo *proof*, che permette di stabilire precisamente l'appartenenza della credenziale ad uno stesso schema fidato, in quanto firmato digitalmente da un ente autorizzato, perché fa

parte della stessa prova di firma comune.

Il meccanismo è in grado di integrarsi con ^g[Zero Knowledge Proof](#), in cui la prova di appartenenza ad uno schema comune è dimostrata senza rivelare informazioni sensibili di alcun tipo, combinando un insieme di *Verifiable Credentials*, verificate tramite un insieme di *issuer* riconosciuti dal sistema della catena di fiducia grazie all'uso della proprietà *proof* descritta sopra e la definizione di uno schema comune a cui fare riferimento, che certifica il riconoscimento in quanto firmato a catena da un insieme di *issuer* (secondo le specifiche dettagliate in [19]).

Questo permette all'utente di definire precisamente quali informazioni vuole rivelare e, attraverso l'utilizzo del contratto librerie, dimostrare la propria identità, verificato con questo meccanismo e accedendo al proprio contenuto di interesse, concludendo la prenotazione presso il sito cinema senza rivelare informazioni sensibili.

4.2 Casi d'uso

In questa sezione, sono presentati i diagrammi dei casi d'uso, definiti secondo il linguaggio standard ^g[UML](#), che descrivono le interazioni tra gli attori, definiti convenzionalmente come utenti del sistema e il sistema stesso. Segue una tabella di tracciamento dei requisiti, che dettaglia come i requisiti individuati nella sezione precedente siano soddisfatti dai casi d'uso definiti in questa sezione, in base all'analisi prevista. Di seguito, un diagramma complessivo (in figura 4.1) dei casi d'uso individuati, descritti nel dettaglio successivamente:

UC1: Registrazione

Attori Principali: Utente non autenticato.

Precondizioni: L'utente ha avviato l'applicazione web e ha aperto la pagina di registrazione.

Descrizione: L'utente vuole registrarsi presso l'applicazione web.

Postcondizioni: L'utente ha completato la registrazione e può effettuare il *login*.

Scenario Principale: L'utente:

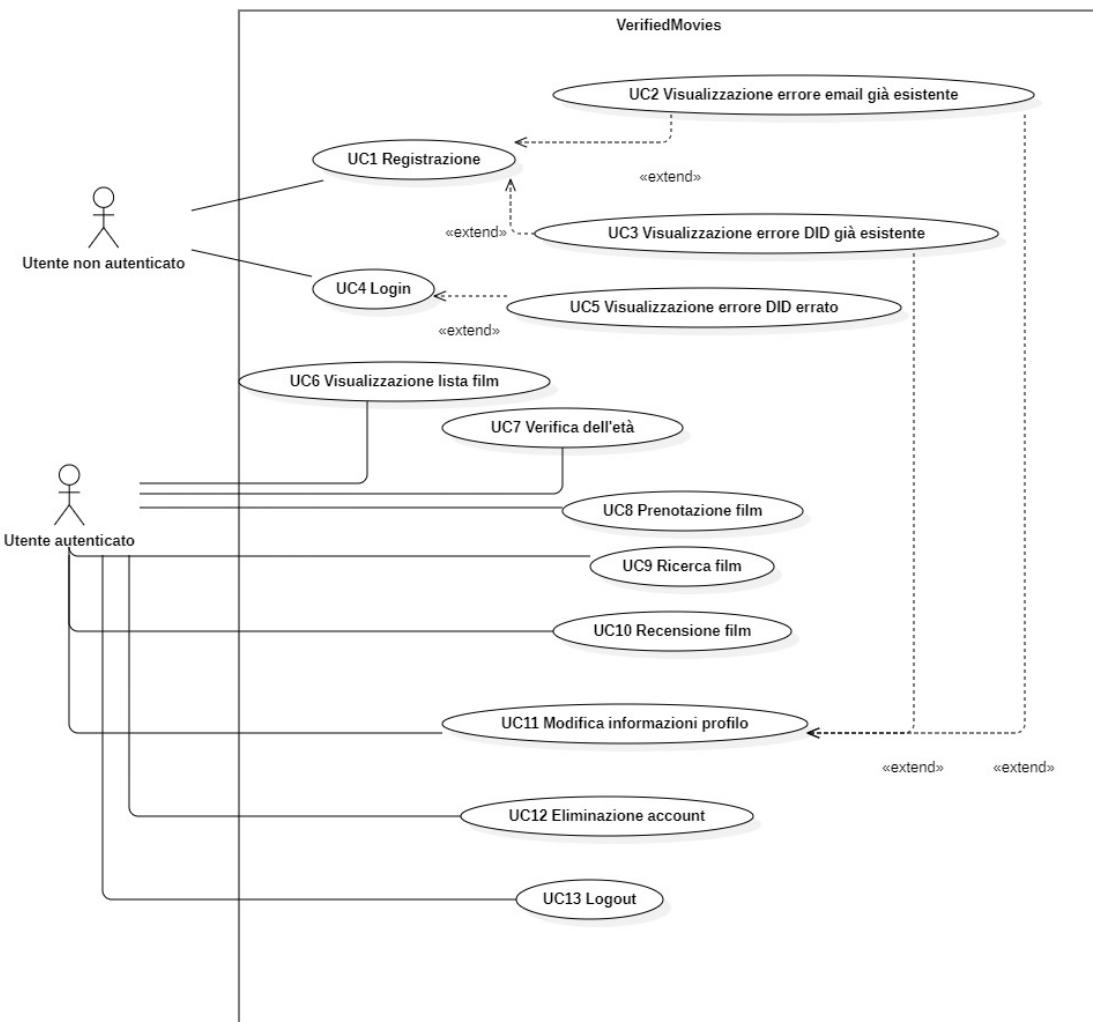
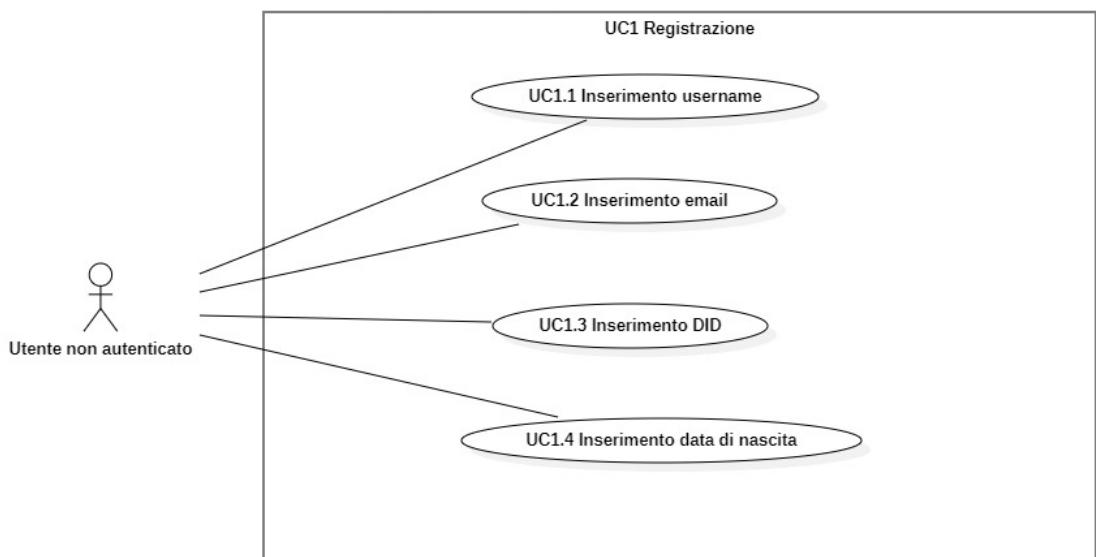
1. inserisce il proprio *username* (UC1.1);
2. inserisce la propria *email* (UC1.2);
3. inserisce il proprio *DID* (UC1.3);
4. inserisce la propria data di nascita (UC1.4).

Estensioni:

1. Visualizzazione errore *email* già esistente (UC2);
2. Visualizzazione errore *DID* già esistente (UC3).

Generalizzazioni: Come da figura 4.2:

1. Inserimento *username* (UC1.1);
2. Inserimento *email* (UC1.2);
3. Inserimento *DID* (UC1.3);
4. Inserimento data di nascita (UC1.4);

**Figura 4.1:** Scenario principale**Figura 4.2:** UC1: Registrazione

UC1.1: Inserimento *username*

Attori Principali: Utente non autenticato.

Precondizioni: L'utente ha avviato l'applicazione web e ha aperto la pagina di registrazione.

Descrizione: L'utente deve inserire uno *username* per registrarsi presso l'applicazione web.

Postcondizioni: L'utente ha inserito lo *username* e può procedere con la registrazione.

Scenario Principale:

1. L'utente inserisce il proprio *username*.

UC1.2: Inserimento *email*

Attori Principali: Utente non autenticato.

Precondizioni: L'utente ha avviato l'applicazione web e ha aperto la pagina di registrazione.

Descrizione: L'utente deve inserire una *email* per registrarsi presso l'applicazione web.

Postcondizioni: L'utente ha inserito la propria mail e può procedere con la registrazione.

Scenario Principale:

1. L'utente inserisce la propria *email*.

UC1.3: Inserimento *DID*

Attori Principali: Utente non autenticato.

Precondizioni: L'utente ha avviato l'applicazione web e ha aperto la pagina di registrazione.

Descrizione: L'utente deve inserire il proprio *DID* per registrarsi presso l'applicazione web.

Postcondizioni: L'utente ha inserito il proprio *DID* e può procedere con la registrazione.

Scenario Principale:

1. L'utente inserisce il proprio *DID*.

UC1.4: Inserimento data di nascita

Attori Principali: Utente non autenticato.

Precondizioni: L'utente ha avviato l'applicazione web, non è autenticato e ha aperto la pagina di registrazione.

Descrizione: L'utente deve inserire la propria data di nascita per registrarsi presso l'applicazione web.

Postcondizioni: L'utente ha inserito la propria data di nascita e può procedere con la registrazione.

Scenario Principale:

1. L'utente inserisce la propria data di nascita.

UC2: Visualizzazione errore *email* già esistente

Attori Principali: Utente non autenticato.

Precondizioni: L'utente ha inserito una *email* già esistente.

Descrizione: L'utente deve inserire una *email* diversa per registrarsi presso l'applicazione web.

Postcondizioni: L'utente ha inserito una *email* diversa e può procedere con la registrazione.

Scenario Principale:

1. L'utente visualizza un messaggio di errore che lo informa che la *email* inserita è già presente nel sistema.

UC3: Visualizzazione errore *DID* già esistente

Attori Principali: Utente non autenticato.

Precondizioni: L'utente ha inserito un *DID* già esistente.

Descrizione: L'utente deve inserire un *DID* diverso per registrarsi presso l'applicazione web.

Postcondizioni: L'utente ha inserito un *DID* diverso e può procedere con la registrazione.

Scenario Principale:

1. L'utente visualizza un messaggio di errore che lo informa che il *DID* inserito è già presente nel sistema.

UC4: Login

Attori Principali: Utente non autenticato.

Precondizioni: L'utente ha avviato l'applicazione web, non è già autenticato e ha aperto la pagina di autenticazione.

Descrizione: L'utente vuole entrare presso il sito e deve inserire le proprie credenziali per autenticarsi.

Postcondizioni: L'utente è autenticato correttamente e può procedere con l'utilizzo dell'applicazione.

Scenario Principale:

1. L'utente inserisce il proprio *DID*.

Estensioni:

1. Visualizzazione errore *DID* errato (UC5).

Generalizzazioni: Come da figura 4.3:

1. Inserimento *DID* (UC4.1);



Figura 4.3: UC4: Login

UC4.1: Inserimento *DID*

Attori Principali: Utente non autenticato.

Precondizioni: L'utente ha avviato l'applicazione web, non è già autenticato e ha aperto la pagina di autenticazione.

Descrizione: L'utente deve inserire il proprio *DID* associato alle proprie credenziali per autenticarsi presso l'applicazione web.

Postcondizioni: L'utente ha inserito il proprio *DID* e può procedere con l'autenticazione.

Scenario Principale:

1. L'utente inserisce il proprio *DID*.

UC5: Visualizzazione errore *DID* errato

Attori Principali: Utente non autenticato.

Precondizioni: L'utente ha inserito un *DID* errato.

Descrizione: L'utente deve inserire un *DID* corretto per autenticarsi presso l'applicazione web.

Postcondizioni: L'utente ha inserito un *DID* non esistente e può procedere con l'autenticazione.

Scenario Principale:

1. L'utente visualizza un messaggio di errore che lo informa che il *DID* inserito non è corretto.

UC6: Visualizzazione lista film

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e ha aperto la pagina principale dell'applicazione web.

Descrizione: L'utente vuole visualizzare la lista dei film presenti nel sistema.

Postcondizioni: L'utente ha visualizzato la lista dei film presenti nel sistema.

Scenario Principale:

1. L'utente visualizza la lista dei film presenti nel sistema.

Generalizzazioni: Come da figura 4.4:

1. Visualizzazione singolo film in lista (UC6.1);

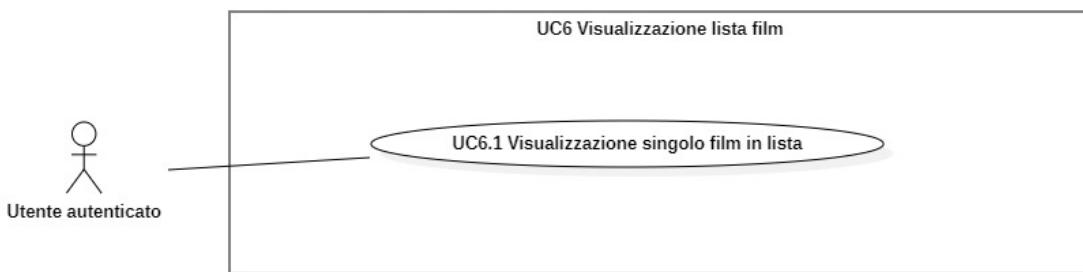


Figura 4.4: UC6: Visualizzazione lista film

UC6.1: Visualizzazione singolo film in lista

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e ha aperto la pagina principale dell'applicazione web.

Descrizione: L'utente vuole visualizzare un singolo film dalla lista dei film presenti nel sistema.

Postcondizioni: L'utente ha visualizzato un film tra la lista di quelli presenti nel sistema.

Scenario Principale:

1. L'utente visualizza un film tra quelli presenti nella lista del sistema.

UC7: Verifica dell'età

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato, possiede una Verifiable Credential che attesta la sua età e intende visualizzare un film per prenotarlo.

Descrizione: L'utente vuole visualizzare un singolo film dalla lista dei film presenti nel sistema.

Postcondizioni: L'utente ha verificato la propria età in modo sicuro e può procedere con la prenotazione del film in oggetto.

Scenario Principale:

1. L'utente seleziona un film dalla lista dei film presenti nel sistema;
2. Il sistema prende la categoria di età del film selezionato e la confronta con l'età dell'utente.
3. L'utente presenta la propria *Verifiable Credential* che attesta la sua età.
4. Il sistema verifica la correttezza della *Verifiable Credential* utilizzando il *DID* del soggetto che l'ha rilasciata e genera una *Verifiable Presentation* contenente una *Zero Knowledge Proof*.
5. Il sistema verifica la validità della *Zero Knowledge Proof* e verifica che l'età dell'utente sia maggiore o uguale a quella del film.
6. Se la verifica ha successo, l'utente può procedere con la prenotazione del film.

UC8: Prenotazione film

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato, ha verificato la propria età e ha selezionato un film dalla lista dei film presenti nel sistema.

Descrizione: L'utente vuole prenotare il film che ha selezionato dalla lista dei film presenti nel sistema.

Postcondizioni: L'utente ha verificato la propria età in modo sicuro e il film che ha selezionato è stato prenotato.

Scenario Principale:

1. L'utente seleziona un film dalla lista dei film presenti nel sistema;
2. Il sistema richiede all'utente di fornire i dettagli della prenotazione;
3. L'utente inserisce la data di prenotazione (UC8.1), l'orario di prenotazione (UC8.2) e il numero di posti da prenotare (UC8.3);

4. Il sistema verifica la disponibilità del film per la data e l'orario selezionati e registra l'avvenuta prenotazione, fornendo un riepilogo all'utente.

Generalizzazioni: Come da figura 4.5:

1. Inserimento data prenotazione (UC8.1);
2. Inserimento orario prenotazione (UC8.2);
3. Inserimento quantità biglietti (UC8.3);
4. Inserimento posti (UC8.4).

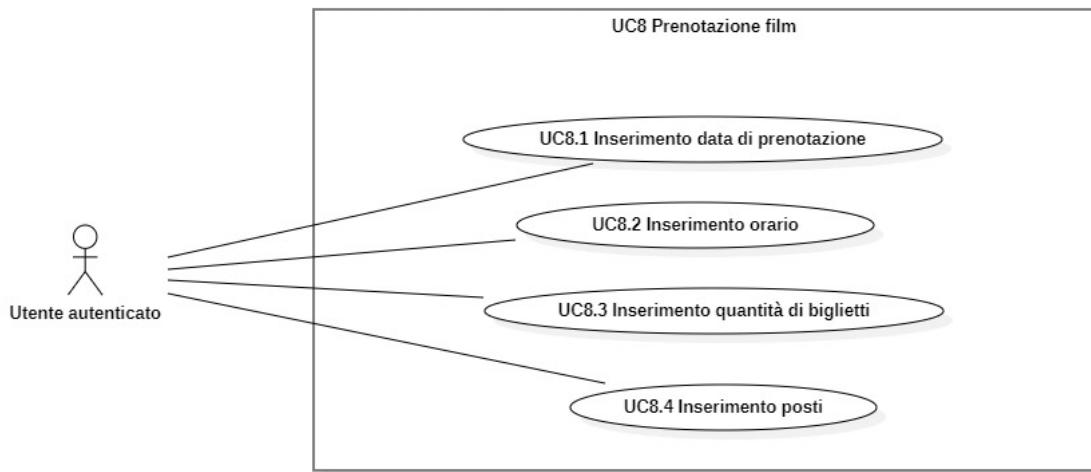


Figura 4.5: UC8: Prenotazione film

UC8.1: Inserimento data prenotazione

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato, ha verificato la propria età e ha selezionato un film dalla lista dei film presenti nel sistema.

Descrizione: L'utente vuole inserire la data di prenotazione del film che ha selezionato dalla lista dei film presenti nel sistema.

Postcondizioni: L'utente ha inserito la data di prenotazione.

Scenario Principale:

1. L'utente inserisce la data di prenotazione per il film che intende prenotare.

UC8.2: Inserimento orario prenotazione

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato, ha verificato la propria età e ha selezionato un film dalla lista dei film presenti nel sistema.

Descrizione: L'utente vuole inserire l'orario di prenotazione del film che ha selezionato dalla lista dei film presenti nel sistema.

Postcondizioni: L'utente ha inserito l'orario di prenotazione.

Scenario Principale:

1. L'utente inserisce l'orario di prenotazione per il film che intende prenotare.

UC8.3: Inserimento quantità biglietti

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato, ha verificato la propria età e ha selezionato un film dalla lista dei film presenti nel sistema.

Descrizione: L'utente vuole inserire il numero di biglietti da prenotare per il film che ha selezionato dalla lista dei film presenti nel sistema.

Postcondizioni: L'utente ha inserito il numero di biglietti che intende prenotare.

Scenario Principale:

1. L'utente inserisce il numero di biglietti da prenotare per il film che intende prenotare.

UC8.4: Inserimento posti

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato, ha verificato la propria età e ha selezionato un film dalla lista dei film presenti nel sistema.

Descrizione: L'utente vuole inserire il numero posti che intende prenotare per il film che ha selezionato dalla lista dei film presenti nel sistema.

Postcondizioni: L'utente ha inserito il numero di posti che intende prenotare.

Scenario Principale:

1. L'utente inserisce il posto o il numero di posti che intende prenotare per il film che ha selezionato.

UC9: Visualizzazione lista prenotazioni

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e ha effettuato almeno una prenotazione.

Descrizione: L'utente vuole visualizzare la lista delle prenotazioni effettuate.

Postcondizioni: L'utente ha visualizzato la lista delle prenotazioni effettuate.

Scenario Principale:

1. L'utente accede al proprio profilo e clicca sulla sezione di lista delle prenotazioni;
2. L'utente visualizza una finestra contenente la lista delle prenotazioni effettuate;
3. Il sistema restituisce la lista delle prenotazioni effettuate dall'utente.

Generalizzazioni: Come da figura 4.6:

1. Visualizzazione singola prenotazione in lista (UC9.1).

UC9.1: Visualizzazione singola prenotazione in lista

Attori Principali: Utente autenticato.

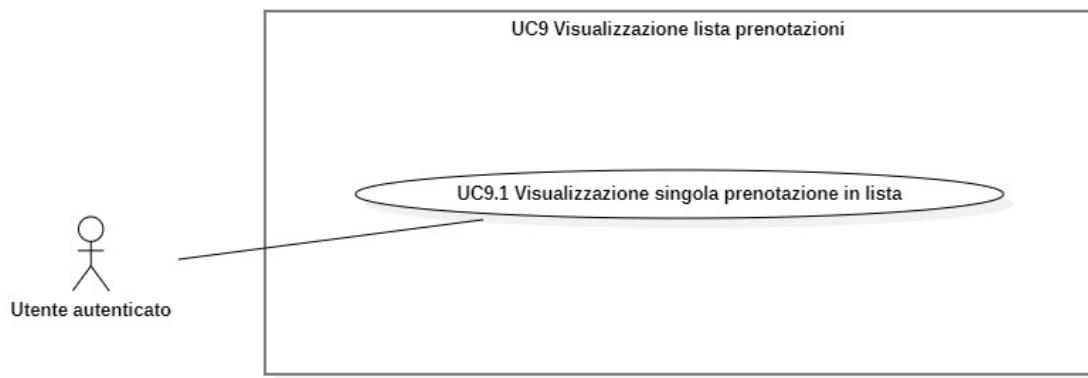


Figura 4.6: UC9: Visualizzazione lista prenotazioni

Precondizioni: L'utente è autenticato e ha effettuato almeno una prenotazione.

Descrizione: L'utente vuole visualizzare i dettagli di una prenotazione effettuata.

Postcondizioni: L'utente ha visualizzato i dettagli di una prenotazione effettuata.

Scenario Principale:

1. L'utente accede al proprio profilo e clicca sulla sezione di lista delle prenotazioni;
2. L'utente visualizza una finestra contenente la lista delle prenotazioni effettuate;
3. L'utente clicca su una prenotazione;
4. Il sistema restituisce i dettagli della prenotazione selezionata dall'utente.

Generalizzazioni: Come da figura 4.7:

1. Visualizzazione dettagli film prenotato (UC9.1.1);
2. Visualizzazione data prenotazione (UC9.1.2);
3. Visualizzazione orario prenotazione (UC9.1.3);
4. Visualizzazione quantità biglietti prenotazione (UC9.1.4);
5. Visualizzazione numero posti prenotati (UC9.1.5).

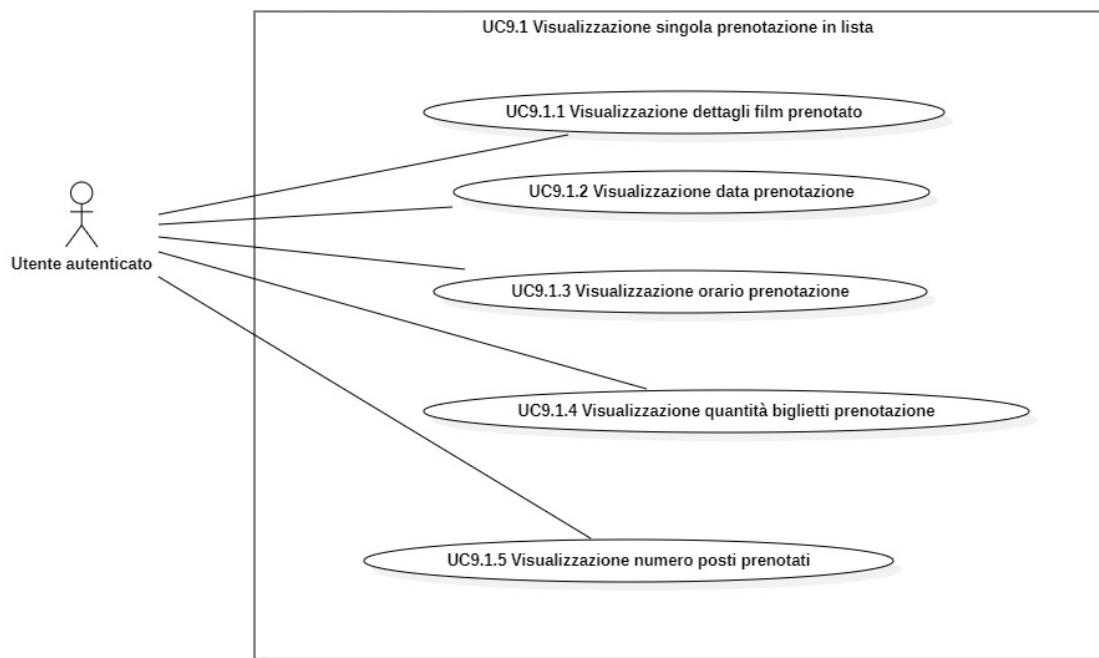


Figura 4.7: UC9.1: Visualizzazione singola prenotazione in lista

UC9.1.1: Visualizzazione dettagli film prenotato

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e ha effettuato almeno una prenotazione.

Descrizione: L'utente vuole visualizzare i dettagli del film prenotato.

Postcondizioni: L'utente ha visualizzato i dettagli del film prenotato.

Scenario Principale:

1. L'utente accede alla propria lista delle prenotazioni e visualizza i dettagli del film per quella prenotazione.

UC9.1.2: Visualizzazione data prenotazione

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e ha effettuato almeno una prenotazione.

Descrizione: L'utente vuole visualizzare la data della singola prenotazione.

Postcondizioni: L'utente ha visualizzato la data della specifica prenotazione.

Scenario Principale:

1. L'utente accede alla propria lista delle prenotazioni e visualizza la data della prenotazione.

UC9.1.3: Visualizzazione orario prenotazione

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e ha effettuato almeno una prenotazione.

Descrizione: L'utente vuole visualizzare l'orario della singola prenotazione.

Postcondizioni: L'utente ha visualizzato l'orario della specifica prenotazione.

Scenario Principale:

1. L'utente accede alla propria lista delle prenotazioni e visualizza l'orario della prenotazione.

UC9.1.4: Visualizzazione quantità biglietti prenotazione

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e ha effettuato almeno una prenotazione.

Descrizione: L'utente vuole visualizzare la quantità di biglietti della singola prenotazione.

Postcondizioni: L'utente ha visualizzato la quantità di biglietti della specifica prenotazione.

Scenario Principale:

1. L'utente accede alla propria lista delle prenotazioni e visualizza la quantità di biglietti della prenotazione.

UC9.1.5: Visualizzazione numero posti prenotati

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e ha effettuato almeno una prenotazione.

Descrizione: L'utente vuole visualizzare il numero di posti prenotati per la singola prenotazione.

Postcondizioni: L'utente ha visualizzato il numero di posti prenotati per la specifica prenotazione.

Scenario Principale:

1. L'utente accede alla propria lista delle prenotazioni e visualizza il numero di posti prenotati per la prenotazione.

UC10: Ricerca film per titolo

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e intende ricercare un film presente nel sistema.

Descrizione: L'utente vuole cercare un film presente nel sistema e nel sistema sono presenti dei film.

Postcondizioni: L'utente ha cercato un film presente nel sistema e il sistema ha restituito i risultati della ricerca, permettendo all'utente di interagire con essi.

Scenario Principale:

1. L'utente accede alla pagina principale contenente la lista dei film presenti nel sistema;
2. L'utente inserisce il titolo del film che vuole cercare;
3. Il sistema ricerca il film all'interno del sistema e restituisce i risultati della ricerca all'utente.

UC11: Recensione film

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e intende recensire un film presente nel sistema.

Descrizione: L'utente vuole recensire un film presente nel sistema.

Postcondizioni: L'utente ha recensito correttamente un film tra quelli presenti, anche senza prenotazione.

Scenario Principale:

1. L'utente accede alla pagina principale contenente la lista dei film prenotati;

2. L'utente seleziona il film che vuole recensire;
3. L'utente inserisce la recensione del film;
4. Il sistema registra la recensione del film e la associa all'utente che l'ha inserita.

UC12: Modifica informazioni profilo

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e intende modificare le proprie informazioni personali all'interno del proprio profilo.

Descrizione: L'utente modifica correttamente le proprie informazioni personali in base alle proprie esigenze.

Postcondizioni: L'utente ha modificato le proprie informazioni personali di proprio interesse.

Scenario Principale:

1. L'utente accede alla pagina principale contenente le proprie informazioni personali;
2. L'utente modifica il proprio *username* (UC12.1);
3. L'utente modifica la propria *email* (UC12.2);
4. L'utente modifica il proprio *DID* (UC12.3);
5. L'utente modifica la propria data di nascita (UC12.4);

Estensioni:

1. Visualizzazione errore *email* già esistente (UC2);
2. Visualizzazione errore *DID* già esistente (UC3).

Generalizzazioni: Come da figura 4.8:

1. Inserimento nuovo *username* (UC12.1);
2. Inserimento nuova *email* (UC12.2);
3. Inserimento nuova data di nascita (UC12.3).

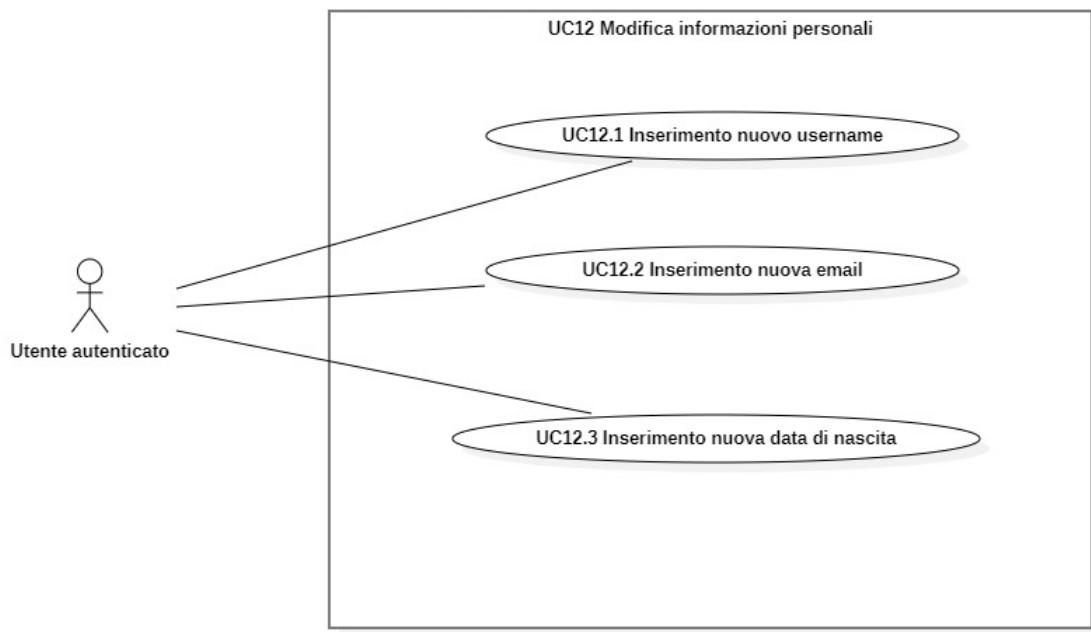


Figura 4.8: UC12: Modifica informazioni profilo

UC12.1: Inserimento nuovo *username*

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e intende modificare il proprio *username*.

Descrizione: L'utente vuole modificare lo *username* associato al proprio *account*.

Postcondizioni: L'utente ha inserito il proprio *username* correttamente.

Scenario Principale:

1. L'utente modifica il proprio *username* e conferma l'operazione.

UC12.2: Inserimento nuova *email*

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e intende modificare la propria *email*.

Descrizione: L'utente vuole modificare la mail associata al proprio *account*.

Postcondizioni: L'utente ha modificato la propria *email* correttamente.

Scenario Principale:

1. L'utente modifica la propria *email* e conferma l'operazione.

UC12.3: Inserimento nuova data di nascita

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e intende modificare la propria data di nascita.

Descrizione: L'utente vuole modificare la data di nascita associata al proprio *account*.

Postcondizioni: L'utente ha modificato la data di nascita correttamente.

Scenario Principale:

1. L'utente modifica la propria data di nascita e conferma l'operazione.

UC13: Eliminazione *account*

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e intende eliminare il proprio *account*.

Descrizione: L'utente vuole eliminare il proprio *account* dal sistema.

Postcondizioni: L'utente ha eliminato il proprio *account* e il sistema ha eliminato tutti i dati ad esso associati.

Scenario Principale:

1. L'utente accede alla pagina principale contenente di modifica del profilo contenente le proprie informazioni;
2. L'utente elimina il proprio *account*;
3. Il sistema elimina l'*account* dell'utente, cancellando il *DID* e le informazioni ad esso associate.

UC14: Logout

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e intende uscire dalla propria sessione.

Descrizione: L'utente vuole effettuare il *logout* dal sistema.

Postcondizioni: L'utente ha effettuato il *logout* e il sistema ha terminato la sessione dell'utente, non permettendo più l'accesso alle funzionalità riservate.

Scenario Principale:

1. L'utente accede alla pagina principale contenente le proprie informazioni personali;
2. L'utente effettua il *logout*;
3. Il sistema effettua il *logout* dell'utente e l'utente è uscito dalla propria sessione.

4.3 Tracciamento dei requisiti

In questa sezione sono individuate le corrispondenze tra i requisiti individuati e i casi d'uso, stilando una serie di tabelle di tracciamento (requisiti funzionali a 4.1, di vincolo a 4.2 e qualitativi a 4.3) tra i casi d'uso e le loro fonti, e viceversa. Ciascun requisito ha come struttura la seguente:

R[Importanza][Tipo][Codice]

avendo come legenda:

- **Importanza** comprendente **O** per obbligatorio e **D** per desiderabile;
- **Tipo** comprendente **F** per funzionale, **Q** per qualitativo e **V** per di vincolo;
- **Codice** come codice identificativo univoco del requisito.

Le fonti dei requisiti sono:

- **UC** = use case ed **Interno** = requisito individuato dall'insieme di stagisti e tutor aziendali.

Tabella 4.1: Tabella del tracciamento dei requisiti funzionali

Requisito	Descrizione	Fonte
RFO-1	Il sistema permette la registrazione di un nuovo utente	UC1
RFO-1.1	L'utente deve inserire il proprio <i>username</i>	UC1.1
RFO-1.2	L'utente deve inserire la propria <i>email</i>	UC1.2
RFO-1.3	L'utente deve inserire il proprio <i>DID</i>	UC1.3
RFO-1.4	L'utente deve inserire la propria data di nascita	UC1.4
RFO-2	Il sistema deve visualizzare un errore se l' <i>email</i> inserita è già esistente	UC2
RFO-3	Il sistema deve visualizzare un errore se il <i>DID</i> inserito è già esistente	UC3
RFO-4	Il sistema deve permettere l'autenticazione ad un utente che abbia già effettuato la registrazione	UC4
RFO-4.1	L'utente deve inserire il proprio <i>DID</i> in fase di autenticazione	UC4.1
RFO-5	Il sistema deve visualizzare un errore se il <i>DID</i> inserito non è esistente	UC5
RFO-6	Il sistema deve poter visualizzare la lista dei film presenti nel sistema	UC6
RFO-6.1	Il sistema deve poter permettere la visualizzazione delle informazioni di un singolo film nella lista	UC6.1
RFO-7	Il sistema deve poter verificare l'età dell'utente, permettendo allo stesso di essere identificato senza rivelare proprie informazioni non necessarie	UC7
RFO-8	Il sistema deve poter permettere la prenotazione di un film selezionato	UC8
RFO-8.1	Il sistema deve poter permettere la selezione di una data per la prenotazione	UC8.1
RFO-8.2	Il sistema deve poter permettere la selezione di un orario per la prenotazione	UC8.2
RFO-8.3	Il sistema deve poter permettere la selezione di un numero di biglietti per la prenotazione	UC8.3
RFO-8.4	Il sistema deve poter permettere la selezione di un posto per la prenotazione	UC8.4
RFO-9	L'utente deve poter visualizzare la lista delle prenotazioni effettuate	UC9
RFO-9.1	L'utente deve poter visualizzare i dettagli di una singola prenotazione	UC9.1
RFO-9.1.1	L'utente deve poter visualizzare i dettagli del film della prenotazione	UC9.1.1
RFO-9.1.2	L'utente deve poter visualizzare la data della singola prenotazione	UC9.1.2
RFO-9.1.3	L'utente deve poter visualizzare l'orario della singola prenotazione	UC9.1.3
RFO-9.1.4	L'utente deve poter visualizzare la quantità di biglietti della prenotazione	UC9.1.4
RFO-9.1.5	L'utente deve poter visualizzare il numero di posti compresi nella singola prenotazione	UC9.1.5
RFO-10	Il sistema deve poter permettere la ricerca di un film presente per titolo	UC10
RFO-11	Il sistema deve poter permettere la recensione dei film presenti	UC11
RFD-12	Il sistema deve poter permettere la modifica del profilo utente	UC12
RFD-12.1	L'utente deve poter modificare il proprio <i>username</i>	UC12.1
RFD-12.2	L'utente deve poter modificare la propria <i>email</i>	UC12.2
RFD-12.3	L'utente deve poter modificare la propria data di nascita	UC12.3
RFO-13	Il sistema deve poter permettere l'eliminazione dell' <i>account</i> utente	UC13
RFO-14	Il sistema deve poter permettere all'utente di uscire dalla propria sessione	UC14

Tabella 4.2: Tabella del tracciamento dei requisiti di vincolo

Requisito	Descrizione	Fonte
RVO-1	Il front-end deve essere sviluppato utilizzando il linguaggio di programmazione <i>TypeScript</i> e con la libreria <i>React</i>	Interno
RVO-2	Il sistema deve comunicare con lo <i>smart contract</i> di Alessio De Biasi usandolo come libreria per la verifica dell'identità	Interno
RVO-3	Il sistema deve utilizzare un'implementazione di <i>Self Sovereign Identity</i> , utilizzando <i>Decentralized Identifiers</i> , <i>Verifiable Credentials</i> , <i>Verifiable Presentation</i> e <i>Zero Knowledge Proof</i>	Interno
RVO-4	Il sistema deve rispettare le leggi sulla <i>privacy</i> e la protezione dei dati personali, permettendo all'utente di non divulgare alcun dato personale nel riconoscimento della propria identità ed età	Interno
RVD-5	Il sistema deve poter integrare un meccanismo interno di <i>challenge-response</i> in fase di registrazione ed autenticazione, senza dipendere da applicazioni esterne quali Metamask	Interno

Tabella 4.3: Tabella del tracciamento dei requisiti qualitativi

Requisito	Descrizione	Fonte
RQO-1	Deve essere redatto un documento che descriva il percorso di studio e una descrizione di massima dell'applicazione realizzata	Interno
RQO-2	Il codice frontend deve essere coperto da test di unità	Interno
RQO-3	Il codice dello <i>smart contract</i> deve essere coperto da test di unità	Interno
RQD-4	L'applicazione web deve essere responsive e visibile da dispositivi mobili	Interno
RQD-5	L'applicazione web deve essere accessibile da tutti i principali <i>browser</i>	Interno

Capitolo 5

Progettazione e codifica del progetto

In questa sezione, saranno elencate le tecnologie principali utilizzate durante lo sviluppo del sistema oggetto del tirocinio. Inoltre, verrà descritto il ciclo di vita del software, le fasi di progettazione e codifica e le scelte architettoniche realizzate, a livello di sistema e design pattern.

5.1 Componenti principali del sistema

Il sistema risulta essere composto da due parti principali:

- ^g**front-end**: è la parte visibile all'utente, che permette di interagire con il sistema. È stata sviluppata utilizzando il ^g**framework** *React* e il linguaggio *TypeScript*. Questa comprende la parte grafica da me realizzata e la relativa interazione con il contratto fornito come libreria dallo studente Alessio De Biasi.
- ^g**back-end**: è la parte che gestisce la logica dell'applicazione, che permette di interagire con la *blockchain* e di gestire le funzionalità del sistema. È stata sviluppata utilizzando il linguaggio *Solidity*. Quest'ultima comprende il contratto sviluppato dallo studente magistrale Alessio De Biasi basato sugli standard ^g**W3C**, ^g**Self Sovereign Identity** e ^g**Decentralized Identifier**.

5.2 Tecnologie utilizzate

5.2.1 Codifica front-end

5.2.1.1 React

React è una libreria *JavaScript* utilizzata per la creazione di interfacce utente. È stata sviluppata da *Facebook* e rilasciata nel 2013. Essa consente di creare dei componenti riutilizzabili che rappresentano parti dell'interfaccia e gestire lo stato dell'applicazione in modo efficiente e scalabile. Questo lo rende adatto allo sviluppo di applicazioni complesse e dinamiche (specifiche e riferimenti a [13]). Nel mio caso, è utilizzato per la progettazione delle componenti e delle pagine.

5.2.1.2 TypeScript

TypeScript è un linguaggio di programmazione ^g**open source** sviluppato da *Microsoft*. È un super-set di *JavaScript* che aggiunge tipi statici opzionali al linguaggio, permettendo di scrivere codice più robusto e manutenibile, grazie alla possibilità di definire interfacce e classi (secondo [13]). All'interno delle pagine realizzate, sono stati creati dei tipi appositi per poter gestire più facilmente le informazioni e le funzionalità del sistema.

5.2.2 Codifica back-end

5.2.2.1 Solidity

Solidity è un linguaggio di programmazione orientato agli oggetti per la scrittura di ^g[smart contract](#). È stato sviluppato da *Ethereum* e permette di gestire lo stato di un contratto, definire funzioni e interagire con altri contratti all'interno delle *blockchain*, grazie alla gestione del sistema di transazioni basato sugli eventi e alla possibilità di definire interfacce (in base a quanto presente in [16]). L'interazione con la *blockchain* e il contratto da richiamare come libreria è stato scritto in questo linguaggio.

5.2.3 Librerie di terze parti

5.2.3.1 Node.js

Node.js è un ambiente di esecuzione *open source* per l'esecuzione di codice *JavaScript* lato *server*. È basato sul motore *JavaScript* V8 di *Google Chrome* e permette di gestire le dipendenze dell'applicazione, grazie al suo gestore di pacchetti *npm*, differenziando le librerie dell'applicazione e quelle di terze parti. Tramite semplici comandi, è possibile installare e rimuovere le dipendenze, aggiornarle e gestire le versioni. Lo strumento ha permesso una facile configurazione delle dipendenze e la gestione delle versioni (riferimenti in [12]).

5.2.3.2 Web3.js

Web3.js è una collezione di librerie *JavaScript* per poter interagire facilmente con le *blockchain*, in particolare con *Ethereum*. Essa permette di connettersi ad un nodo della *blockchain*, inviare transazioni e interagire con gli *smart contract*, gestendo facilmente il collegamento e l'interazione tra la parte grafica e gli *smart contract*, definendo modularmente le funzionalità presenti ([24]). Grazie a questo, è stato possibile gestire facilmente l'interazione con la *blockchain* e gli *smart contract*, permettendo la chiamata al contratto e alle sue funzioni.

5.2.3.3 Hardhat

Hardhat è un ambiente di sviluppo per le *blockchain* che permette di testare e distribuire *smart contract*. Esso permette di distribuire facilmente in locale per poter testare le funzionalità degli *smart contract*, e di distribuirli su una *blockchain* pubblica, come ad esempio *Ethereum* (in base alle definizioni e guide in [10]).

5.2.4 Versionamento

5.2.4.1 GitHub

GitHub è una piattaforma di *hosting* per progetti software. Essa permette di gestire il versionamento del codice tramite il sistema di controllo di versione *Git*. In particolare, consente lo sviluppo di codice attraverso un ^g[repository](#) remoto, che permette di gestire le modifiche e le versioni del codice, tenendo traccia attraverso un registro delle modifiche effettuate e permettendo di tornare ad una versione precedente del codice ([9]).

5.2.5 Verifica

5.2.5.1 ESLint

ESLint è uno strumento di analisi statica del codice per identificare i modelli problematici trovati nel codice *JavaScript* e *TypeScript*. È stato utilizzato per garantire la qualità del codice prodotto, tramite la definizione di regole e la segnalazione di eventuali errori.

5.2.5.2 Jest

Jest è un *framework* di test per *JavaScript* e *TypeScript*. È stato utilizzato per la definizione di test unitari e di integrazione, per verificare il corretto funzionamento delle funzionalità implementate tra le pagine ed i componenti.

5.3 Configurazione ambiente di sviluppo

5.3.1 Smart Contract

All'interno del mio progetto di stage, per realizzare l'implementazione degli *smart contract*, ho utilizzato un ambiente di sviluppo locale, basato su *Hardhat*, un ambiente di sviluppo per ^gEthereum che permette di testare e distribuire *smart contract*. Lo strumento permette la creazione di un ambiente di test locale, sulla base dell'impostazione fornita da specifici *script*, in grado di gestire la compilazione e la successiva esecuzione (*deploy*) del contratto fornito sulla rete locale. Per poter effettuare le singole chiamate, lo strumento fornisce un insieme di *account* di test con un saldo iniziale di 10000 *ETH* (valute della rete Ethereum, rete ^gblockchain utilizzata dallo strumento), che possono essere utilizzati per effettuare le chiamate desiderate qualora siano presenti transazioni (normalmente in esecuzione sulla porta 8545 all'interno della rete locale).

Nello specifico, la strutturazione prevede:

- uno script di *deploy*, scritto in *TypeScript*, che definisce il contratto che viene chiamato, il suo indirizzo sulla rete locale e la generazione di un file ^gJSON chiamato *ABI*, che contiene le informazioni in formato binario del contratto. Esso viene utilizzato nella parte frontend;
- uno script di test, scritto in *TypeScript*, che permette di testare le funzionalità del contratto, tramite l'ausilio di *web3.js*;
- uno script di configurazione, che definisce l'*account* utilizzato sulla rete locale e le librerie presenti.

5.3.2 Front-end

Per lo sviluppo del front-end, ho utilizzato un ambiente di sviluppo locale, basato su *Node.js*, un ambiente di esecuzione per *JavaScript*. Tramite questo, è stato possibile gestire le dipendenze del progetto, tramite il package manager *npm*, e avviare un *server* locale per poter testare e scrivere le pagine presenti. In particolare, il *server* è stato avviato sulla porta 3000, per poter permettere l'interazione con il contratto tramite le funzionalità del front-end e le chiamate al contratto. Nello specifico, la strutturazione prevede:

- una cartella *src*, contenente i file *.js* e *.ts* che definiscono le funzionalità del front-end;
- una cartella *public*, contenente i file *.html* e *.css* che definiscono le pagine del front-end;
- una cartella *build*, contenente i file *.js* e *.ts* compilati, che vengono utilizzati per l'esecuzione del front-end;
- una cartella *node_modules*, contenente le dipendenze del progetto.

5.4 Progettazione

5.4.1 Architettura e pattern front-end

L'utilizzo di *React* ha permesso di definire una struttura modulare per il front-end, in modo da poter definire facilmente le pagine e le funzionalità. Esso è formato da un insieme di componenti, che definiscono un insieme di funzioni riutilizzabili, ognuna delle quali definisce un suo comportamento. La logica dell'applicazione permette di gestire singolarmente le funzionalità del codice, definendo le pagine come funzioni che vengono esportate e utilizzate come entità indipendenti. In questo modo, si incrementa la modularità del codice e la sua manutenibilità, in quanto ogni componente o pagina può essere riutilizzato in modo semplice e può essere modificato senza influenzare il resto del codice.

La struttura della pagina è stata definita tramite *.tsx*, un'estensione di *React* che permette di definire la struttura della pagina sfruttando elementi strutturali *HTML*, la presentazione con *CSS* e il linguaggio di programmazione *TypeScript* per la gestione della logica, dei tipi e delle variabili. Tramite l'utilizzo dei cosiddetti *hook*, è stato possibile definire le funzionalità del front-end, in modo da poter gestire le chiamate tra le componenti della pagina in base all'attivazione di specifici eventi in modo dinamico, grazie all'uso degli *state*. In questo modo, i singoli componenti e le pagine, definite come *views*, riescono ad avere singola responsabilità e ne permette un miglior riutilizzo.

La logica viene mantenuta separata dalla presentazione, utilizzando la strutturazione a componenti in grado di comunicare nativamente con il *DOM (Document Object Model)* della pagina ma aggiungendo la gestione degli eventi in modo nativo tramite *React*, permettendo così di poter gestire le interazioni con l'utente solo tramite le componenti e le pagine stesse, separando il codice della pagina dalla sua presentazione realizzata con *CSS*.

Dato che *React* è una libreria grafica, non viene forzato alcun preciso pattern architetturale, ma viene lasciata libertà di scelta allo sviluppatore. In questo caso, il pattern utilizzato è il *Flux Pattern*, che permette di definire un flusso unidirezionale dei dati, in modo da poter gestire le interazioni dell'utente con la pagina e le chiamate al contratto in modo semplice e senza dover gestire la sincronizzazione dei dati tra le varie componenti. Ciò permette all'applicazione di essere gestita da funzionalità precise a cascata, andando ad interagire con l'esterno con semplici aggiunte di moduli o dipendenze. Come visibile dalla figura 5.1, la logica si compone di un componente radice che descrive l'applicazione, a sua volta costituita eventualmente da uno o più componenti, innestati a vari livelli secondo la relazione di composizione, includendo anche eventuali componenti di terze parti per permettere la gestione e l'interazione all'esterno, esportando la funzionalità realizzata dal componente o dalla pagina modularmente.

Secondo le *best practices* di *React*, sono stati utilizzati un insieme di *design pattern* per normare la codifica tipici della libreria grafica: è stato utilizzato il pattern *Functional Component*, che permette di definire proprietà specifiche (definite come *props*) ed uno stato interno in grado di interagire ad eventi asincroni generati da azioni compiute dall'utente nelle varie condizioni definite. La strutturazione si compone di un insieme di funzioni e stati definiti al caricamento della pagina (utilizzando l'*Hook Pattern*, attraverso gli *hooks useState* e *useEffect*), in grado di caricare e gestire dinamicamente le informazioni al suo interno, esportando poi il componente funzionale o la stessa pagina, secondo un insieme di funzionalità definite dal contesto specifico. Le componenti stesse supportano nativamente il cambiamento dei propri stati tramite la definizione di funzioni di *callback*, eseguendo le funzioni definite al cambiamento di stato, osservando le azioni dell'utente (definito come *Controlled Components* nel contesto *React*) secondo il noto pattern *Observer* alle chiamate definite dagli eventi stessi.

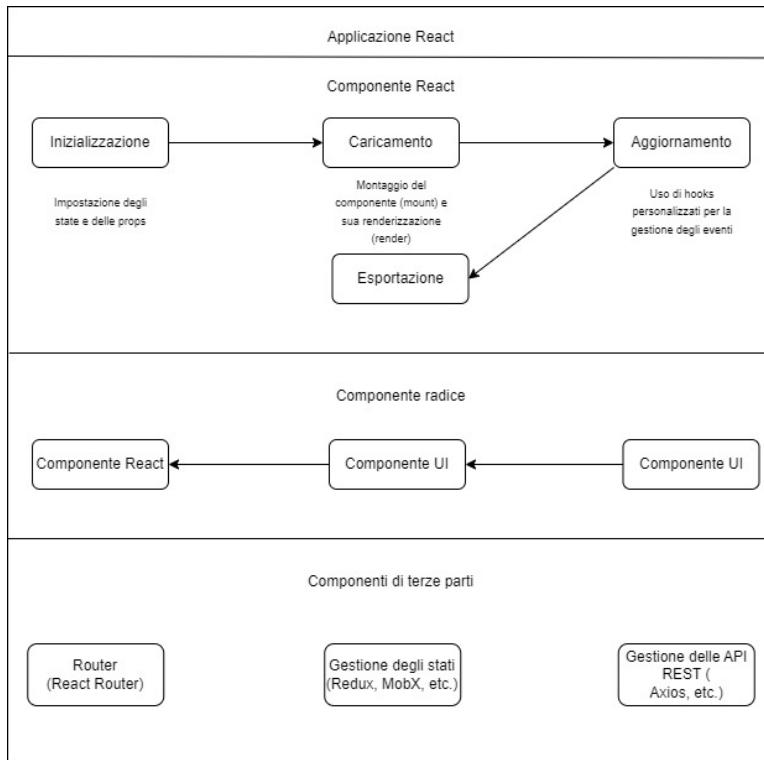


Figura 5.1: Schema logico applicazione React

Nello specifico, l'applicazione definisce per ogni pagina una cartella comprensiva del componente funzionale esportato e del suo stile, e un file di test per verificare le principali funzionalità previste. La suddivisione delle cartelle individuata riflette la divisione delle responsabilità; di massima, è presente una cartella *components*, contenente i componenti utilizzati da tutte le pagine comprensive della barra di navigazione, del *footer* e di eventuali componenti di terze parti, una cartella *views* con le pagine realizzate per l'applicazione ed una cartella *utils*, con l'insieme delle funzionalità definite all'avvio dell'applicazione e centrali per la comunicazione con *blockchain*.

L'autenticazione viene gestita tramite un contesto, definito dal *Provider Pattern* di *React*, che permette di definire un contesto globale in cui tutta l'applicazione e le sue componenti vengono racchiuse per poter accedere alle informazioni definite al suo interno in ogni momento. Le singole opzioni visibili tra utente autenticato e non autenticato sono invece gestite secondo il pattern *Conditional Rendering*, in grado di differenziare le opzioni visibili in base al suo stato.

5.4.2 Architettura e pattern back-end

Lo *smart contract* da me utilizzato come libreria per la gestione delle funzionalità di base è stato realizzato in *Solidity*, un linguaggio di programmazione orientato agli oggetti per la definizione di *smart contract* per la piattaforma *Ethereum*. Il contratto è stato creato secondo il pattern *Factory*, che permette di definire un contratto principale, in grado di creare e gestire altri contratti secondari, ciascuno responsabile della creazione di un'istanza dei documenti di identità con ⁶[Decentralized Identifier](#) con relativi attributi ed operazioni associate. Inoltre, per permettere ai singoli componenti di interagire con il contratto, è stato utilizzato il pattern *Singleton*, che assicura la creazione dell'unica istanza del contratto, in grado di essere utilizzata da tutte le componenti dell'applicazione.

L'interazione tra il contratto e la parte frontend si basa sull'interazione nativa e la gestione delle chiamate generate dallo *smart contract* tramite l'utilizzo della libreria *Node.js*, in grado di gestire le chiamate asincrone e la comunicazione con il contratto stesso tramite ascolto da parte dei componenti e delle pagine presenti nel

front-end dell'applicazione, secondo il pattern *Observer*. In questo modo, secondo l'architettura a componenti definita come flusso, è possibile separare completamente la logica di interazione con il contratto dalla logica di visualizzazione, utilizzando le funzionalità definite con la creazione di una sola riga di codice ed effettuando le chiamate alle funzionalità previste nella parte di codifica solamente quando necessario, permettendo la relativa sincronizzazione dei componenti che, ascoltando, aggiornano i propri stati secondo le chiamate effettuate.

Hardhat, in grado di creare una locale di test, agisce quindi da *middleware* tra la parte front-end e la *block-chain*, permettendo di simulare le chiamate al contratto ed ugualmente all'utente di interagire con l'applicazione, senza dover effettivamente effettuare le transazioni sulla rete principale. Per semplicità di implementazione, le chiamate sono effettuate sulla rete locale con l'utilizzo di profili di test già esistenti, in grado di effettuare le transazioni (ad ogni chiamata corrisponde una spesa in termini di valuta, coperta completamente dai profili di test).

Si consideri che l'applicazione simula il funzionamento della *blockchain Ethereum*, per la quale la descrizione di un'architettura viene descritta secondo il modello *client-server*, in cui il *client* è rappresentato dal nodo che effettua la richiesta e il *server* dal nodo che la riceve e la processa. Il modello definito come *Dapp architecture*, descrive complessivamente l'interazione adottata nel progetto, come visibile da figura 5.2.

- l'utilizzo di una base di dati locale nel quale vengono salvati i dati relativi all'utente, in grado di essere utilizzati per l'autenticazione e per la creazione dei documenti di identità. Nel caso del progetto, ogni dato viene salvato in locale all'interno di *localStorage*, successivamente cifrata localmente secondo quanto sarà descritto in sezioni di codifica e poi effettuando le chiamate al contratto per la creazione dei documenti di identità tramite le chiamate del *server* presente;
- l'utilizzo del codice per memorizzare le transazioni, che permette di considerare uno *smart contract*, compilarlo e inserire il suo *bytecode* in formato binario e così poterlo eseguire sulla *blockchain*. Ogni operazione effettuata presso il contratto comporta l'operazione di *mining* di un blocco e conseguente spesa di valuta. Questo viene reso possibile grazie alla *Ethereum Virtual Machine (EVM)*, che fornisce un codice univoco ed immutabile, poi usato dai nodi della rete.

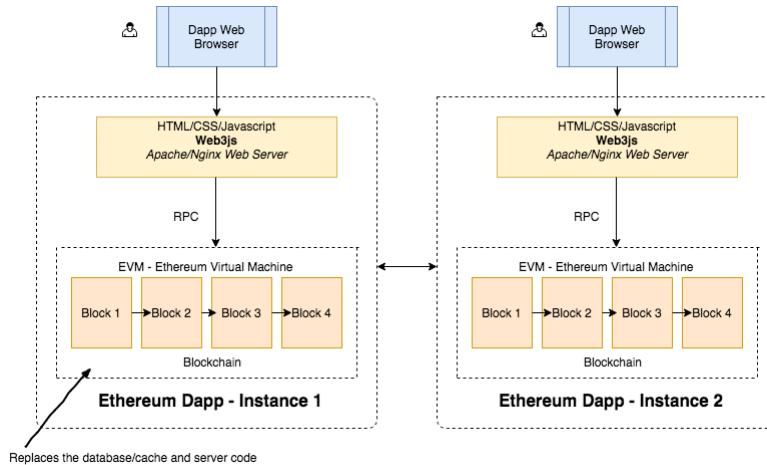


Figura 5.2: Architettura Ethereum
Fonte dell'immagine: [7]

Complessivamente, l'applicazione avrà graficamente l'architettura visibile in figura 5.3, dove viene descritta la composizione del contratto e la sua interazione con la parte front-end dell'applicazione, interagendo con la *blockchain* di test tramite *Hardhat*.

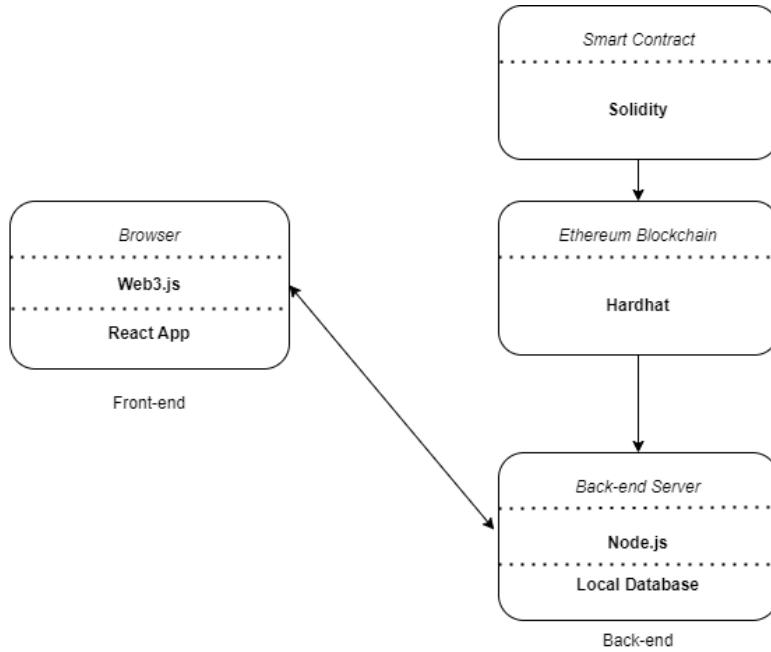


Figura 5.3: Architettura applicazione

5.5 Codifica

In questa sezione la descrizione della codifica è suddivisa in due parti, una per la parte back-end e una per la parte front-end. Si è scelta questa suddivisione per permettere di comprendere subito il contratto librerie utilizzato e le sue funzionalità, al fine di poter avere una comprensione più chiara della parte front-end, che si basa su di esso per poter implementare quanto previsto in precedenza.

5.5.1 Codifica back-end

Il contratto che viene utilizzato come librerie è parte di un'implementazione della tecnologia ⁶[Self Sovereign Identity](#) descritta opportunamente nella sezione 2.3. Esso offre funzionalità avanzate che consentono la creazione e la gestione di documenti di identità unici per ciascun utente, associando a ciascuno un ⁶[Decentralized Identifier](#) considerato univoco. Globalmente, il contratto definisce le seguenti strutture dati:

- **Parent**, che rappresenta il collegamento gerarchico tra i documenti di identità rilasciato, associando un identificativo al genitore, un campo di conferma validità ed una firma;
- **VerificationMethod**, che rappresenta il metodo di verifica associato al documento di identità, comprensivo di un indice, un identificativo, un metodo di verifica di firma digitale, un controllore, un identificativo della *blockchain* associata e un indirizzo dell'*account* associato a *blockchain* da verificare;
- **Service**, che rappresenta il servizio associato al documento di identità, comprensivo di un identificativo, un tipo di servizio e il suo indirizzo *URL*;
- **DidDocumentData**, che rappresenta i metadati associati ad uno specifico documento di identità, con un suo identificativo, la sua prova di validità, tre mappe per i tipi di autenticazione, deleghe di capacità e servizi dei documenti ad essi associati e un *e* un *Parent*. Ciò risulta utile nella verifica della catena di fiducia implementata successivamente;

- **DidDocument**, che rappresenta il documento di identità associato all’utente e comprende un identificativo, un metodo di verifica, una delegazione di capacità, un servizio e un *Parent*;
- **ResolutionResult**, che rappresenta il risultato della risoluzione del documento di identità, comprendente un identificativo il documento di tipo *DidDocument* associato;
- **ChainResolutionResult**, un vettore di stringhe che racchiude i risultati della risoluzione dei vari documenti di identità associati.

Il contratto prevede una serie di metodi, ciascuno con un suo scopo, da me successivamente utilizzati nella parte front-end dell’applicazione. Ogni metodo descritto è stato utilizzato all’interno di una pagina o componente a seconda dello scopo voluto; globalmente, possiamo specificare:

- **createDid**, che permette la creazione di un documento di identità, associando un *DID* univoco all’utente che ha effettuato la transazione e ritorna il *DID* del nuovo utente aggiunto;
- **createChildTrustedDid**, in grado di creare un nuovo *DID Document* che afferma la delegazione di fiducia dall’utente certificatore (cosiddetto *certification authority*) all’utente con il *DID* specificato. Questo permette di creare la catena degli *issuer* fidati, partendo dalla detta *certification authority* e arrivando fino all’utente che ha effettuato la transazione, verificando i suoi dati in modo sicuro. Esso prende come parametri l’indirizzo dell’utente a cui delegare la fiducia e la firma della *certification authority*;
- **initializeDidDocument**, richiamato dal metodo *createDid*, che inizializza il documento di identità associato all’utente, prendendo come parametri il *DID* dell’utente e il suo indirizzo;
- **addCapabilityDelegation**, che permette di aggiungere una delega di capacità al documento di identità, prendendo come parametro l’indirizzo dell’utente che autenticherà la transazione;
- **addService**, che permette di aggiungere un servizio al documento di identità, prendendo come parametri, l’identificativo, il tipo di servizio e il suo indirizzo *URL*;
- **deactivate**, il quale disattiva il *DID Document* associato all’utente che ha effettuato la transazione;
- **resolve**, che ritorna il *DID Document* associato all’utente secondo un determinato *DID* passato come parametro;
- **resolveChain**, che ritorna la catena di fiducia percorsa avendo come ultimo nodo l’utente con il *DID* specificato e ritorna la lista di utenti certificatori a cui si è passati per arrivare all’utente finale;
- **getAuthentication**, per ottenere il metodo di autenticazione associato al documento di identità e provare per certo l’utente abbia acceduto secondo il metodo corretto.

Al fine di evitare le vulnerabilità descritte in sezione [2.2.4](#) e garantire la sicurezza del contratto, sono state implementate le seguenti misure:

- **accesso controllato ai metodi**: i metodi del contratto sono stati resi accessibili solo all’utente che ha effettuato la transazione, in modo da evitare che altri utenti possano accedere ai dati di altri utenti;
- **cifratura dei dati e dei profili presenti**: i dati sensibili sono stati cifrati prima di essere salvati sulla *blockchain*, in modo da evitare che altri utenti possano accedervi. In particolare, sono stati usati dei metodi di controllo forniti dalla libreria *web3.js* che permette la creazione di *wallet* sicuri partendo dalla coppia di chiavi per ogni account sulla rete locale di test presente;
- **utilizzo di nonce e gas limit**: per ogni transazione effettuata, è stato specificato un *nonce* e un *gas limit* per evitare che altri utenti possano effettuare transazioni al posto dell’utente che ha effettuato la transazione oppure per eseguire transazioni ad un prezzo ben più alto di quello previsto;
- **utilizzo di modifier**: per ogni metodo del contratto è stato definito un *modifier* apposito che permette di controllare che l’utente che ha effettuato la transazione sia effettivamente quello che ha creato il documento di identità e che il documento di identità sia attivo;
- **utilizzo di eventi**: per ogni metodo del contratto è stato definito un *evento* apposito che permette di notificare l’utente che ha effettuato la transazione dell’esito della stessa, verificando la corretta esecuzione del metodo chiamato.

5.5.2 Codifica front-end

5.5.2.1 Pagine realizzate

5.5.2.1.1 Home Page La pagina definita in codice come *HomePage* è la prima schermata visibile collegandosi al sito, come visibile in figura 5.4. Essa permette di accedere alle funzionalità principali dell'applicazione. In particolare, è possibile accedere alla pagina di registrazione e alla pagina di *login* attraverso l'utilizzo degli appositi componenti. Essa utilizza i componenti principali di navigazione (nello specifico *Footer* e *NavBar* per gestire le opzioni previste per gli utenti a seconda che siano o meno autenticati). L'intera applicazione è infatti racchiusa nel componente *AuthContext*, che permette di gestire lo stato di autenticazione dell'utente. Per questa, come le altre pagine, ove non diversamente specificato, è prevista l'esportazione di default come funzione *TypeScript*, includendo i componenti utili ove previsto e separando la struttura impostata in *HTML* e la presentazione impostata in *CSS* da file omonimo, presente nella stessa cartella della pagina.



Figura 5.4: Pagina principale

Si noti l'utilizzo di un'immagine di sfondo senza diritti d'autore, proveniente dal sito *Unsplash*, che permette di rendere la pagina più accattivante e di presentare l'idea di un'identità digitale. Tutto il sito è stato anche sviluppato per consentire una piena navigazione da mobile, come visibile in figura 5.5. Presenta infatti un *layout responsive*, che si adatta alle dimensioni dello schermo del dispositivo utilizzato, avendo come punto di rottura una dimensione di 768 pixel.



Figura 5.5: Pagina principale vista da cellulare

In questa pagina, viene creato il *DID* associato all'utente e la catena di *issuer* di test che permette di verificare la validità del meccanismo di catena di fiducia, poi spiegato in dettaglio nella pagina dei film dove viene attivato tale meccanismo.

5.5.2.1.2 Registrazione Nella pagina in oggetto, definita in codice come *RegisterView*, l’utente è portato a registrarsi all’applicazione, inserendo i dati richiesti, come visibile in figura 5.6. Come descritto dai casi d’uso, vi è un controllo sulla validità dei dati inseriti, verificando se *DID* o la email inserita siano già esistenti. Questa è la prima che utilizza lo *smart contract* libreria direttamente; viene infatti implementato un meccanismo di *challenge-response* per verificare che l’utente sia in possesso del *DID* inserito. Tale meccanismo è stato implementato per non dipendere da piattaforme esterne legate a *blockchain* come la nota estensione *MetaMask*, al fine di realizzare un sistema interno che permetta di confermare la validità dei dati dell’utente, sempre usando *blockchain*, sia per il meccanismo di *login* che registrazione, come richiesto dal proponente.

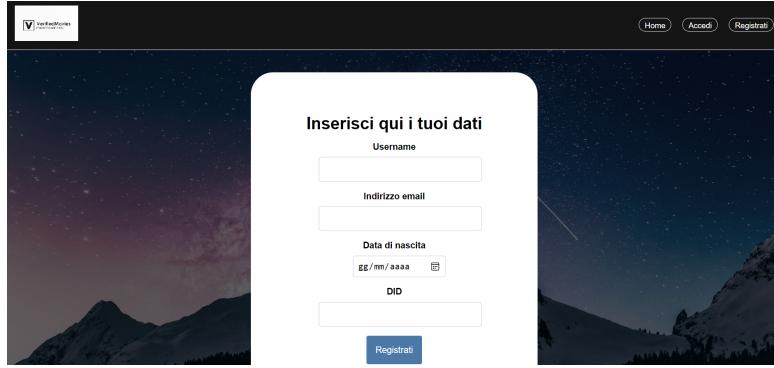


Figura 5.6: Pagina principale

Si prevede, come visibile in figura 5.7, che l’utente cifri con la propria chiave privata un numero presentato a schermo; in questa fase viene generata una prova di autenticazione in formato *JSON*, che dimostra che l’utente vuole autenticarsi con il suo *DID* avendo firmato il numero e chiamando il metodo *getAuthentication* dello *smart contract*. Se la firma digitale corrisponde e il *DID* corrisponde a quello creato inizialmente, allora la verifica passa e la registrazione è confermata con successo, avvisando l’utente con un messaggio apposito.

Conferma la tua identità.

Questo è il tuo numero:

900479

Verifica

Chiudi

Figura 5.7: Meccanismo di conferma identità dell’utente

5.5.2.1.3 Login Nella pagina, definita in codice come *LoginView*, è presente un modulo di inserimento dei dati di accesso, come visibile in figura 5.8. Al fine di minimizzare i dati raccolti, l’utente dovrà inserire solo il suo *DID*, come previsto dai casi d’uso, in quanto i dati inseriti nella fase di registrazione sono inseriti ed associati all’interno del documento presentato per l’accesso. Anche in questo caso, come per la registrazione, è previsto un controllo sulla validità dei dati inseriti, verificando se il *DID* inserito sia già esistente. In base alle specifiche dette, è previsto lo stesso meccanismo di *challenge-response* dettagliato nel paragrafo precedente da figura 5.7 per verificare la validità dei dati inseriti.

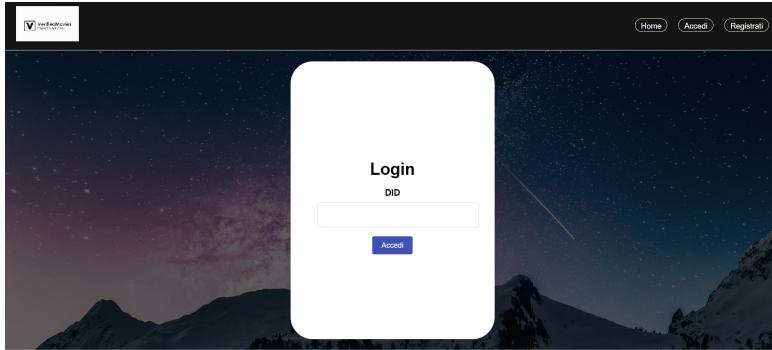


Figura 5.8: Pagina di login

5.5.2.1.4 Film In questa pagina, definita in codice come *MoviesView*, sono presenti i film disponibili all’interno del sistema, come visibile in figura 5.9. Come visibile, in questo caso, le opzioni di navigazione previste per l’utente cambiano, mostrando la pagina in oggetto, al link */Film*, e le successive pagine di questa sezione, comprendente la pagina di gestione del profilo, di visualizzazione delle prenotazioni presenti e il link per effettuare il *logout*. Per ogni film è presente un’immagine, il titolo, la descrizione, una categoria e una valutazione in termini di età. È inoltre presente il componente *SearchBox* che permette di filtrare i film in base al titolo.



Figura 5.9: Pagina dei film dell’applicazione

Ogni film permette la sua prenotazione cliccando sopra l’elemento grafico visualizzato; ogni film soggetto a limite d’età rispetto all’età calcolata dalla data di nascita dell’utente, innesta il meccanismo di verifica dell’età su cui si basa l’intera applicazione, visibile graficamente in figura 5.10. Come per la pagina home, le immagini presenti sono state prese dal noto portale di film *IMDB*, utilizzate per fini puramente dimostrativi e non commerciali.

Il meccanismo prevede l’utilizzo di specifici tipi *TypeScript* che permettono di definire il tipo di *Verifiable Credential* e *Verifiable Presentation* utilizzati, secondo la definizione presente in sezione 4.1 coerente agli standard. L’implementazione realizzata prevede come primo passaggio una funzione di creazione della *Verifiable Credential* che genera una firma digitale secondo lo standard *CLSignature2019*, con le specifiche seguenti:

- l’utilizzo della chiave privata dell’*issuer*, usato per firmare la credenziale;
- l’utilizzo della chiave pubblica, usata per verificare la firma;

Verifica la tua età per continuare

Questo film è valutato R. Per favore, dimostra la tua età per accedere al film e prenotarlo.

[Procedi](#) [Chiudi](#)

Figura 5.10: Finestra di verifica dell'età

- la generazione di un numero casuale, usato per creare la prova di correttezza della firma;
- la generazione di un valore di firma, usato per creare la prova di correttezza della firma;
- la generazione di una prova di correttezza della firma, usato per verificare la validità della firma.

Questo tipo di firma viene utilizzato in quanto permette di realizzare il meccanismo ^g[Zero Knowledge Proof](#) (basandomi sulla standardizzazione in [25]); infatti, i dati dell'utente vengono creati ed incapsulati all'interno di uno schema di appartenenza comune. La verifica della firma permette di certificare che l'utente sia in possesso di una credenziale valida, senza rivelare i dati contenuti al suo interno. Questa realizza un'implementazione che, a livello logico, riflette quanto utilizzato nella piattaforma *Hyperledger Indy*, descritto in sezione 2.3.1 (schema logico e logica applicativa seguita adattando a questo tipo di firma quanto presente in [1]).

Inoltre, secondo il meccanismo di ^g[Self Sovereign Identity](#), occorre fare riferimento a tre soggetti in questo sistema:

- l'*issuer*, che crea la credenziale e la firma. Questo viene generato in fase di accesso al sito;
- l'*holder*, che possiede la credenziale ed è l'utente in possesso del suo *DID*;
- il *verifier*, che verifica la validità della credenziale e ne estrae i dati. In questo caso è il sito stesso del cinema che verifica la validità della credenziale.

Le specifiche del flusso logico seguito fanno riferimento a in [2].

La credenziale creata internamente ha il seguente aspetto:

```

1 const vc: VCDIVerifiableCredential = {
2   '@context': ['https://www.w3.org/2018/credentials/v1'],
3   id: 'http://localhost:3000/ageCredentialSchema',
4   type: ['VerifiableCredential'],
5   credentialSchema: {
6     id: userDid ? userDid : '',
7     type: "VerifiableCredential"
8   },
9   issuer: {
10     id: issuerDid,
11     publicKey: publicKeyHex
12   } as IssuerObject,
13   issuanceDate: new Date().toISOString(),
14   credentialSubject: {
15     id: userDid,
16     age: userData.age,
17     type: 'VerifiableCredential',
18   } as CredentialSubject,
19   proof: {
20     type: "CLSignature2019",
21     issuerData: issuerDid,
22     attributes: masterSecret,
23     signature: signature,
24     signatureCorrectnessProof: proof.signatureCorrectnessProof,
```

```
25     },
26 }
```

L'utilizzo di questo tipo di firma digitale è usata in soluzioni *blockchain* di secondo livello, come *Hyperledger Indy*, secondo le specifiche dettagliate in [3].

Successivamente, viene utilizzata una funzione che prevede la creazione di una *Verifiable Presentation* che contiene la *Verifiable Credential* creata precedentemente, e avendo come parametri al firma e la prova di correttezza generata prima. L'implementazione adatta flessibilmente i campi di verifica presenti per il tipo di firma digitale, adattando al tipo di firma quanto previsto in [20], diventando come segue:

```
1 const vp: VerifiablePresentation = {
2   '@context': [
3     "https://www.w3.org/2018/credentials/v1",
4     "https://www.w3.org/2018/credentials/examples/v1"
5   ],
6   type: "VerifiablePresentation",
7   verifiableCredential: [vc],
8   proof: {
9     type: "CLSignature2019",
10    proofValue: {
11      signatureValue: {
12        r: signature.r.toString(16),
13        s: signature.s.toString(16),
14      },
15      signatureCorrectnessProof: signatureCorrectnessProof,
16    },
17  }
18};
```

Nella successiva fase, viene verificata la firma digitale immessa, utilizzando la chiave pubblica dell'*issuer*, e la prova di correttezza generata precedentemente. Tale passaggio prevede l'utilizzo di una funzione che utilizza il calcolo della curva ellittica *secp256k1*, usato all'interno della piattaforma Bitcoin, per verificare all'interno di *blockchain* la validità delle transazioni trasmesse. Questa si compone di vari passaggi:

- generazione della coppia di chiavi, pubblica e privata, generando un valore casuale intero definito come *nonce*;
- generazione di un valore casuale, generando un valore *hash* univoco ai dati trasmessi e una firma, composta da una coppia di valori *r* e *s*, dati dal valore del punto di firma e del valore *nonce* generato precedentemente;
- verifica della firma, utilizzando la chiave pubblica, la firma fornita e il valore hash per verificare se la coppia di chiavi iniziale è valida.

Come ultimo punto, viene verificata la catena di fiducia della *Verifiable Presentation* creata, utilizzando la chiave pubblica dell'*issuer* e risalendo la catena di fiducia fino alla radice, verificando la firma digitale di ogni *issuer* creato nella pagina iniziale del sito attraverso le funzioni del contratto librerie *resolve* e *resolveChain*.

Graficamente, ciascuno di questi passaggi simula una fase di caricamento, mostrando all'utente un messaggio e un'icona di caricamento, per indicare che il sistema sta eseguendo le operazioni fin qui descritte, come visibile in figura 5.11.

Verifica la tua età per continuare

In corso...



Questo film è valutato R. Per favore, dimostra la tua età per accedere al film e prenotarlo.

[Procedi](#) [Chiudi](#)

Figura 5.11: Esempio di caricamento durante la verifica dell'età

In ultimo, come descritto inizialmente in questo paragrafo, per l'utente è possibile lasciare recensioni e condividere il film in oggetto. La recensione prevede, a fini dimostrativi, l'inserimento di un testo, salvato convenzionalmente all'interno del portale in locale. L'inserimento è visibile in figura 5.12.

Scrivi una recensione

Inserisci la tua recensione...

[Invia](#) [Annulla](#)

Figura 5.12: Recensione per un film

Infine, la condivisione di un film genera un link con i metadati dello stesso e comprendente il titolo del film come parametro di riconoscimento. Questa azione è visibile in figura 5.13.

Condividi i dettagli del film

Film: Il Padrino
 Anno: 1972
 Valutazione: 9.2
 Categorie: Noir, Drammatico, Gangster
 Età: R

[Condividi](#) [Annulla](#)

Figura 5.13: Condivisione di un film

5.5.2.1.5 Prenotazione del film Per questa pagina, definita in codice come *MovieBookingView*, accessibile una volta effettuata la verifica dell'età, viene mostrata una finestra per confermare la prenotazione del film selezionato, inserendo il numero di posti desiderati, la data di prenotazione, l'orario e il numero di biglietti tra una serie di opzioni disponibili, come visibile in figura 5.14.

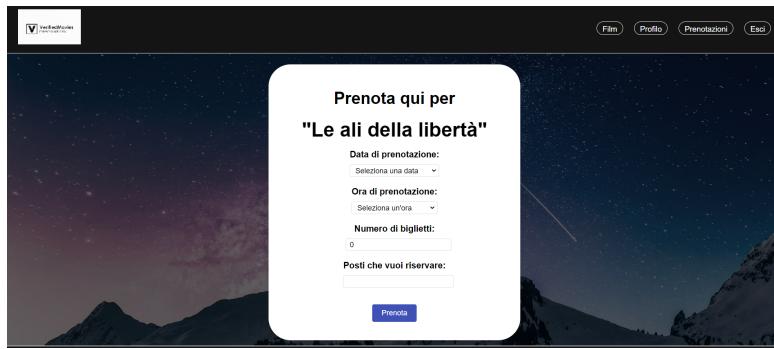


Figura 5.14: Pagina di prenotazione del film selezionato

Una volta effettuata la prenotazione, viene mostrata una finestra di conferma, come visibile in figura 5.15.

Prenotazione avvenuta con successo!

Riepilogo della prenotazione

Data: 26-05

Ora: 10:00

Numero di biglietti: 2

Posti riservati: 2

Chiudi

Tutte le prenotazioni

Figura 5.15: Finestra di conferma della prenotazione del film selezionato

5.5.2.1.6 Lista delle prenotazioni Per questa pagina, definita in codice come *BookingListView* ed accessibile una volta effettuata la verifica dell'età e la prenotazione del film, viene mostrata una lista delle prenotazioni effettuate dall'utente. I dati del film sono salvati e mostrati in seguito uno all'altro, recuperati dal salvataggio in locale all'interno di *localStorage*, come visibile in figura 5.16.

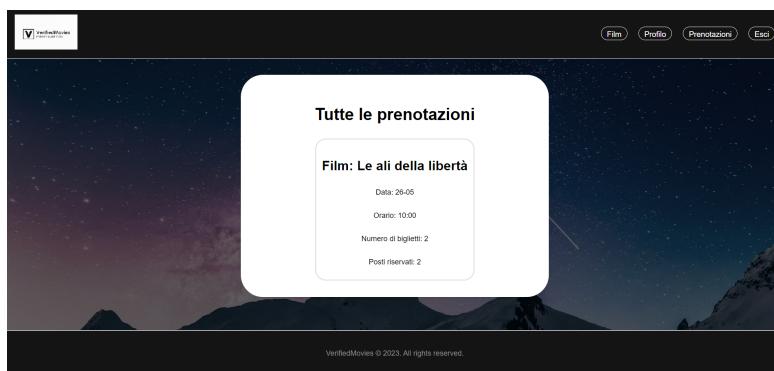


Figura 5.16: Pagina della lista delle prenotazioni effettuate

5.5.2.1.7 Profilo All'interno di questa pagina, definita in codice come *AccountView*, viene mostrato un riepilogo dei dati dell'utente, come visibile in figura 5.17. Questi dati, salvati sempre in locale all'interno di *sessionStorage*, sono recuperati e mostrati all'utente, visibili nella pagina. Questa implementazione è stata realizzata per motivi di semplicità e per evitare di dover creare un database esterno per la memorizzazione dei dati, che sarebbe stato comunque necessario per un'applicazione reale. La pagina offre la possibilità di modificare i propri dati oppure cancellare il proprio profilo, chiamando il metodo dello *smart contract deactivate* che permette di disattivare il *DID* associato all'utente, cancellando poi i suoi dati in modo corretto.

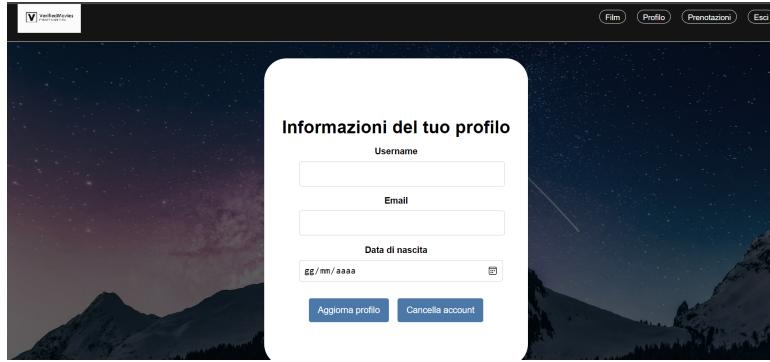


Figura 5.17: Pagina di gestione dei dati dell'utente

5.5.2.1.8 Errore 404 La pagina viene definita in codice come *ErrorView*. Se l'utente inserisse un URL non valido, verrà mostrata una pagina di errore 404, come visibile in figura 5.18. Questa contiene un link che reindirizza l'utente alla pagina principale del portale.

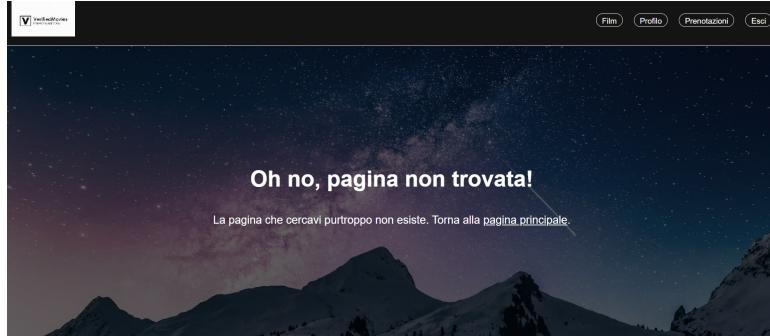


Figura 5.18: Pagina di errore 404

5.5.2.2 Componenti realizzati

5.5.2.2.1 AuthContext Il componente è stato realizzato per gestire l'autenticazione in tutta l'applicazione; infatti, viene utilizzato per salvare lo stato dell'utente, permettendo di encapsulare la logica di autenticazione e di renderla disponibile in tutti i componenti dell'applicazione. In questo modo, è possibile discriminare quali voci di menu visualizzare in base all'utente e se un determinato utente può accedere ad una determinata pagina. La navigazione è permessa tramite l'utilizzo della libreria *react-router-dom*, che permette di definire delle rotte e di reindirizzare l'utente in base alla rotta richiesta. Ciascun componente, da qui in avanti, prevede, come descritto per le pagine, l'esportazione di default come funzione *TypeScript*, includendo la sua struttura impostata in *HTML* e la presentazione impostata in *CSS* da file omonimo e presente nella stessa cartella del componente.

5.5.2.2.2 Footer Il componente è stato realizzato per gestire il footer dell'applicazione, che viene visualizzato in tutte le pagine dell'applicazione. Il footer contiene i principali dati di navigazione del sito e contiene un link che permette di ritornare su nella pagina corrente, nascosto a livello visivo, ma accessibile tramite ^g[screen reader](#).

5.5.2.2.3 NavBar Il componente è stato realizzato per gestire la barra di navigazione dell'applicazione, che viene visualizzata in tutte le pagine dell'applicazione; in questo caso, se l'utente è autenticato vengono mostrati i collegamenti relativi alla pagina principale, al *login* e alla registrazione, altrimenti vengono visualizzate le voci di menu relative alla pagina principale, alla pagina dei film, al profilo e alla lista di prenotazioni effettuate. Per poter effettuare correttamente il *logout* e riconoscere l'utente, è stato necessario utilizzare la funzione *useContext* di *React*, utilizzando degli *hooks* personalizzati per gestire la presentazione e i propri link di navigazione.

5.5.2.2.4 PrivateRoute Questo componente è stato realizzato per gestire le rotte protette dell'applicazione, che vengono utilizzate per accedere alle pagine di prenotazione e di lista delle prenotazioni. Nel caso in cui l'utente non sia autenticato e voglia accedere ad uno dei link protetti, per semplicità a fini applicativi, l'utente viene reindirizzato automaticamente alla pagina di *login*. Di fatto, gestisce con la funzione di autenticazione gli elementi presenti e rileva, in base al contesto, se l'utente sia autenticato o meno, reindirizzandolo alla pagina di *login* o alla pagina richiesta.

5.5.2.2.5 ScreenReaderHelp Il componente è stato progettato e sviluppato con l'obiettivo di fornire un supporto agli *screen reader* all'interno dell'applicazione. Esso è stato creato per garantire che i contenuti presenti sulla pagina siano accessibili agli utenti che utilizzano screen reader per navigare e comprendere il contenuto. Tramite il suo stile, è possibile nasconderne il contenuto visivamente, assicurandosi che sia accessibile solo agli screen reader.

5.5.2.2.6 SearchBox Il componente è stato realizzato per gestire la barra di ricerca dei film, che viene visualizzata nella pagina dei film. La ricerca viene effettuata in base al titolo dello stesso e, a fini di estensibilità, è possibile aggiungere altri parametri di ricerca, previsti per il tipo interno *Movie* con le caratteristiche descritte nella pagina di visualizzazione dei film presenti. La funzionalità di ricerca è resa possibile grazie all'uso di un semplice *hook*, che permette di gestire lo stato della ricerca e di aggiornarlo in base al testo inserito dall'utente.

5.5.2.2.7 Notification Tale componente è stato realizzato per gestire la visualizzazione degli errori e dei relativi messaggi nelle varie situazioni, come ad esempio la registrazione di un utente già esistente oppure la mancata compilazione di campi all'interno dei moduli del sito. In questo caso, viene utilizzato un *hook* per gestire lo stato della notifica e per aggiornarlo in base al messaggio di errore ricevuto, mostrando poi il messaggio all'utente.

Capitolo 6

Verifica e validazione

In questo capitolo verranno riportate le attività di verifica e validazione svolte durante lo sviluppo del progetto, dettagliando i test svolti a tutti i livelli, dai test di unità fino alla fase di collaudo ed accettazione del prodotto

6.1 Accessibilità

Al fine di rendere il sito più accessibile possibile, si è deciso di seguire le linee guida fornite da [W3C](#), in particolare quelle riguardanti l'accessibilità dei contenuti web (WCAG) ([23]), mantenendo il livello AA. Questo ha comportato l'utilizzo di attributi *alt* per le immagini di contenuto (lasciandolo vuoto per le immagini di presentazione) e l'utilizzo di attributi HTML semanticici per la struttura del sito, come ad esempio *<header>*, *<main>*, *<footer>* e *<nav>* per gli elementi di navigazione, *label* negli elementi dei form, *<article>* per i contenuti principali della pagina. Si riporta inoltre l'utilizzo degli attributi *lang* per specificare la lingua del contenuto qualora fosse diversa dall'italiano.

Come descritto per la parte dei componenti, è stato creato un componente apposito per aiuto agli [screen reader](#), in grado di fornire un *link* per saltare la navigazione e accedere direttamente al contenuto principale della pagina oppure l'opzione per tornare su in pagine grandi. Il sito utilizza un layout a tre pannelli, cercando di rendere fluida l'esperienza di navigazione, evitando lo scorrimento laterale nelle condizioni di visualizzazione più comuni. La struttura cerca di seguire un livello di intestazioni logiche, utilizzando *<h1>* per il titolo principale della pagina, *<h2>* per i titoli di sezione e *<h3>* per i titoli di sottosezione, tale da non saltare livelli di intestazione e consentire una navigazione più fluida nelle varie condizioni. Il carattere utilizzato è *Calibri*, un font *sans-serif* che risulta essere mediamente ben leggibile, adottando convenzionalmente un'interlinea di 1.5 *em* per migliorare la leggibilità del testo.

In merito ai contrasti utilizzati, le linee guida richiedono un rapporto di contrasto almeno pari a 4.5:1 per il testo normale e 3:1 per il testo grande, con alcune eccezioni per il testo in grassetto e per il testo non essenziale. Con l'aiuto del sito *WCAG Contrast Checker* ([22]) si è cercato di mantenere un livello di contrasto adeguato nelle varie situazioni, considerando lo sfondo, il testo e lo stato dei link di navigazione (visitato/non visitato) e dei bottoni (*hover/focus*).

In dettaglio, possiamo precisare:

- per i contrasti nella pagina principale e per il testo nelle varie pagine, un colore con sfumatura `#fffff80`, correttamente visibile rispetto all'immagine di sfondo;
- per i link di navigazione, un colore con sfumatura `#fffff`, per lo stato di hover pari a `#ff8a8a`, lo stato di visitato con sfumatura `#ffec80`, tutti pienamente contrastanti con la barra di navigazione di colore `#141414`, usato qui e nel *footer*;
- per tutti i bottoni, il colore `#4c78a8`, per lo stato di hover il colore `#303f9f` e per lo stato di focus il colore `#3f51b580`, contrastanti con lo sfondo `#fffff` delle finestre in cui sono usati.

Le convenzioni interne del sito, a livello grafico e di navigazione, sono state mantenute coerenti in tutte le pagine, cercando di rendere l'esperienza di navigazione più fluida possibile. Ogni azione nel sito ha un apposito messaggio di *feedforward* che indirizza l'utente sulle azioni che può compiere e sulle conseguenze che queste hanno, ricevendo un apposito *feedback*. Nella pagina con più contenuti è stata offerta un'opzione di ricerca per trovare più facilmente i contenuti desiderati, con un apposito messaggio di errore nel caso in cui non vengano trovati risultati. A questo proposito, è stato utilizzato un componente apposito per la gestione degli errori, che permette di visualizzare un messaggio di errore e di tornare alla pagina precedente.

Le immagini utilizzate sono state ottimizzate in dimensione e peso, cercando di mantenere un livello di qualità adeguato, senza appesantire troppo il sito e il suo caricamento. Le stesse ancora di navigazione hanno nomi esplicativi e sono state posizionate in modo da essere facilmente raggiungibili e cliccabili dall'apposita barra di navigazione. Inoltre, si è deciso di evitare l'uso di tabelle, che possono risultare poco accessibili, e di utilizzare un layout a griglia per la disposizione dei contenuti grafici, offrendo come descritto in sezione 5.2.1 dei punti di rottura per ogni pagina per garantire la fruizione del sito anche su dispositivi mobili. Un controllo generale di quanto descritto è stato effettuato con l'aiuto dell'estensione *WAVE* ([21]), che ha permesso di individuare alcuni errori e avvisi, corretti e poi modificando di conseguenza, portando il sito ad essere conforme alle linee guida definite.

6.2 Test di unità

Qui vengono descritti i test di unità effettuati sul codice, utilizzando il *framework Jest* ([11]), che permette di effettuare test di unità su codice *JavaScript* e *TypeScript* e la libreria di test *React Testing Library* ([14]), che permette di effettuare test di unità su codice *React*. Il codice identificativo dei test è strutturato da:

TU[Numer]

avendo come legenda:

- **TU**, cioè il test di unità, in grado di testare una singola unità di codice e di verificare che il suo comportamento sia corretto;
- **Numer**, come numero identificativo univoco e progressivo del test in questione.

Ciascun test riporta le seguenti sigle:

- **I**: indica che il test è stato implementato e superato;
- **NI**: indica che il test non è stato implementato.

Per una migliore comprensione del riferimento alle pagine e alle relativi componenti, con denominazione di riferimento in codifica, si rimanda alla sezione 5.5.2.

Tabella 6.1: Tabella di tracciamento dei test di unità

ID	Elemento	Descrizione	Stato
TU1	AuthContext	Verificare se il componente AuthContext gestisce correttamente l'autenticazione, salvando lo stato dell'utente e fornendo la logica di autenticazione.	I
TU2	Footer	Verificare se il componente Footer viene visualizzato e caricato correttamente in tutte le pagine dell'applicazione e se il link di ritorno funziona con <i>screen reader</i> .	I

Continuazione della tabella 6.1			
ID	Elemento	Descrizione	Stato
TU3	NavBar	Verificare se il componente NavBar viene visualizzato e caricato correttamente in tutte le pagine dell'applicazione	I
TU4	NavBar	Verificare se il componente NavBar permette la visualizzazione dei collegamenti corretti in base allo stato di autenticazione dell'utente	I
TU5	PrivateRoute	Verificare se il componente PrivateRoute gestisce correttamente le rotte protette dell'applicazione, reindirizzando l'utente alla pagina di login se non è autenticato	I
TU6	ScreenReaderHelp	Verificare se il componente ScreenReaderHelp è progettato correttamente per fornire supporto agli <i>screen reader</i> , rendendo il contenuto accessibile solo agli <i>screen reader</i>	I
TU7	SearchBox	Verificare se il componente SearchBox gestisce correttamente la barra di ricerca dei film, effettuando la ricerca in base al titolo e aggiornando correttamente lo stato della ricerca	I
TU8	Notification	Verificare se il componente Notification gestisce correttamente la visualizzazione degli errori e dei messaggi di notifica nelle diverse situazioni	I

6.3 Test di integrazione

In questa sezione sono introdotti i test di integrazione, cioè la verifica del corretto funzionamento di più componenti del sistema, in grado di interagire tra loro. Questo, in particolare, implica l'analisi del comportamento delle pagine e dei componenti grafici da cui sono composte.

Il codice identificativo dei test è strutturato da:

TI[Numero]

avendo come legenda:

- **TI**, cioè il singolo test di integrazione;
- **Numero**, come numero identificativo univoco e progressivo del test in questione.

Tabella 6.4: Tabella del tracciamento dei test di integrazione

ID	Elemento	Descrizione	Stato
TI1	HomePage	La pagina viene testata per verificarne il caricamento grafico e delle sue componenti	I
TI1	HomePage	La pagina viene testata per verificare se le funzioni di creazione del <i>DID</i> e della catena degli <i>issuer</i> sono chiamate correttamente all'avvio dell'applicazione	I

Continuazione della tabella 6.4			
ID	Elemento	Descrizione	Stato
TI3	RegisterView	La vista di registrazione viene testata per verificare che venga visualizzata correttamente	I
TI4	RegisterView	La vista di registrazione viene testata per verificare che venga gestito correttamente il processo di registrazione tramite il meccanismo <i>challenge-response</i>	I
TI5	RegisterView	La vista di registrazione viene testata per verificare che vengano gestiti correttamente i controlli di validazione dei campi e le notifiche di successo o di errore in caso di <i>email</i> o <i>DID</i> già registrati	I
TI6	LoginView	La vista di <i>login</i> viene testata per verificare che venga visualizzata correttamente	I
TI7	LoginView	La vista di <i>login</i> viene testata per verificare che venga gestito correttamente il processo di accesso tramite <i>DID</i>	I
TI8	LoginView	La vista di <i>login</i> viene testata per verificare che vengano gestiti correttamente i controlli di validazione dei campi e le notifiche di errore in caso di <i>DID</i> non esistente	I
TI9	LoginView	La vista di <i>login</i> viene testata per verificare che venga aperta correttamente la finestra modale di verifica dell'identità in caso di <i>DID</i> esistente	I
TI10	LoginView	La vista di <i>login</i> viene testata per verificare che venga gestito correttamente il processo di verifica dell'identità	I
TI11	MoviesView	La vista dei film viene testata per verificarne il caricamento grafico e delle sue componenti	I
TI12	MoviesView	La vista dei film viene testata per verificare l'apertura di un film e la visualizzazione dei dettagli corretti	I
TI13	MoviesView	La vista dei film viene testata per verificare l'apertura della finestra di verifica dell'età e il corretto funzionamento del processo di verifica	I
TI14	MoviesView	La vista dei film viene testata per verificare la possibilità di condividere un film e l'apertura della finestra di condivisione	I
TI15	MoviesView	La vista dei film viene testata per verificare la visualizzazione di un messaggio quando non ci sono film disponibili	I

Continuazione della tabella 6.4			
ID	Elemento	Descrizione	Stato
TI16	MoviesView	La vista dei film viene testata per verificare la visualizzazione del profilo utente se autenticato	I
TI17	MovieBookingView	La vista di prenotazione dei film viene testata per verificare che venga visualizzata correttamente	I
TI18	MovieBookingView	La vista di prenotazione dei film viene testata per verificare che vengano inizializzati correttamente gli stati di prenotazione	I
TI19	MovieBookingView	La vista di prenotazione dei film viene testata per verificare che venga gestito correttamente il processo di prenotazione	I
TI20	MovieBookingView	La vista di prenotazione dei film viene testata per verificare che venga gestito il processo di visualizzazione del riepilogo della prenotazione in modo corretto	I
TI21	MovieBookingView	La vista viene testata per verificare che venga gestito correttamente il processo di chiusura del modale di conferma prenotazione	I
TI22	MovieBookingView	La vista viene testata per verificare che venga gestito correttamente il reindirizzamento alla pagina delle prenotazioni	I
TI23	MovieBookingView	La vista viene testata per verificare che venga gestito il processo di validazione della prenotazione	I
TI24	BookingListView	La vista di visualizzazione delle prenotazioni viene testata per verificare che venga renderizzata correttamente	I
TI25	BookingListView	La vista di visualizzazione delle prenotazioni viene testata per verificare che venga mostrato il messaggio corretto quando non ci sono prenotazioni	I
TI26	BookingListView	La vista di visualizzazione delle prenotazioni viene testata per verificare che vengano mostrate correttamente le prenotazioni presenti	I
TI27	ErrorView	La vista di errore viene testata per verificare che venga renderizzata correttamente, riportando l'utente alla pagina principale	I

6.4 Test di regressione

Qui sono introdotti i test di regressione, cioè la verifica del corretto funzionamento del sistema dopo l'introduzione di modifiche, verificando di non aver introdotto errori in parti del sistema che precedentemente funzionavano correttamente.

Il codice identificativo dei test è strutturato da:

TR[Numero]

avendo come legenda:

- **TR**, cioè il singolo test di regressione;
- **Numero**, come numero identificativo univoco e progressivo del test in questione.

Tabella 6.8: Tabella del tracciamento dei test di regressione

ID	Elemento	Descrizione	Stato
TR1	Applicazione	Si testa che l'applicazione venga correttamente caricata e visualizzata nelle sue funzionalità sul browser <i>Google Chrome</i>	I
TR2	Applicazione	Si testa che l'applicazione venga correttamente caricata e visualizzata nelle sue funzionalità sul browser <i>Microsoft Edge</i>	I
TR3	Applicazione	Si testa che l'applicazione venga correttamente caricata e visualizzata nelle sue funzionalità sul browser <i>Safari</i>	I
TR4	Applicazione	Si testa che l'applicazione venga correttamente caricata e visualizzata nelle sue funzionalità sul browser <i>Mozilla Firefox</i>	I
TR5	Applicazione	Si testa che l'applicazione venga correttamente caricata e visualizzata nelle sue funzionalità sul browser <i>Opera</i>	I
TR6	MovieBookingView	Si verifica che le modifiche apportate alla vista di prenotazione non abbiano causato malfunzionamenti	I
TR7	MoviesView	Si verifica che eventuali modifiche apportate al meccanismo di verifica della pagina non causino errori o malfunzionamenti compromettendo l'intera logica applicativa	I
TR8	Applicazione	Si verifica che se vengono implementate modifiche al sistema di gestione di dati, l'applicazione sia comunque in grado di recuperarli correttamente	I

6.5 Accettazione e collaudo

Settimanalmente, al fine di conoscere lo stato di avanzamento del prodotto, veniva organizzato un incontro interno con il proponente e gli altri stagisti per discutere i progressi raggiunti e discutere eventuali dubbi o problemi riscontrati ad alto livello. Ciò ha permesso di monitorare il lavoro e di conoscere lo stato dell'implementazione realizzata, garantendo così una maggiore trasparenza e una migliore comunicazione tra le parti, permettendo di avere un *feedback* indicativo sulle attività da svolgere. Alcuni chiarimenti si sono avuti con incontri autonomi con lo stagista magistrale Alessio De Biasi, di cui utilizzo il contratto come codice libreria per il funzionamento dell'applicazione.

Durante l'ultima settimana di stage, il proponente ha richiesto di effettuare un incontro per discutere del prodotto realizzato e per mostrare l'applicazione completa e funzionante, dimostrando il raggiungimento degli obiettivi prefissati e delle funzionalità richieste, partendo dalle pagine ad alto livello e discutendo dei test e dei tipi di firma digitale e gli standard da me implementati.

Capitolo 7

Conclusioni

In questo capitolo verranno riportate le conclusioni del lavoro svolto, analizzando i risultati ottenuti e le conoscenze acquisite, dando una valutazione personale del lavoro svolto.

7.1 Obiettivi raggiunti e consuntivo finale

In merito al raggiungimento degli obiettivi prefissati per il tirocinio (in sezione 3.3), si può confermare la soddisfazione di ciascuno. In particolare prima con lo studio e poi con la realizzazione dell'intero progetto, i concetti di ^gblockchain e di ^gsmart contract sono stati acquisiti e compresi, dimostrando con la pratica la comprensione del L'identità sovrana di ^gSelf Sovereign Identity è stato applicato ed implementato in un contesto reale, con la possibilità di interagire con un'applicazione web e con un'applicazione mobile, dimostrando la sua applicabilità in diversi ambiti. Questo ha permesso, tramite lo studio e l'implementazione autonoma descritta di standard di identità e di firma digitale, di acquisire conoscenze e competenze in ambiti di sicurezza informatica e di crittografia, che sono stati approfonditi e studiati in modo mirato nel progetto realizzato, garantendo la copertura dei requisiti obbligatori.

Come richiesto dalla sezione degli obiettivi desiderabili presenti nella sezione citata, l'applicazione implementa la libreria ^gWeb3.js per l'interazione con la *blockchain*, permettendo di interagire con lo *smart contract* in modo corretto. Inoltre, l'applicazione è stata completamente testata, come riportato dal capitolo 6. La stessa creazione dell'applicazione permette di esplorare uno scenario di applicazione dell'ambito della *blockchain* e dell'identità connessa senza trasmettere informazioni personali, risultando così in un'applicazione che rispetta la *privacy* degli utenti.

7.2 Conoscenze acquisite e analisi del lavoro svolto

Il tirocinio svolto ha soddisfatto delle aspettative principalmente da un punto di vista conoscitivo, dato che ho capito le implicazioni del mondo *blockchain* e come questo rappresenti, se ben applicato, un passo importante da un punto di vista di sicurezza. Il punto più rilevante di questo tirocinio è stata certamente l'implementazione di questo progetto sfruttando un codice di un laureando magistrale, ampliando questa applicazione a degli standard ^gW3C di non poca importanza, come ^gDecentralized Identifier e ^gVerifiable Credentials, sviluppando una parte non completamente normata come ^gZero Knowledge Proof.

L'attività è stata molto impegnativa e, nonostante il supporto di massima presente, la parte analitica e di effettivo sviluppo del progetto, specie della parte effettivamente più difficile come *Zero Knowledge Proof* sono risultate estremamente teoriche e senza un vero supporto, se non da un punto di vista di puro ragionamento logico, da parte del laureando magistrale Alessio De Biasi, di cui ho dovuto adattare il codice per il mio progetto. Le scelte implementative e la loro realizzazione sono state complesse da gestire e da capire in autonomia, ma sono state comunque affrontate con successo, permettendo di acquisire conoscenze e competenze in ambiti di sicurezza informatica e di crittografia decisamente importanti per un percorso di Laurea Triennale come il mio.

Di massima, è possibile descrivere le principali conoscenze maturate nell'ambito progettuale:

- **comprendere dell'ambito blockchain e sviluppo di smart contract:** l'interazione di uno *smart contract* e la sua interazione con un'applicazione web e mobile, permettono di comprendere come implementare una struttura dati pubblica ed immutabile, cambiando l'idea stessa di programmazione classica e facendo ben comprendere come ogni azione fatta in codice possa avere potenziali implicazioni di sicurezza. Ogni transazione è infatti pubblica e richiede una vera comprensione della struttura sottostante e di come scrivere il codice in modo sicuro e corretto, per evitare perdite di dati o di denaro. In questo ambito, utile certamente l'apprendimento autonomo del linguaggio ^a*Solidity* e della libreria *web3.js*, che permettono di interagire con la *blockchain* in modo semplice e adattando tale logica a quella di un'applicazione realmente utilizzabile;
- **analisi di standard di identità digitale e di protocolli di firma digitale:** l'implementazione di *Decentralized Identifiers (DID)* e *Verifiable Credentials (VC)* ha permesso di comprendere come sia possibile implementare un sistema di identità digitale partendo da implementazioni definite e preesistenti, tuttora sporadicamente applicate, ma con un ambito di applicazione molto ampio. Di fatto, in un mondo come quello odierno dove la *privacy* rappresenta una giusta preoccupazione per gli utenti medi e per la stessa informatica, tale progetto dimostra come, in modo relativamente semplice, sia possibile certificare che l'utente sia chi dice di essere, senza trasmettere informazioni personali, ma solo una firma digitale, che può essere verificata da chiunque sul meccanismo di catena di fiducia creato e descritto. Il meccanismo di riconoscimento dell'autenticità delle informazioni trasmesse dall'utente ha richiesto una creazione personalizzata di quanto esistente in ambito sicurezza e *blockchain*, creando un meccanismo conforme agli standard e di facile utilizzo, data la complessità degli argomenti descritti e da me trattati, che richiedono una conoscenza approfondita di basi crittografiche, di sicurezza e di come implementare questi standard in modo corretto e concreto;
- **studio e analisi di Zero Knowledge Proof:** la realizzazione e l'applicazione di *Zero Knowledge Proof* è stata una delle parti più complesse da implementare, data la natura estremamente teorica della stessa e della sua successiva applicazione, che richiede necessariamente, dati gli standard precedentemente descritti, la comprensione di standard di firma digitale spesso non accuratamente documentati e che richiedono di studiare a fondo gli standard di riferimento e una ricerca in gran parte teorica, ben maggiore di quella prevista da alcuni ambiti di studio, dato il tempo relativamente limitato di effettiva implementazione conforme agli standard.
- **nuove competenze di programmazione e progettuali:** l'analisi e la progettazione dei dettagli di un'applicazione web e mobile, con l'implementazione di un ^a*back-end* e di un ^a*front-end* hanno permesso di sviluppare ulteriormente le mie conoscenze apprese durante il corso di Ingegneria del Software, normando in modo più efficace le attività di codifica e di realizzazione della documentazione, risolvendo sul campo problemi progettuali ma anche concettuali presenti e sviluppando modifiche in modo agile, separando le responsabilità delle componenti e delle pagine, migliorando così la manutenibilità e la comprensione del codice. L'applicazione del linguaggio di programmazione *TypeScript* unito alla comprensione di vulnerabilità legate al linguaggio *Solidity* risulta utile nell'ulteriore comprensione e applicazione da un punto di vista di sviluppo di applicativi web e mobile, analizzando in dettaglio le caratteristiche offerte da questi linguaggi e librerie e strutturando il codice attraverso *design pattern* di riferimento e architet-

ture utili da un punto di vista logico e di sviluppo attraverso la *Continuous Integration* e la *Continuous Delivery* ([4]), così implementando modifiche in modo continuativo e strutturato, a calendario e nel corso dell'intero progetto.

7.3 Possibili sviluppi futuri e scenari di applicabilità

7.4 Valutazione personale

L'attività di stage si è rivelata molto utile per la crescita personale e professionale, permettendo di acquisire nuove conoscenze e competenze in ambiti di sicurezza informatica e di crittografia, che sono stati approfonditi e studiati in modo mirato nel progetto realizzato. Inoltre, l'ambito *blockchain* è stato studiato in modo approfondito, permettendo di esplorare in autonomia un campo con ripercussioni decisamente interessanti nel mondo dell'informatica, decisamente non visti né praticati in corsi universitari ed è stata un'esperienza assolutamente rilevante. Ho potuto in questo ambito confrontarmi su aspetti interessanti che offrono nuove prospettive in ambito di sicurezza e che danno una visione delle tecnologie informatiche più consapevole.

Spesso ambiti sconosciuti possono essere rilevanti da un punto di vista professionale, e questo è stato un esempio di come andare oltre i preconcetti dati da un ambito ampiamente teorico e non ormato, si rivelò in realtà un'occasione di crescita e di apprendimento, richiedendo necessariamente di praticare a fondo uno studio mirato delle attività svolte. In questi casi, la passione per il conoscere guida e anticipa la necessità di apprendere, dando un'importante possibilità di conoscere e maturare, altrimenti non offerta in modo così vicino alle realtà aziendali e future rimanendo in ambito puramente accademico. L'attività di supporto è comunque stata presente di massima, dando ampia possibilità di sviluppo e organizzazione in modo autonomo, rimanendo vicino a me come studente e alle mie necessità.

Il progetto per me rappresenta una crescita che matura quanto compreso e visto in questi ultimi anni, permettendo di applicare in modo nuovo concetti per la maggior parte sconosciuti e con tecnologie nuove, calandomi in un contesto attuale e futuro, usando delle librerie specifiche e sviluppando un pensiero critico nell'analisi, affinando ulteriormente una visione d'insieme dei corsi fin qui frequentati nel mio percorso di Laurea Triennale, perfezionando il mio metodo di studio ed il mio modo di affrontare un progetto di codifica con ripercussioni attuali e future in modo così intenso, completando quanto realizzato nel corso di Ingegneria del Software e usando tali metodologie per affrontare e suddividere il lavoro in modo autonomo, partendo da quanto appreso in questi mesi. Quanto studiato rappresenta un importante partenza per il mio futuro percorso di Laurea Magistrale, adattando già ora un codice oggetto di tesi da parte di un laureando magistrale, ampliando delle competenze trasversali certamente utili per il mio futuro, accademico e non, ed aggiornandomi su un ambito decisamente attuale e di sicuro interesse per possibili sviluppi futuri.

Glossario

Agile In ingegneria del software, con il termine *Agile* si indica un insieme di metodi di sviluppo del software emersi a partire dai primi anni 2000 e fondati su un insieme di principi comuni, direttamente o indirettamente derivati dai principi del *Manifesto per lo sviluppo agile del software* (ing. *Manifesto for Agile Software Development*). L'approccio di sviluppo viene definito iterativo e incrementale, prevedendo la suddivisione del progetto in piccole attività chiamate *sprint*, della durata di una o due settimane, al termine delle quali viene presentato un incremento del prodotto finale. Questa si basa su un'interazione costante e flessibile, basato più sugli individui e le interazioni che sugli strumenti e i processi. [2](#), [73](#)

Application Program Interface In informatica con il termine *Application Programming Interface API* (ing. interfaccia di programmazione di un'applicazione) si indica ogni insieme di procedure disponibili al programmatore, che consente a diverse applicazioni di avere un set definito di regole e protocolli comuni per l'interazione e lo scambio di dati. Queste possono essere utilizzate per semplificare l'integrazione complessiva, permettendo un'astrazione dei livelli sottostanti. [12](#), [73](#)

back-end In informatica con il termine *back-end* si indica la parte di un'applicazione che interagisce con il database, fornendo un'interfaccia per la gestione dei dati e delle informazioni, permettendo di fornire le effettive interazioni con l'utente finale. [43](#), [70](#), [73](#)

Blockchain In informatica con il termine *Blockchain* si indica una struttura dati condivisa e immutabile. La blockchain è resa immutabile dall'utilizzo di funzioni crittografiche di *hash* e dalla struttura dati a blocchi concatenati. ed è condivisa in quanto è distribuita in una rete *peer-to-peer*. Essa è costituita da una catena di blocchi, in cui ciascuno contiene un insieme di transazioni o di informazioni, che viene validato e aggiunto alla catena da un processo di consenso tra i nodi della rete. [5](#), [19](#), [21](#), [45](#), [69](#), [73](#)

Decentralized Application In informatica con il termine *Decentralized Application (DApp)* (ing. applicazione decentralizzata) si indica un'applicazione che utilizza una *blockchain* per la gestione dei dati e delle transazioni. Queste applicazioni sono eseguite in una rete peer-to-peer, senza un'autorità centrale che ne controlla il funzionamento, ma sono gestite da tutti i nodi della rete. [21](#)

Decentralized Identifier In informatica con il termine *Decentralized Identifier (DID)* (ing. identificatore decentralizzato) si indica un identificatore univoco, che può essere utilizzato per identificare un'entità digitale, come una persona, un'organizzazione o un dispositivo. Esso è composto da un prefisso che identifica la rete in cui è stato creato, seguito da un identificatore univoco generato dall'utente. Questo è basato su tecnologie decentralizzate e distribuite come *blockchain*, sviluppato come parte dello standard *W3C Verifiable Credentials*, al fine di avere un insieme di tecnologie per invio e verifica di credenziali digitali verificabili e sicure. [15](#), [19](#), [23](#), [24](#), [43](#), [47](#), [49](#), [69](#), [73](#)

Ethereum In informatica con il termine *Ethereum* si indica una piattaforma decentralizzata per la creazione e pubblicazione di applicazioni decentralizzate. La piattaforma è basata su una *blockchain* pubblica e permette di creare applicazioni decentralizzate che possono essere eseguite in una macchina virtuale chiamata *Ethereum Virtual Machine (EVM)*, utilizzando il linguaggio di programmazione *Solidity* basato su *smart contract*, che consentono di automatizzare e garantire la sicurezza delle informazioni trasmesse in modo tracciato e immutabile. [11](#), [21](#), [45](#), [73](#)

Ethers.js In informatica con il termine *Ethers.js* si indica una libreria *JavaScript* per interagire con la *block-chain* di *Ethereum*. Questa libreria fornisce un’interfaccia semplificata e sicura per la gestione di contratti intelligenti e delle loro transazioni secondo le tecnologie di crittografia asimmetrica e la comunicazione tra i nodi della rete. [21](#), [74](#)

framework In informatica con il termine *framework* si indica un’architettura logica di supporto su cui un software può essere progettato e realizzato, facilitandone lo sviluppo da parte del programmatore. Questo permette di avere un’astrazione del codice, permettendo di concentrarsi sulla logica dell’applicazione, senza dover gestire le parti più complesse. [16](#), [20](#), [43](#), [62](#), [74](#)

front-end In informatica con il termine *front-end* si indica la parte di un’applicazione che interagisce direttamente con l’utente, fornendo un’interfaccia grafica per l’interazione con l’applicazione stessa oppure con sistemi esterni in grado di produrre input. [43](#), [70](#), [74](#)

Git *Git* è un software di controllo versione distribuito utilizzabile da interfaccia a riga di comando, creato da Linus Torvalds nel 2005. Lo scopo di Git è quello di gestire progetti con velocità e semplicità, garantendo allo stesso tempo la possibilità di gestire flussi di lavoro complessi sulla base di un sistema di controllo di versione non lineare e distribuito. Git permette di tenere traccia di tutte le modifiche apportate al codice sorgente di un progetto sviluppato da più persone e di coordinarle. [3](#), [74](#)

Integrated Development Environment In informatica con il termine *Integrated Development Environment* (ing. ambiente di sviluppo integrato) si indica un software che, in fase di programmazione, supporta i programmatore nello sviluppo del codice sorgente di un programma. Solitamente un *IDE* è composto da un editor di codice sorgente, un compilatore ed un debugger. Inoltre, spesso, fornisce strumenti per l’automazione di alcune operazioni ripetitive, per la navigazione all’interno del codice e per semplificare alcune operazioni di sviluppo. [3](#), [74](#)

JSON In informatica con il termine *JSON* si indica un formato di testo per la memorizzazione e lo scambio di dati, basato sul linguaggio di programmazione *JavaScript*. Questo formato è basato su coppie chiave-valore, in cui la chiave è una stringa e il valore può essere un oggetto, un array, un numero, una stringa, un booleano o nullo. Tale formato si adatta bene alla memorizzazione di dati strutturati, come ad esempio le informazioni contenute in un database. [23](#), [45](#), [74](#)

open source In informatica con il termine *open source* si indica un software di cui viene reso pubblico il codice sorgente, permettendo a chiunque di poterlo utilizzare, studiare, modificare e distribuire liberamente. [5](#), [43](#), [74](#)

repository In informatica con il termine *repository* si indica un ambiente di un sistema informativo, in cui vengono gestiti i metadati, attraverso tabelle relazionali, e i file, attraverso un file system gerarchico. [44](#), [74](#)

screen reader In informatica con il termine *screen reader* si indica un software che permette di convertire il testo presente sullo schermo in un formato audio, permettendo di usufruire del contenuto presentato attraverso sintesi vocale. [59](#), [61](#), [74](#)

Scrum In ingegneria del software, *Scrum* è una metodologia di sviluppo iterativa ed incrementale per la gestione del ciclo di sviluppo del software, iterativa in quanto il lavoro viene suddiviso in blocchi (*sprint*) e incrementale perché il lavoro viene suddiviso in parti che vengono consegnate in modo incrementale. In particolare, ciascuna attività prevede un coinvolgimento delle parti molto spesso, definendo periodi di retrospettiva e di confronto, per valutare il lavoro svolto e le eventuali modifiche da apportare. Il termine *scrum* deriva dal rugby, dove indica una formazione composta dalla linea dei giocatori che si fronteggiano e si accaparrano il pallone con le gambe, cercando di spingerlo verso la meta avversaria. [2](#), [21](#), [74](#)

smart contract In informatica con il termine *Smart Contract* (ing. contratto intelligente) si indica un programma informatico che permette di automatizzare e garantire la sicurezza delle informazioni trasmesse in modo tracciato e immutabile. Questi sono eseguiti in una *blockchain*, in particolare in quella di *Ethereum*, e sono scritti in un linguaggio di programmazione ad alto livello chiamato *Solidity*. [21](#), [23](#), [44](#), [69](#), [75](#)

Software Insieme di programmi, dati e documentazione che compongono un sistema o un progetto informatico. Esso viene progettato, sviluppato e mantenuto da un gruppo di persone, chiamate sviluppatori, al fine di soddisfare le esigenze del prodotto portando una soluzione ad uno specifico problema nel dominio applicativo di interesse. Questo può essere sviluppato seguendo varie metodologie, tra cui ad esempio *Agile* o *Scrum*. [2](#), [75](#)

Solidity In informatica con il termine *Solidity* si indica un linguaggio di programmazione ad alto livello per la creazione di *smart contract*. Esso ha una sintassi simile ai linguaggi *C++*, *Java* e *JavaScript*, progettato per fornire un livello di sicurezza e affidabilità elevato per la gestione ed esecuzione dei contratti intelligenti sulla *blockchain* di *Ethereum*. Esso supporta diverse funzionalità, tra cui la definizione di variabili, l'implementazione di funzioni, la definizione di strutture dati e la gestione degli eventi. [21](#), [23](#), [70](#), [75](#)

Self Sovereign Identity In informatica con il termine *Self-Sovereign Identity (SSI)* (ing. identità autonoma) modello di identità digitale basato sulla proprietà e la gestione dei dati personali da parte dell'utente. In un sistema di identità digitale *SSI*, l'utente ha il pieno controllo dei propri dati personali e decide quali informazioni condividere e con quali soggetti. L'obiettivo è quello di ridurre la dipendenza da terze parti per la gestione delle identità digitali, aumentando la sicurezza e la *privacy* degli utenti. L'*SSI* si basa su tecnologie come la *blockchain* e la crittografia, che consentono di creare registri distribuiti e sicuri delle informazioni personali, conservate in modo certificato ed immutabile. [15](#), [19](#), [21](#), [23](#), [43](#), [49](#), [54](#), [69](#), [75](#)

Unified Modeling Language In ingegneria del software *UML*, *Unified Modeling Language* (ing. linguaggio di modellazione unificato) è un linguaggio di modellazione utilizzato per descrivere e progettare sistemi complessi. Questo fornisce un insieme di notazioni grafiche atte a definire il sistema e i suoi comportamenti, comprendendo ogni parte di interazione e comunicazione tra le varie parti. Esso viene utilizzato nelle fasi di analisi dei requisiti, progettazione, implementazione e test del prodotto software. Esso è normalmente composto da diversi tipi di diagrammi con diversi significati, quali diagrammi delle classi o dei casi d'uso. [3](#), [26](#), [75](#)

URI In informatica con il termine *URI* si indica una sequenza di caratteri che identifica in modo univoco una risorsa. Questo formato è utilizzato per identificare le risorse in Internet, come ad esempio i documenti, le immagini, i video, i servizi, le risorse di rete e altro ancora. [25](#), [75](#)

Verifiable Credentials In informatica con il termine *Verifiable Credential (VC)* (ing. credenziale verificabile) si indica un documento digitale che contiene informazioni relative ad un'entità digitale, come una persona, un'organizzazione o un dispositivo. Una *VC* è un documento firmato digitalmente da un'autorità che ne certifica l'autenticità e che può essere verificato da terze parti, contenendo le informazioni principali di una persona come nome, cognome, data di nascita, luogo di nascita, ecc. Anche questo è basato su tecnologie decentralizzate e distribuite come *blockchain*, parte dello standard *W3C* omonimo. [15](#), [19](#), [24](#), [69](#), [75](#)

Verifiable Presentation In informatica con il termine *Verifiable Presentation (VP)* (ing. presentazione verificabile) si indica un documento digitale che contiene informazioni relative ad un'entità digitale, come una persona, un'organizzazione o un dispositivo. Una *VP* è un documento firmato digitalmente da un'autorità che ne certifica l'autenticità e che può essere verificato da terze parti, ciascuna contenente una serie di credenziali con un verificatore specifico, in grado di evidenziare la paternità dei dati dopo un processo di verifica, non contenenti informazioni personali, ma solo un identificatore univoco. Anche questo è basato su tecnologie decentralizzate e distribuite come *Ethereum*. [24](#), [73](#), [75](#)

World Wide Web Consortium Il *World Wide Web Consortium* (W3C) è un'organizzazione internazionale che ha come scopo quello di sviluppare tutte le potenzialità del World Wide Web. Il W3C produce e promuove standard tecnici aperti e liberi per il Web, allo scopo di garantirne la crescita e l'innovazione attraverso l'implementazione di nuove tecnologie e standard. Tra gli standard più conosciuti si possono citare *HTML*, *CSS*, il protocollo *HTTP*, le linee guide *WCAG* per l'accessibilità e, nel contesto della tesi, lo standard *Verifiable Credentials* e *Decentralized Identifiers*. [16](#), [19](#), [23](#), [43](#), [61](#), [69](#), [76](#)

Web3.js In informatica con il termine *Web3.js* si indica una libreria *JavaScript* per interagire principalmente con la *blockchain* di *Ethereum*. La libreria permette di creare applicazioni decentralizzate che possono essere eseguite in un *browser* web. Questa fornisce un'API completa per la creazione e la gestione dei contratti intelligenti, la creazione di transazioni, la gestione delle chiavi crittografiche e la lettura dei dati dalla blockchain. *ethers.js* è basata su tecnologie di crittografia asimmetrica e sulla comunicazione con i nodi della *blockchain* di *Ethereum* attraverso il protocollo *JSON-RPC*. [21](#), [69](#), [76](#)

Zero Knowledge Proof In crittografia con il termine *Zero-Knowledge Proof (ZKP)* (ing. prova a conoscenza zero) si indica un protocollo che permette ad un soggetto di dimostrare di conoscere un certo dato, senza doverlo rivelare. Le *ZKP* sono utilizzate in diverse applicazioni, come la verifica dell'identità digitale, la gestione delle transazioni finanziarie, la condivisione delle informazioni sensibili. [16](#), [19](#), [21](#), [23](#), [26](#), [54](#), [69](#), [76](#)

Bibliografia

Siti web consultati

- [1] *Anoncreds*. URL: <https://hyperledger.github.io/anoncreds-spec/#anoncreds-setup-data-flow> (cit. a p. 54).
- [2] *CL Signature 2019 - Implementazione*. URL: <https://blog.goodaudience.com/cl-signatures-for-anonymous-credentials-93980f720d99> (cit. a p. 54).
- [3] *CL Signature 2019 - Riferimenti (documento e schema di pag. 23)*. URL: <https://arxiv.org/pdf/2208.04692.pdf> (cit. a p. 55).
- [4] *Continuous Integration and Continuous Delivery*. URL: <https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment> (cit. a p. 71).
- [5] *Decentralized Identifiers (DIDs) v1.0*. URL: <https://www.w3.org/TR/did-core/> (cit. a p. 23).
- [6] *Ethereum*. URL: <https://www.ethereum.org/>.
- [7] *Ethereum Architecture*. URL: <https://www.zastrin.com/courses/ethereum-primer/lessons/1-5> (cit. a p. 48).
- [8] *Ethers.js*. URL: <https://docs.ethers.io/ethers.js/html/>.
- [9] *GitHub*. URL: <https://docs.github.com/en> (cit. a p. 44).
- [10] *Hardhat*. URL: <https://hardhat.org/> (cit. a p. 44).
- [11] *Jest*. URL: <https://jestjs.io/> (cit. a p. 62).
- [12] *Node.js*. URL: <https://nodejs.org/en/> (cit. a p. 44).
- [13] *React*. URL: <https://reactjs.org/> (cit. a p. 43).
- [14] *React Testing Library*. URL: <https://testing-library.com/docs/react-testing-library/intro/> (cit. a p. 62).
- [15] *Scrum*. URL: <https://www.scrum.org/resources/what-is-scrum> (cit. a p. 2).
- [16] *Solidity*. URL: <https://solidity.readthedocs.io/en/v0.5.3/> (cit. a p. 44).
- [17] *TypeScript*. URL: <https://www.typescriptlang.org/>.
- [18] *Verifiable Credentials Data Model v1.1*. URL: <https://www.w3.org/TR/vc-data-model/>.
- [19] *Verifiable Credentials Data Model v1.1 - Sezione Zero-Knowledge Proofs*. URL: <https://www.w3.org/TR/vc-data-model/#zero-knowledge-proofs> (cit. a p. 26).
- [20] *Verifiable Credentials Data Model v1.1 - Verifiable Presentation*. URL: <https://www.w3.org/TR/vc-data-model/#example-a-verifiable-presentation-that-supports-cl-signatures> (cit. alle pp. 25, 55).
- [21] *WAVE Web Accessibility Evaluation Tool*. URL: <https://wave.webaim.org/> (cit. a p. 62).

- [22] *WCAG Contrast Checker*. URL: <https://webaim.org/resources/contrastchecker/> (cit. a p. 61).
- [23] *Web Content Accessibility Guidelines (WCAG) 2.1*. URL: <https://www.w3.org/TR/WCAG21/> (cit. a p. 61).
- [24] *Web3.js*. URL: <https://web3js.readthedocs.io/en/1.0/> (cit. a p. 44).
- [25] *ZKP Standard*. URL: <https://docs.zkproof.org/reference.pdf> (cit. a p. 54).