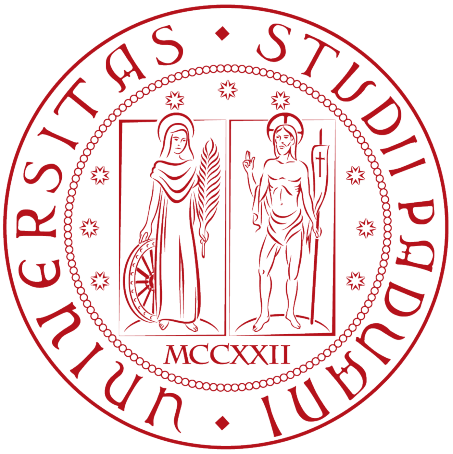
**Università degli Studi di Padova**

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica


# Titolo della tesi

*Tesi di laurea*


*Relatore*

Prof. Tullio Vardanega

*Laureando*

Pietro Lauriola



Anno Accademico 2022-2023

# Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di trecentoventi (320) ore, dal laureando Pietro Lauriola presso l'azienda Sync Lab S.r.l. Gli obbiettivi da raggiungere erano i seguenti:
In primo luogo era richiesto la stesura di uno Studio di fattibilità circa la possibilità di utilizzare la Zero Knowledge Proof per sviluppare una piattaforma di voto verificabile ma anonima, ovvero che raccolga le votazioni degli utenti approvati senza che sia possibile risalire a cosa abbiano votato ma rendendo facile la verifica del fatto che abbiano effettivamente votato. In secondo luogo era richiesta l'implementazione di un Proof of Concept (PoC) per dimostrare la fattibilità del progetto.

*"And yet I smile"*
— Ezekiel

# Ringraziamenti

*Padova, Settembre 2023* Pietro Lauriola

# Indice

# Elenco delle figure



# Elenco delle tabelle

# Capitolo 1

# Contesto aziendale

In questo capitolo presento il contesto organizzativo e produttivo dell'azienda *SyncLab S.r.L.*. Viene fornita una descrizione di ciò che ho potuto osservare riguardo le tecnologie utilizzate, i processi interni dell'azienda, il tipo di clientela e la propensione dell'azienda per l'innovazione.

## 1.1 L'azienda

L'azienda ospitante è stata ***SyncLab S.r.L.***, nata nel 2002 a Napoli e attiva nel settore dell' *Information and Communication Technology* (ITC). Con il passare degli anni si è espansa aprendo in tutto 6 sedi in Italia, a Napoli, Roma, Milano, Padova, Verona e Como.

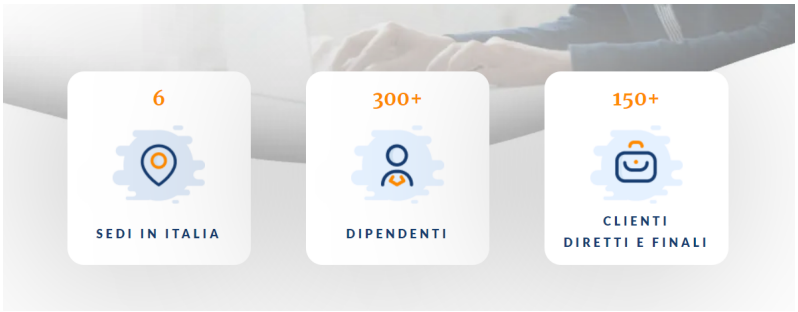



**Figura 1.1:** Dati relativi all'azienda
**Fonte:** synclab.it

Grazie a una struttura interna che favorisce la collaborazione, l'interazione non si limita ai colleghi della stessa sede, ma si estende globalmente in tutta l'azienda.

Lo scopo è quello di promuovere lo scambio di conoscenze all'interno dell'organizzazione e creare un ambiente in cui il progresso personale non sia il risultato esclusivo degli sforzi individuali, ma anche della collaborazione attiva tra i membri del *team*.

È rilevante sottolineare che ho notato che i dipendenti presenti, almeno nella sede di Padova, sono principalmente giovani.

*SyncLab S.r.L.*è identificabile come *System Integrator*, sebbene sia nata come una *Software House*. La differenza tra i due ambiti è rilevante per comprendere il modus operandi dell'azienda.

**Software House**: un'azienda che sviluppa internamente delle soluzioni software che soddisfino una certa opportunità di mercato, e offre i propri prodotti ai clienti interessati.

**System Integrator**: un'azienda che, contattata da aziende esterne, effettua manutenzione e evoluzione delle funzionalità di prodotti *software* già sviluppati e in uso.

Si tratta quindi di due approcci allo sviluppo ben diversi : mentre una *Software House* si concentra sulla creazione e sviluppo di soluzioni *software* innovative da zero, basandosi sulle esigenze e opportunità del mercato, un *System Integrator* si focalizza sull'ottimizzazione, l'integrazione e la manutenzione di software esistenti in base alle esigenze dei clienti che già utilizzano tali prodotti.

## 1.2 Settori di attività e progetti

*SyncLab S.r.L.*collabora con numerosi clienti, che operano in diversi ambiti, tra cui: *EHealt, Telco, Web and Mobile, Data Management, Blockchain, Maritime.*


**Figura 1.2:** Alcuni ambiti in cui l'azienda opera
**Fonte:** synclab.it

Dopo aver discusso degli attuali ambiti di operatività di *SyncLab*, è interessante fare un salto indietro e analizzare le radici dell'azienda.

Quando *SyncLab* operava come *Software House*, ha creato una serie di prodotti e soluzioni innovative che hanno contribuito a definire il suo prestigio nel settore *IT*. Prodotti e soluzioni che non solo hanno consolidato la sua posizione nel mercato, ma hanno anche gettato le basi per la sua evoluzione futura.

Vediamo alcuni dei progetti più rappresentativi e influenti realizzati da *SyncLab* in quella fase cruciale della sua storia.

- **SynClinic**: Una soluzione nel settore dell'*EHealth* progettata per centralizzare e facilitare la gestione sia clinica che amministrativa di ospedali, strutture sanitarie e residenze mediche.

  Il suo obiettivo principale è assistere il personale sanitario nell'identificazione e nella gestione del rischio clinico, garantendo al contempo un'ottima tracciabilità e organizzazione delle varie tappe del trattamento del paziente.

- **SeaStream**: Questa piattaforma è stata progettata per potenziare l'efficienza, la sicurezza e incoraggiare l'innovazione nel dominio marittimo.
  SeaStream presenta un *Fleet Operation Center* (FOC) che offre una sorveglianza avanzata delle flotte navali in operazione globalmente.
  Inoltre, mette a disposizione una *Harbor Operation Platform* (HOC), che fornisce una gamma completa di servizi destinati ai professionisti del settore portuale.
- **Fast Reservation**: Una piattaforma digitale ideata per la gestione delle prenotazioni, adattabile a diverse realtà come stabilimenti balneari, parchi e attività nel settore della ristorazione. L'obiettivo è rendere il processo di prenotazione fluido e intuitivo per gli utenti.
- **Sobereye**: Una soluzione basata sul *web* progettata per analizzare e monitorare lo stato psicofisico di una persona mediante l'osservazione della pupilla. Questa applicazione è particolarmente utile per identificare alterazioni potenzialmente causate da stanchezza eccessiva o assunzione di sostanze come alcool e droghe, contribuendo a ridurre potenziali rischi sul luogo di lavoro.

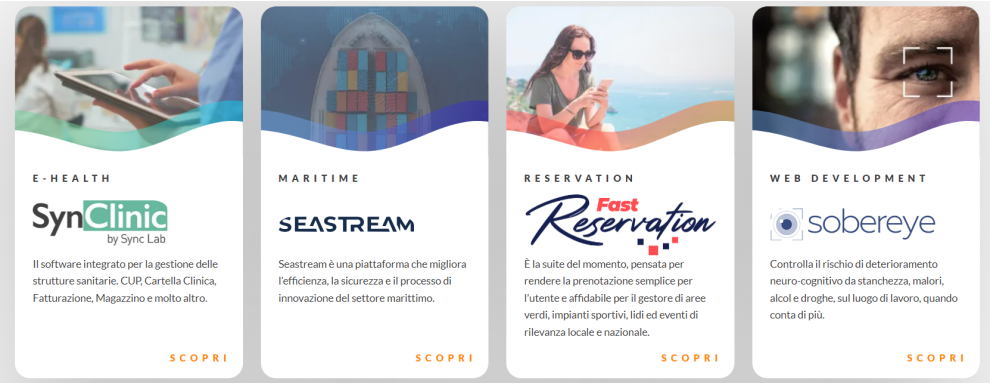



**Figura 1.3:** Alcuni progetti sviluppati dall'azienda
**Fonte:** synclab.it

## 1.3 Way of working

### 1.3.1 Gestione dello smart working

Durante il percorso di tirocinio, sebbene la sede dell'azienda fosse a Padova, ho svolto la maggior parte del lavoro da remoto. L'azienda adotta un metodo di lavoro che consiste nel trovarsi in sede circa una volta a settimana per confrontarsi sul lavoro svolto, sui progressi fatti e sulle difficoltà riscontrate.

Il restante del tempo lavorativo viene svolto in autonomia e, qualora ci si trovi in difficoltà, si possono utilizzare gli strumenti di comunicazione da remoto per risolvere dubbi o problemi.

### 1.3.2 Tecnologie interne

In questa sezione tratteremo alcune tecnologie di cui ho avuto esperienza diretta. Tuttavia, non procedo a fornire specifiche dettagliate riguardo al loro utilizzo interno

per la gestione del lavoro.

*SyncLab* utilizza un'ampia gamma di tecnologie, che includono linguaggi di programmazione e *framework* all'avanguardia nel settore dell'*Information and Communication Technology*.

L'azienda fa ampio uso di linguaggi di programmazione come **JavaScript, TypeScript, Java, Python, Solidity**.


**Figura 1.4:** Alcuni linguaggi di programmazione in uso da SyncLab

Solitamente questi linguaggi vengano affiancati da *framework*, che offrono un'infrastruttura predefinita per lo sviluppo di applicazioni. Questi *framework* facilitano la creazione di *software* efficiente e scalabile, fornendo librerie predefinite, strumenti di sviluppo e modelli architetturali. Grazie alla loro natura modulare e flessibile, i *framework* consentono ai programmatori di concentrarsi sull'implementazione delle funzionalità specifiche, migliorando la qualità del *software* sviluppato. Di seguito i principali *framework* utilizzati:

- **Angular**: *Framework* di sviluppo *front-end* basato su JavaScript. Offre una potente piattaforma per la creazione di applicazione *web* scalabili e reattive. Attualmente è una delle soluzioni più utilizzate nel settore.
- **Java Spring**: *Framework* di sviluppo *back-end* basato su Java. Fornisce una vasta gamma di moduli e funzionalità per la creazione di applicazioni Java robuste. Attualmente si è imposto come *standard de facto* per lo sviluppo di servizi *web* in Java.
- **Odoo**: *Framework open source* per lo sviluppo di applicazioni di gestione aziendale, basato in Python. Odoo è altamente personalizzabile e modulare grazie al vasto insieme di moduli offerti per la gestione delle vendite, degli acquisti, delle risorse umane, della contabilità e molti altri.


**Figura 1.5:** Alcuni framework in uso da SyncLab

Questa vasta gamma di tecnologie consentono di adattarsi alle esigenze specifiche dei clienti e offrire soluzioni *software* all'avanguardia che combinano efficienza, funzionalità e usabilità.

All'interno di un **gruppo di lavoro**, la normazione, regolamentazione e sincronizzazione delle attività sono fondamentali per garantire un flusso di lavoro efficiente ed efficace. A tal fine, l'utilizzo di *software* appositamente progettati volge un ruolo cruciale.

Di seguito alcuni *software* che svolgono questo ruolo:

- **Git**: Un **sistema di controllo di versione distribuito**, ampiamente adottato. Consente di tenere traccia delle modifiche apportate ai *file* e coordinare il lavoro

di più persone. Attraverso *Git*, i membri del gruppo possono collaborare in modo sincronizzato, gestire i conflitti, apportare modifiche senza sovrascrivere il lavoro degli altri e recuperare versioni precedenti dei *file*.

- **VS Code**: Un **ambiente di sviluppo integrato** (IDE) che offre un'interfaccia unificata per la scrittura del codice, la gestione dei *file* e la condivisione dei progetti. *VS Code* facilita la codifica collaborativa, fornendo strumenti per *debugging*, completamento del codice e integrazione con *Git*, agevolando la gestione e la tracciabilità delle modifiche al codice.

- **IntelliJ IDEA**: Un **ambiente di sviluppo integrato** (IDE) concepito primariamente per facilitare la programmazione in *Java*. Offre una vasta gamma di funzionalità destinate a assistere lo sviluppatore in ogni fase della codifica. Tra le caratteristiche più rilevanti, *IntelliJ IDEA* incorpora l'analisi statica del codice, permettendo di rilevare e segnalare errori sia logici che sintattici ancor prima dell'esecuzione del programma. Inoltre, si integra perfettamente con strumenti di versionamento esterni, come *Git*, agevolando la gestione e la tracciabilità delle modifiche al codice.

**Figura 1.6:** Alcuni software in uso da SyncLab

Nell'ambito della **comunicazione interna**, l'utilizzo di strumenti dedicati è essenziale per garantire una comunicazione efficace, per questo sono state individuate soluzioni come:

- **Discord**: Piattaforma di **comunicazione vocale e testuale**, ampiamente utilizzata e consente ai membri del gruppo di scambiare messaggi istantanei ed effettuare chiamate. *Discord* offre anche funzionalità aggiuntive, come la creazione di canali tematici per avere una comunicazione *topic based*.

- **Google Meet**: Piattaforma per **videoconferenze**, che permette di tenere riunioni *online*, condividendo schermi, documenti e presentazioni in tempo reale. Questo strumento è stato maggiormente usato durante e in seguito alla pandemia di SARS-CoV-2.

- **Google Calendar**: **Software gestionale** per la creazione di calendari privati e condivisi tra più utenti. Attraverso *Google Calendar* è possibile creare eventi, impostare promemoria e condividere le proprie disponibilità con gli altri membri. Viene anche utilizzato per organizzare l'alternanza tra *smart working* e lavoro in presenza.

- **Trello**: **Software gestionale** che adotta la filosofia Kanban, essendo ispirato alla metodologia della *Scrum board* tipica del modello agile. Permette di organizzare e tracciare il progresso dei lavori attraverso l'uso di schede specifiche, ciascuna associata a un determinato *task*. Questa strutturazione consente di avere una visione chiara e aggiornata dello stato dei progetti, e di facilitare la comunicazione e sincronizzazione tra i membri del *team* di sviluppo.

## 1.4 Dallo stack tecnologico ad applicativo

Nel vasto dominio dell'ingegneria del *software*, non si possono considerare le tecnologie in isolamento. Esse sono piuttosto dei pezzi che, messi insieme, costituiscono uno "*stack* tecnologico".
Questa integrazione ha particolare rilevanza quando ci si concentra sullo sviluppo di applicazioni *web*. Queste applicazioni, infatti, si articolano in due settori chiave: il *front-end*, focalizzato sull'esperienza e l'interfaccia utente, e il *back-end*, che gestisce la logica, la manipolazione dei dati e la loro persistenza.

Se dovessimo esemplificare utilizzando gli strumenti menzionati precedentemente:

- **Front-end**: La scelta potrebbe cadere su Angular come *framework*, concepito per creare interfacce *web* dinamiche. Questa scelta trascina con sé l'uso del linguaggio TypeScript e dell'HTML5. Sono tecnologie intrinsecamente collegate, funzionando in tandem per fornire l'esperienza desiderata.

- **Back-end**: In questo contesto, si potrebbe optare per Java Spring, ideale per elaborare logiche dati complesse, mentre la scelta del DBMS per la gestione dei dati potrebbe essere più flessibile e non direttamente vincolata dalla scelta del *framework*.

A queste tecnologie principali, si aggiungono strumenti che supportano il processo di sviluppo: IDE come IntelliJ IDEA per Java e VSCode per TypeScript; strumenti di versionamento come Git e soluzioni per la comunicazione e gestione di progetto, come Discord e Trello.

Questi *stack*, o insiemi di tecnologie, interagiscono tra loro, formando un mosaico tecnologico che costituisce lo scheletro di qualsiasi progetto.
L'immagine seguente offre una sintesi visuale delle tecnologie discusse, categorizzate per ambito di utilizzo.

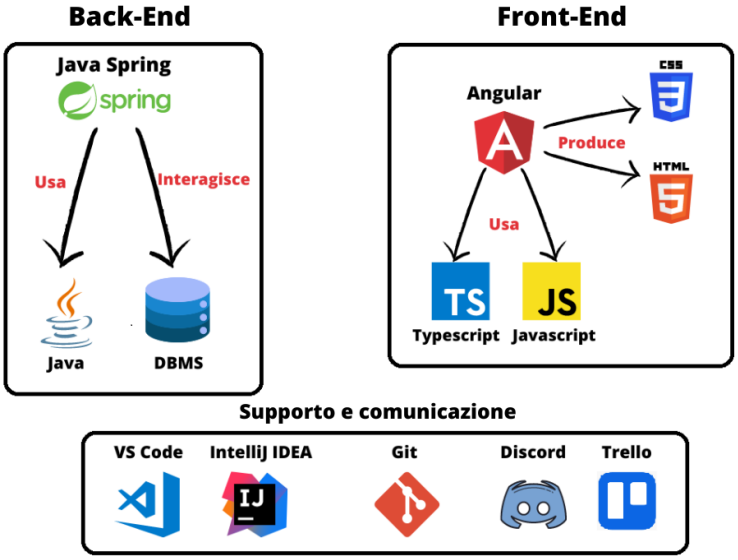



**Figura 1.7:** Esempio di *stack* tecnologico

Il punto cardine è che le tecnologie adottate da *SyncLab* (o qualsiasi *team* di sviluppo) devono essere viste come elementi di un ecosistema più ampio, con una visione orientata verso l'obiettivo finale piuttosto che sui dettagli specifici.

Vale la pena sottolineare che gli *stack* tecnologici proposti qui sono illustrativi. Sono modellati intorno alle tecnologie prevalenti in *SyncLab* e servono come esempi di come si potrebbero combinare diverse tecnologie. La scelta di uno *stack* non è prescrittiva; dipende da numerosi fattori e non è intenzione di questo lavoro prescrivere un approccio unico o limitante.

## 1.5 Propensione all'innovazione

L'azienda si distingue per la sua costante propensione all'innovazione, che costituisce uno dei pilastri fondamentali della sua filosofia aziendale.

L'arduo e persistente impegno di *SyncLab* nel perseguire l'innovazione è palpabile in ogni aspetto della sua attività. Questa dedizione si traduce in una incessante e meticolosa ricerca di metodologie all'avanguardia, tecnologie di ultima generazione e approcci rivoluzionari.

**Vi è un dualismo**, che emerge dalla filosofia di *SyncLab*:
**Da un lato**, vi è l'incessante impegno per affinare le dinamiche interne, dall'ottimizzazione della gestione dei progetti, al potenziamento delle relazioni con i clienti, dalla ricerca di una maggiore efficienza operativa alla dedizione nella formazione continua del suo *team*. Queste iniziative riflettono la passione dell'azienda per l'eccellenza in ogni aspetto della sua operatività.

**Dall'altro lato**, *SyncLab* non si limita a perfezionare l'infrastruttura interna. La sua visione trascende queste operazioni interne, rivolgendo lo sguardo verso il panorama esterno. L'obiettivo predominante è la creazione e proposizione di soluzioni di avanguardia, formulate per rispondere alle sfide crescenti e intricate del mercato attuale. Questi prodotti e servizi non sono semplicemente reattivi, ma anticipatori, mirando a generare un impatto significativo e a lungo termine nell'ecosistema tecnologico.

**La visione pionieristica**
Oltre a rispondere alle immediate necessità dei clienti, l'intento è di anticipare le tendenze future e posizionarsi come pionieri nel settore. La visione dell'azienda è chiara: migliorare costantemente sia l'infrastruttura interna sia le soluzioni offerte, assicurando che entrambe siano sempre all'avanguardia.

La mentalità aperta all'innovazione e la capacità di adattamento rappresentano quindi i *driver* fondamentali che guidano l'azienda nel suo percorso di crescita e successo nel panorama competitivo odierno.

**Un esempio** tangibile della visione avanguardista di *SyncLab* è la sua dedizione verso la tecnologia *blockchain*.

Questo impegno non si riflette solo in una mera scelta tecnologica, ma evidenzia un profondo desiderio dell'azienda di essere al passo con le innovazioni più pregnanti nel dominio *IT*.

La mia esperienza all'interno dell'azienda ha rivelato come *SyncLab* non solo riconosca l'importanza cardine della *blockchain*, allocando risorse significative per comprenderla in profondità, ma stia anche realizzando lavori concreti basati su questa tecnologia per conto di clienti che ne hanno riconosciuto il potenziale e ne hanno richiesto l'implementazione
Tali lavori, ancorati nella pratica e richiesti da clienti reali, mirano a sfruttare tutto il potenziale della *blockchain*.

L'obiettivo è doppio: da un lato, innovare e ottimizzare i processi esistenti, dall'altro, fornire nuove soluzioni che possano determinare una svolta nel panorama tecnologico attuale.

Questo approccio, che fonde analisi approfondita con applicazione diretta, dimostra la determinazione di *SyncLab* nel portare avanti l'innovazione.
L'azienda non si accontenta di stare al passo con le ultime tendenze, ma si posiziona attivamente come un agente di cambiamento, erogando servizi di alto valore ai suoi clienti.

# Capitolo 2

# Lo stage

## 2.1 Strategia aziendale

### 2.1.1 Offerte aziendali

*SyncLab* ha costruito una rete solida con importanti università italiane. Queste collaborazioni si manifestano in diversi modi: oltre a partecipare attivamente a progetti proposti all'interno di specifici corsi di laurea, *SyncLab* è regolarmente presente e attiva in eventi accademici, come Stage-IT, sottolineando la sua dedizione nell'interazione diretta con la comunità accademica e gli studenti.


**Figura 2.1:** Partner accademici di Synclab
**Fonte:** synclab.it

In particolare, attraverso programmi di *stage* ben strutturati, l'azienda ha potuto trarre vantaggio dal flusso costante di nuove idee, conoscenze e competenze.

Sulla base della mia interazione diretta con l'azienda, ho identificato un aspetto saliente relativo alle loro offerte di *stage*: queste non sono unicamente intese come strumenti per la formazione e l'orientamento di potenziali professionisti. Invece, rappresentano anche un mezzo attraverso il quale l'azienda stessa mira ad acquisire nuove competenze e ad approfondire aree inesplorate del settore.

Questo approccio dimostra la visione olistica dell'azienda riguardo all'apprendimento e all'innovazione: un ciclo continuo in cui l'azienda impara dagli stagisti tanto

quanto gli stagisti imparano dall'azienda.

I programmi di *stage* offerti da *SyncLab* riflettono chiaramente la strategia aziendale di rimanere all'avanguardia nel panorama *IT*. Essi sono strutturati in base alle esigenze attuali dell'azienda e alle tendenze emergenti nel settore:

- **Sviluppo e Integrazione**: In questo ambito, gli stagisti hanno la responsabilità di perfezionare *software* preesistenti, lavorando su specifiche funzionalità atomiche. Sebbene queste funzionalità non siano destinate alla produzione finale, rappresentano, in alcuni casi, soluzioni alternative che l'azienda aveva precedentemente considerato.
  Gli stagisti, attraverso questo approccio, hanno l'opportunità di esplorare e sviluppare questi concetti, contribuendo a una comprensione più ampia delle potenzialità e delle alternative del *software* in questione.

- **Ricerca e Innovazione**: Questo percorso si focalizza sull'approfondimento teorico di tecnologie emergenti e concetti in evoluzione nel settore *IT*.
  Gli stagisti sono incaricati di condurre studi specifici, mirati a esplorare aspetti particolari e dettagliati delle nuove tecnologie, sempre in linea con le esigenze e gli obiettivi precisi dell'azienda.
  Questa immersione teorica fornisce una solida base per comprendere le nuove tendenze, valutare il loro potenziale e considerare possibili applicazioni pratiche nel contesto aziendale.

- **Analisi e Ottimizzazione**: Questa area si concentra sulla valutazione dettagliata delle soluzioni *software* correntemente in uso all'interno dell'azienda.
  Gli stagisti sono incaricati di esaminare questi sistemi, identificare possibili inefficienze o aree di miglioramento e proporre soluzioni ottimizzate.
  La chiave di questa fase è un approccio critico e analitico, che mira a garantire che il *software* dell'azienda funzioni al suo massimo potenziale, sfruttando al meglio le risorse disponibili e rispondendo in modo efficiente alle esigenze degli utenti.

### 2.1.2 Stage in azienda

Gli *stage* presso *SyncLab* sono quindi visti non solo come un'opportunità per identificare e formare futuri professionisti, ma anche come un canale per iniettare innovazione e nuove idee nel tessuto dell'organizzazione.

Questo approccio sottolinea l'importanza strategica che l'azienda attribuisce all'integrazione di giovani nel suo ecosistema.

La partecipazione attiva degli stagisti ai progetti aziendali è centrale in questa prospettiva. L'azienda li coinvolge in attività che hanno un impatto diretto sugli esiti dei progetti, garantendo che le loro competenze e visioni vengano valorizzate e utilizzate per raggiungere gli obiettivi dell'organizzazione.

Inoltre, è da notare come l'esperienza di *stage* presso *SyncLab* possa rappresentare un'opportunità significativa per una carriera a lungo termine all'interno dell'organizzazione. Molti stagisti, avendo dimostrato valore e impegno, ricevono offerte di assunzione come membri effettivi del *team*.

Infine, è degno di nota come *SyncLab* non solo valorizzi individualmente ogni stagista, ma promuova anche l'interazione tra di loro.

Quando più stagisti lavorano su tematiche o aspetti correlati, l'azienda li mette attivamente in contatto, favorendo un ambiente collaborativo.

Questa strategia permette agli stagisti di condividere le proprie competenze, di aiutarsi reciprocamente e di arricchire ulteriormente la loro esperienza formativa.

**Metti un'immagine tra i paragrafi!**

### 2.1.3 Ruolo del tutor

Un aspetto fondamentale dell'esperienza di *stage* presso *SyncLab* è la presenza di un *tutor* personale che accompagna lo stagista lungo tutto il percorso. Questo *tutor* non è mai scelto a caso, ma è una figura esperta nel settore specifico in cui lo stagista opererà. La sua presenza è cruciale per guidare, consigliare e offrire *feedback* costruttivi basati sulla sua vasta esperienza.

Questa strategia dell'azienda assicura che ogni stagista sia supportato e guidato in maniera ottimale.

Nel mio caso specifico, a seguito della mia scelta specifica di *stage*, è stato assegnato *Matteo Galvagni* come mio *tutor*, proprio perché la sua esperienza era particolarmente in linea con il settore in cui avrei lavorato, ossia la *blockchain*.

Il *tutor*, oltre a fornirmi indicazioni e strumenti utili per velocizzare la comprensione e assimilazione delle conoscenze necessarie, ha giocato un ruolo fondamentale anche nella fornitura di *feedback*.

Un elemento distintivo dell'esperienza degli stagisti è proprio la frequenza e la qualità del *feedback* ricevuto. *SyncLab* si impegna a fornire valutazioni regolari e dettagliate, garantendo agli stagisti opportunità di crescita e sviluppo professionale coerenti con le esigenze aziendali.

## 2.2 Progetto proposto

### 2.2.1 Introduzione

Nell'era digitale, la ricerca di sistemi di votazione sicuri, trasparenti e al contempo garantisti della *privacy* è divenuta una priorità.
Il progetto intrapreso ha come scopo primario lo sviluppo di una piattaforma di votazione online che affronta **due sfide fondamentali**:

- Garantire l'anonimato totale degli elettori
- Garantire l'integrità inalterabile delle votazioni.

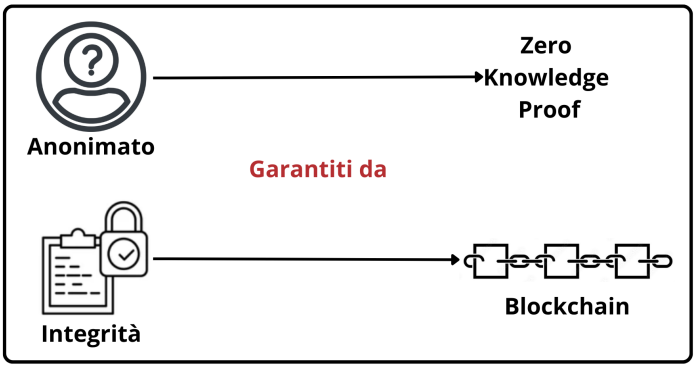



**Figura 2.2:** Le due sfide del progetto

Per quanto riguarda la **prima sfida**, il sistema deve essere progettato per rendere il voto online completamente anonimo. Gli elettori devono poter partecipare alle votazioni senza rivelare la propria identità, garantendo così la massima *privacy*. Ciò si è supposto fosse possibile attraverso l'utilizzo di tecnologie di crittografia, in particolare la *Zero Knowledge Proof (ZKP)*. La *ZKP* può consentire agli elettori di dimostrare che hanno il diritto di votare senza rivelare chi sono o cosa hanno votato.

Per quanto riguarda la **seconda sfida**, la *blockchain* svolge un ruolo cruciale nel garantire l'integrità delle votazioni. La *blockchain* è una tecnologia distribuita che consente di registrare le votazioni in modo permanente e immutabile. Una volta che dei dati, nel nostro caso i voti, vengono registrati sulla *blockchain*, diventa impossibile modificarli o cancellarli senza lasciare traccia. Questo livello di sicurezza è fondamentale per garantire che le votazioni siano affidabili e immuni da frodi o manipolazioni.

Lo *stage* è finalizzato non solo a valutare la fattibilità di tale sistema ma anche a sviluppare un *Proof of Concept (PoC)* funzionante. Il *PoC* servirà a dimostrare concretamente che il sistema può essere implementato e che è in grado di garantire l'anonimato degli elettori e l'integrità delle votazioni.
In sintesi, il progetto mira a rivoluzionare il modo in cui il voto online è condotto, garantendo una maggiore *privacy* agli elettori e una maggiore sicurezza alle votazioni stesse attraverso l'uso combinato della *Zero Knowledge Proof* e della *blockchain*. La fase di sviluppo del *Proof of Concept* sarà cruciale per dimostrare l'efficacia di questo approccio innovativo e per gettare le basi per futuri sviluppi nel campo delle votazioni online.

### 2.2.2 Zero knowledge proof

Nell'ambito di questo progetto, la *Zero Knowledge Proof (ZKP)* svolge un ruolo di primaria importanza. Essa consente a una delle parti coinvolte, chiamata "*prover*," di dimostrare la veridicità di un'affermazione senza dover rivelare alcuna informazione specifica all'altra parte, nota come "*verifier*." Questa capacità di coniugare autenticità e riservatezza riveste un'importanza cruciale in contesti quali le transazioni *blockchain* e i sistemi di voto elettronico.

Per effettuare questa dimostrazione, viene generata una "prova" utilizzando complesse funzioni di crittografia. Questa "prova," che rappresenta il fulcro del concetto in una *ZKP*, è una rappresentazione crittograficamente robusta che il *prover* fornisce al *verifier* per dimostrare la veridicità di una determinata affermazione. La sua generazione avviene in modo tale da risultare convincente, ma senza mai rivelare dati specifici.

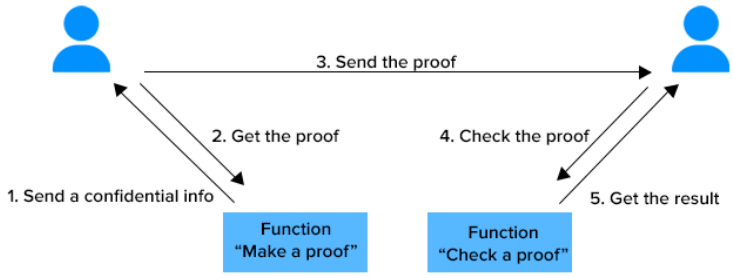



**Figura 2.3:** Generazione e controllo prova
**Fonte:** Medium

Per esempio, nel contesto delle elezioni elettroniche, un elettore (*prover*) può dimostrare che il proprio voto è stato registrato correttamente senza dover rivelare il contenuto effettivo del voto. Questa "prova" rappresenta l'evidenza crittografica che il voto è legittimo e corretto, ma non rivela cosa sia stato votato. Il sistema di verifica (*verifier*) può quindi confermare la validità del voto senza conoscere il suo contenuto, proteggendo così l'anonimato degli elettori e l'integrità dei risultati elettorali.

Oltre alle applicazioni nelle votazioni, la *ZKP* gioca un ruolo determinante nella protezione della *privacy* in transazioni *blockchain*. Ad esempio, gli utenti possono dimostrare la disponibilità di fondi sufficienti per una transazione senza dover specificare l'ammontare esatto dei loro averi. Questa "prova" crittografica consente agli utenti di effettuare transazioni in modo sicuro senza rivelare informazioni sensibili.

In conclusione, la *Zero Knowledge Proof* offre una soluzione avanzata per garantire sia la sicurezza che la *privacy* nelle transazioni digitali e nei processi di voto, grazie all'uso di "prove" crittografiche che dimostrano la veridicità senza la necessità di rivelare dati specifici.

## 2.3 Obiettivi

Nel contesto di qualsiasi progetto, definire obiettivi chiari rappresenta una fase cruciale. Senza obiettivi ben definiti, un progetto rischia di perdersi nell'incertezza, di deviare dalla sua finalità e di non produrre risultati concreti. Pertanto, nella realizzazione di questo progetto la determinazione di obiettivi ben definiti è stata un imperativo.
Di seguito, gli obiettivi specifici del progetto sono lo **studio di fattibilità e lo sviluppo del *Proof of Concept***, come di seguito:

**Studio di Fattibilità**: Condurre uno studio dettagliato sulla fattibilità dell'utilizzo della *Zero Knowledge Proof (ZKP)* per garantire la *privacy* nelle votazioni elettroniche. Questa analisi ha un focus particolare su due componenti fondamentali: la tecnologia *blockchain* e la *ZKP*.

Per quanto riguarda la tecnologia *blockchain*, si esaminano le diverse piattaforme e protocolli disponibili per determinare quale sia più adatta al contesto delle votazioni elettroniche. Questo include l'analisi di varie *blockchain*, nonché considerazioni sulla scalabilità e sulla sicurezza.

Per quanto riguarda la *Zero Knowledge Proof (ZKP)*, vengono esplorate le diverse varianti e implementazioni di questa tecnologia crittografica. Inoltre, vengono valutate le loro applicazioni potenziali nel contesto delle votazioni elettroniche, identificando le loro capacità e le eventuali limitazioni.

Lo studio di fattibilità ha quindi l'obiettivo di fornire una panoramica chiara delle strade possibili per l'implementazione della *ZKP* nelle votazioni elettroniche. Così da concentrarci sulle soluzioni tecniche più promettenti e identificare le sfide chiave da affrontare durante lo sviluppo del *Proof of Concept (PoC)*.

**Sviluppo di un *Proof of Concept (PoC)***: Il *Proof of Concept (PoC)* è un'implementazione pratica e limitata della *Zero Knowledge Proof (ZKP)* all'interno della piattaforma di voto elettronico. Il suo scopo principale è dimostrare che la *ZKP* può essere applicata con successo per garantire l'anonimato degli elettori e la sicurezza delle votazioni elettroniche.

In questa fase, ci concentriamo sull'implementazione dei concetti teorici relativi alla *ZKP* in un ambiente controllato. Il *PoC* funge da prototipo iniziale e dimostra come un elettore può votare in modo anonimo e come il sistema può verificare l'autenticità dei voti senza rivelare i dettagli specifici del voto.

I risultati del *PoC* influenzeranno le eventuali fasi successive del progetto, guidando le decisioni riguardo all'implementazione su larga scala della piattaforma di voto basata sulla *ZKP*.

## 2.4 Vincoli

Nel contesto di qualsiasi progetto, è fondamentale considerare attentamente i vincoli che possono influenzare la sua realizzazione. I vincoli definiscono le restrizioni e le limitazioni che devono essere prese in considerazione per garantire il successo del

progetto. Nel caso specifico della piattaforma di voto, sono presenti diversi vincoli che devono essere affrontati per garantire la riservatezza, l'integrità e la verificabilità del processo di voto.

- **Vincolo di riservatezza**
  Il sistema deve garantire che, sebbene ogni voto possa essere verificato come legittimo, l'identità dell'elettore e la sua scelta specifica rimangano completamente anonime. In altre parole, è fondamentale che sia impossibile determinare cosa ha votato un singolo utente

- **Vincolo di unicità del voto**
  Ogni indirizzo utilizzato per votare è considerato come un'entità unica nel sistema. Pertanto, è essenziale che un utente non possa esprimere più di un voto utilizzando lo stesso indirizzo.

- **Vincolo di integrità del voto**
  La *blockchain*, per sua natura, è una struttura dati immutabile. Questo vincolo è fondamentale per garantire l'integrità del sistema di votazione. Non deve essere possibile, in nessun caso, modificare i voti una volta che questi sono stati registrati. Questo assicura che non sia possibile alterare il voto di altri utenti, garantendo che ogni voto conteggiato sia effettivamente quello espresso dall'elettore.

- **Vincolo di verificabilità**
  Chiunque deve poter verificare che i voti siano validi e il conteggio corretto. La piattaforma di voto deve fornire un meccanismo di verifica che consenta agli utenti di accedere ai risultati del voto in modo trasparente e affidabile. Questo viene realizzato attraverso l'utilizzo della *blockchain*, che permette la registrazione pubblica e immutabile dei voti, e l'utilizzo di algoritmi crittografici per garantire l'integrità dei dati.

Affrontare questi vincoli è fondamentale per creare una piattaforma di voto sicura, affidabile e trasparente. La considerazione di questi vincoli durante tutto il processo di sviluppo e implementazione del progetto garantisce che la piattaforma soddisfi le esigenze di *privacy* degli elettori, l'integrità del processo di voto e la verificabilità dei risultati.

## 2.5 Motivazione della scelta

Ho conosciuto l'azienda tramite l'evento Stage-IT 2023, un'iniziativa promossa da Confindustria Veneto Est in collaborazione con i Dipartimenti di Matematica e Scienze Statistiche dell'Università di Padova, a cui ha partecipato anche il Dipartimento di Ingegneria Informatica. Durante l'evento, ho avuto l'opportunità di entrare in contatto con diverse aziende che proponevano progetti innovativi nel campo dell'*IT*. Sebbene vi fossero molteplici opportunità offerte da diverse aziende, la proposta di *Synclab* ha suscitato un interesse particolare in me.

Oltre alla mia conoscenza dell'azienda tramite Stage-IT, ci sono state altre ragioni che mi hanno spinto a scegliere questo progetto, che possiamo riassumere nei seguenti punti:

- **Padroneggiare almeno una tecnologia legata alla blockchain**: Mi sono posto l'obiettivo di padroneggiare una tecnologia *blockchain* specifica. Ho riconosciuto questa sfida come un'opportunità per acquisire competenze di alto livello

nel campo in rapida crescita della *blockchain* . La *blockchain* sta diventando sempre più rilevante in molteplici settori, e ho visto questa occasione come un passo decisivo per diventare un esperto in questo campo. L'opportunità di lavorare direttamente con la tecnologia *blockchain* e di approfondirne la comprensione rappresentava una proposta irresistibile che ho deciso di abbracciare.

- **Sviluppare con successo *smart contract* funzionanti e sicuri**: Ancorato al mondo della *blockchain*, lo sviluppo e l'implementazione di *smart contracts* rappresenta una delle aree di maggiore crescita e innovazione. Gli *smart contract* sono programmi autonomi che eseguono automaticamente le condizioni stabilite al loro interno.
  Il mio obiettivo è stato quello di acquisire competenze concrete nello sviluppo di *smart contracts* complessi che siano in grado di gestire transazioni sofisticate e applicazioni decentralizzate.

- **Partecipare attivamente a riunioni aziendali**: La teoria e la pratica sono due facce della stessa medaglia. L'opportunità di osservare e partecipare attivamente alla vita quotidiana di un'azienda *software* avrebbe garantito una visione pratica e applicata delle mie conoscenze teoriche. È stata un'opportunità per imparare da professionisti del settore e per sviluppare competenze trasversali, come la gestione del tempo, la collaborazione e la comunicazione efficace

- **Possibilità di Proseguire il Lavoro in Azienda**: Al di là dell'esperienza immediata dello *stage*, la prospettiva di una possibile continuazione professionale con *Synclab* rappresentava un ulteriore incentivo. Lavorare su un progetto significativo con la possibilità di una futura collaborazione a lungo termine è stata un'ottima opportunità .

In conclusione, ho scelto di lavorare su questo progetto presso questa azienda perché mi ha colpito fin dal primo incontro durante Stage-IT. Le opportunità di acquisire competenze *blockchain* e *smart contract*, l'esperienza diretta nel contesto aziendale e la possibilità di proseguire il lavoro in azienda sono state le principali motivazioni che mi hanno spinto a fare questa scelta.

# Capitolo 3

# Il progetto: Svolgimento

## 3.1 Pianificazione

### 3.1.1 Il lavoro

La gestione del tempo e delle risorse è un fattore critico per il successo di qualsiasi progetto, in particolare in ambito informatico, dove le fasi di sviluppo, *test* e rilascio devono essere accuratamente sincronizzate. Nella seguente sezione verrà analizzata la pianificazione delle attività relative all'implementazione del sistema, scomponendo la durata complessiva di otto settimane in fasi specifiche.

È essenziale avere una chiara comprensione di come le risorse temporali vengano allocate e gestite. Una distribuzione ottimale delle ore tra le diverse attività non solo evidenzia l'approccio metodologico adottato, ma sottolinea anche le priorità e gli impegni assunti nel corso del periodo di sviluppo. La tabella presentata di seguito fornisce un quadro dettagliato e accurato dell'allocazione temporale per ciascuna fase del progetto, mettendo in evidenza con trasparenza le specifiche aree in cui sono stati dedicati maggiori sforzi e risorse durante l'intero periodo di sviluppo.

| | **Settimane** | | | | | | | | | | **Ore** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | |
| **Incontro Stakeholder e Analisi** | X | | | | X | | | | | X | **30** |
| **Stesura relazione** | | X | X | X | | | | | | | **45** |
| **Formazione Blockchain** | X | X | | | | | | | | | **25** |
| **Formazione ZKP** | X | X | X | X | | | | | | | **30** |
| **Formazione Angular** | | | X | X | | | | | | | **20** |
| **Sviluppo backend** | | | | | X | X | X | | | | **80** |
| **Sviluppo frontend** | | | | | | | X | X | X | | **35** |
| **Sviluppo test di unità** | | | | | | | X | X | | | **35** |
| **Verifica e validazione** | | | | | | | | | X | X | **20** |
| **Totale** | | | | | | | | | | | **320** |

**Tabella 3.1:** Pianificazione del lavoro.

## 3.2 Analisi dei requisiti

### 3.2.1 Tracciamento requisiti

L'analisi dei requisiti costituisce una componente essenziale nell'architettura di un progetto, in quanto rappresenta la base su cui ogni fase successiva viene sviluppata e strutturata.

Una componente determinante nella nostra strategia di sviluppo è stata l'adozione della **metodologia agile**. Questo approccio, a differenza di metodologie più tradizionali, promuove un ciclo iterativo di sviluppo e *testing*, consentendo una maggiore reattività ai cambiamenti e alle nuove necessità che emergono nel corso del progetto. La natura incrementale della metodologia agile si basa sulla collaborazione tra diversi *team* e sulla continua interazione con gli *stakeholder*, permettendo di raffinare e adattare le specifiche in risposta ai *feedback* ricevuti.

Inoltre, con la sua enfasi sul rilascio frequente di versioni funzionanti del *software* e sulla valutazione regolare delle priorità, l'agile assicura che lo sviluppo sia sempre allineato con gli obiettivi del progetto e le aspettative degli utenti. In sintesi, la scelta dell'approccio agile ha fornito la flessibilità e l'adattabilità necessarie per affrontare e superare le sfide inerenti alla complessità del nostro progetto.

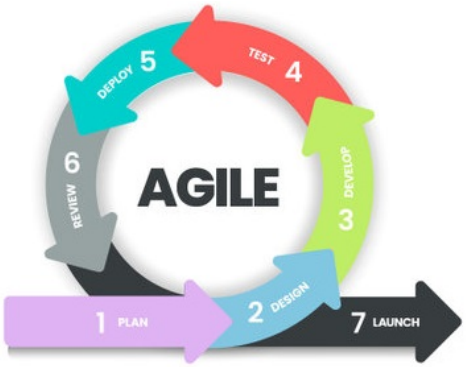



**Figura 3.1:** La metodologia agile
**Fonte:** Vecteezy

Gli incontri con il *tutor*, programmati con cadenza settimanale, hanno rappresentato momenti cardine nel processo di sviluppo e definizione del progetto, ogni sessione ha rappresentato un'opportunità per ricevere consigli, suggerimenti e *feedback*. Il *tutor*, grazie alla sua profonda conoscenza del settore e alla sua vasta esperienza pratica, non solo ha delineato le linee guida generali e messo in evidenza i requisiti chiave, ma ha anche proposto e consigliato l'utilizzo di specifiche tecnologie e strumenti, questo ha velocizzato il processo di sviluppo, riducendo notevolmente le lunghe fasi di ricerca.

A seguito di queste riflessioni, abbiamo potuto delineare chiaramente i requisiti del nostro sistema. I **requisiti funzionali** comprendevano:

- Effettuare una Votazione: Gli utenti, una volta collegati tramite *Metamask*, devono poter esprimere liberamente la loro preferenza.

- Creazione di Votazioni: Possibilità per gli utenti di iniziare nuove votazioni, definendo i parametri e i *validator*.
- Verifica della Votazione: Garantire trasparenza e verificabilità pubblica dei risultati.
- Singolarità del voto: Un meccanismo per assicurare che ogni utente effettui un'unica votazione.

Per quanto riguarda i **requisiti non funzionali**, abbiamo identificato:

- Privacy: Garantire l'anonimato dell'utente durante tutto il processo di votazione.
- Integrazione con Ethereum: Assicurare la piena compatibilità con gli *smart contracts* e le altre caratteristiche offerte dalla *blockchain*.
- Sicurezza: Offrire ulteriori livelli di sicurezza, oltre alla protezione naturale fornita dalla *blockchain*.
- Scalabilità: Gestire un alto numero di utenti e votazioni in simultanea.
- Affidabilità: Garantire un servizio senza interruzioni, assicurando che ogni voto venga registrato e verificato correttamente.

In conclusione, durante l'attività di analisi dei requisiti, abbiamo identificato **9 requisiti** per il nostro progetto: di cui 4 funzionali e 5 non funzionali.

Quest'attività si è rivelata un viaggio esplorativo, un'occasione per sondare le profondità del progetto, confrontarsi con le sfide e delineare le soluzioni. Questo percorso, arricchito dalla collaborazione con il *tutor* e dalla metodologia agile adottata, ha fornito un solido fondamento su cui costruire il nostro progetto, assicurando che ogni decisione fosse basata su una comprensione chiara e dettagliata delle esigenze e delle aspettative.

### 3.2.2 Procedura di voto

Durante la fase di progettazione, diverse metodologie relative alla procedura di voto sono state analizzate e valutate insieme al *tutor*. Il punto centrale di ciascuna di esse è stata la produzione di una "Ricevuta", utilizzata come prova di conoscenza zero (*ZKP*). Questa prova è fondamentale per assicurare l'integrità del processo di voto, garantendo simultaneamente l'anonimato dell'elettore.
Di seguito le tre principali metodologie analizzate:

- **Ricevuta personale**: In questa prima soluzione, l'utente riceve una Ricevuta strettamente associata al proprio indirizzo. Questa Ricevuta è unica per ogni indirizzo e non può essere utilizzata da altri. Sebbene questa soluzione offre una certa sicurezza, presenta dei problemi: l'utente deve ricordare l'indirizzo con cui ha generato la Ricevuta. Inoltre, esiste una potenziale vulnerabilità in cui un determinato indirizzo potrebbe essere correlato e riconducibile ad un specifico utente, mettendo a rischio l'anonimato.
- **Ricevuta configurabile**: Questa soluzione permette all'utente di scegliere autonomamente con quale indirizzo impiegare la Ricevuta, potendo, in teoria, utilizzare un indirizzo meno direttamente riconducibile alla propria identità.

Tuttavia, questo approccio non elimina la problematica riscontrata nella prima metodologia: l'utente deve comunque tenere traccia e ricordarsi l'indirizzo specifico scelto.

- **Ricevuta generica**: Questa terza soluzione, che è stata infine adottata, propone una Ricevuta che non è legata a un indirizzo specifico e può essere utilizzata da chiunque. Questa metodologia, pur presentando il potenziale rischio che gli utenti possano accumulare Ricevute di terzi, è particolarmente vantaggiosa per organizzazioni dove l'integrità del voto è prioritaria. I membri di tali organizzazioni tendono a valorizzare fortemente il proprio diritto di voto e sono meno inclini a cederlo. La natura generica della Ricevuta e la capacità di usarla con qualsiasi indirizzo assicurano che l'anonimato dell'utente sia preservato, evitando la divulgazione involontaria di dati sensibili come l'indirizzo principale.

  Quindi, pur accettando un maggior rischio di accumulo di Ricevute, abbiamo amplificato notevolmente la riservatezza e la protezione dell'identità del votante.

### 3.2.3 Blockchain Ethereum-Compatibile

La scelta di utilizzare una *blockchain* compatibile con *Ethereum* nel nostro progetto ha motivi ben precisi. *Ethereum* ha rivoluzionato il mondo della *blockchain* introducendo il concetto di *smart contract*, di cui abbiamo già parlato. Questi contratti digitali auto-eseguibili, con le regole dell'accordo direttamente scritte in codice, hanno permesso una vasta gamma di applicazioni decentralizzate.

Esistono anche alternative notevoli, piattaforme come *EOS, Tron, Cardano*, tra le altre, hanno introdotto i loro propri meccanismi di *smart contracts*, ognuna con particolari sfumature e differenziazioni. Tuttavia, nonostante la presenza di questi *competitor*, gli *smart contract* di *Ethereum* rimangono una scelta popolare, e ci sono ragioni chiare per questo.

Gli *smart contracts* vantano una vasta comunità di sviluppatori e un ecosistema maturo. Questo si traduce in una migliore documentazione, strumenti più raffinati e un maggior supporto per la risoluzione dei problemi.
Alcune critiche agli *smart contracts* di *Ethereum* riguardano le questioni di scalabilità, costi di transazione elevati durante i picchi di congestione, questioni però risolvibili utilizzando altre *blockchain*, *ethereum*-compatibili.

Infine, è importante notare che non ho selezionato una rete definitiva per il progetto, in quanto uno dei vantaggi delle *blockchain* compatibili con *Ethereum* è proprio la loro intercambiabilità. La decisione di non fissarmi su una rete specifica è stata intenzionale, non volevo impegnare risorse significative nella scelta della "rete perfetta", poiché il cuore del progetto non riguardava tale selezione.

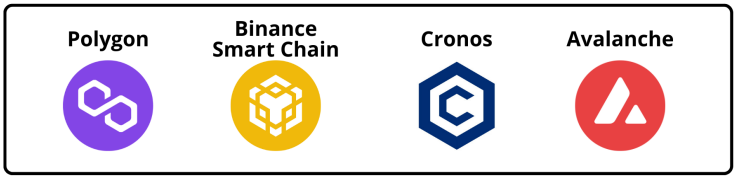



**Figura 3.2:** Esempi di reti Ethereum-compatibili

### 3.2.4 Framework per smart contract

Nel panorama della *blockchain*, la selezione di un *framework* per lo sviluppo degli *smart contracts* è un passo importante, anche se le differenze tra i principali *framework* sono spesso sottili. Mentre la scelta può avere un impatto marginale sull'efficienza dello sviluppo, essa può influenzare l'approccio, gli strumenti disponibili e la facilità d'uso durante il ciclo di sviluppo.

Diverse sono state le opzioni che ho individuato nel valutare i *framework* per gli *smart contracts*, tra le principali ci sono *Truffle, Remix e Hardhat*. Ciascuno di essi offre un insieme di strumenti specifici progettati per agevolare le fasi di creazione, test e distribuzione di *smart contracts* sulla rete *Ethereum*.

- **Truffle**: Uno dei *framework* più affermati nell'ecosistema *Ethereum*. Oltre alle funzionalità di *test* e migrazione, *Truffle* vanta una vasta libreria di risorse, una console interattiva che facilita l'interazione diretta con i contratti, e una rete di supporto ben consolidata. La sua lunga presenza nel campo ha anche portato alla creazione di una vasta comunità di sviluppatori, rendendo la risoluzione dei problemi e la condivisione delle migliori pratiche più accessibili.
- **Remix**: Un ambiente di sviluppo *open-source* basato su *browser* per lo sviluppo di *smart contracts* scritti in *Solidity*. È particolarmente apprezzato per la sua interfaccia utente intuitiva e la capacità di testare rapidamente *smart contracts* senza l'installazione di strumenti aggiuntivi. Tuttavia, essendo basato su *browser*, può non offrire la stessa potenza e flessibilità di soluzioni *desktop* più robuste, questo può limitare la gestione di progetti più grandi o complessi
- **Hardhat**: Il *framework* che ho scelto per guidare lo sviluppo di questo progetto. Si tratta di un *framework* relativamente nuovo, ma che si è rapidamente distinto per l'attenzione posta sull'esperienza dello sviluppatore. A differenza di altri *framework*, *Hardhat* è stato costruito con l'obiettivo di rendere ogni fase dello sviluppo di *smart contracts* più intuitiva e produttiva.

  Un punto di forza di *Hardhat* è senza dubbio l'"*Hardhat Network*", un ambiente *Ethereum* locale pensato appositamente per lo sviluppo. Questo *network* integrato consente di eseguire test, simulare il comportamento della rete e effettuare operazioni di *debug* in un ambiente altamente controllato, garantendo *feedback* immediato e un processo di *debug* più fluido.

  Ma quello che rende *Hardhat* particolarmente efficace è la sua natura didattica. Il *framework* offre una serie di *tutorial* completi che guidano passo dopo passo in ogni fase del progetto: dall'installazione e la configurazione dell'ambiente, alla scrittura e compilazione degli *smart contracts*, fino alla pubblicazione sulla *blockchain*. Questi *tutorial*, insieme alla documentazione ben strutturata, hanno reso la curva di apprendimento decisamente più agevole.


**Figura 3.3:** Framework per sviluppo *smart contract*

### 3.2.5 Tecnologie e protocolli per ZKP

Dopo un'approfondita analisi delle tecnologie disponibili nel campo delle *Zero-Knowledge Proofs (ZKP)*, ho identificato due protocolli fondamentali: *zk-SNARK e zk-STARK*. Questi protocolli rappresentano le fondamenta su cui molte delle tecnologie *ZKP* moderne sono costruite.

- **zk-SNARK**: Questo protocollo, il cui acronimo sta per *"Zero-Knowledge Succinct Non-Interactive Argument of Knowledge"*, è diventato popolare per la sua capacità di fornire prove concise che possono essere verificate rapidamente. Un elemento chiave che rende le *zk-SNARKs* particolarmente efficaci è l'uso dei *Merkle Trees*. L'importanza dei *Merkle Trees* risiede nel loro potere di garantire l'integrità dei dati. Tramite l'uso di *hash*, un *Merkle Tree* permette di verificare un singolo elemento di un *set* di dati (come una transazione in un blocco) senza dover controllare ogni singolo elemento. Questa proprietà lo rende ideale per le applicazioni *blockchain*, in cui l'efficienza e la scalabilità sono cruciali.

- **zk-STARK**: Acronimo di *"Zero-Knowledge Scalable Transparent ARgument of Knowledge"*. Le *zk-STARKs* si distinguono per vari vantaggi, tra cui trasparenza e scalabilità. Un elemento distintivo delle *zk-STARKs* è l'assenza di ciò che è conosciuto come "configurazione fiduciaria". In molti protocolli di *zero-knowledge*, come le *zk-SNARKs*, è necessaria una fase iniziale di configurazione in cui vengono generate chiavi pubbliche e private. Durante questo processo, è cruciale che la chiave privata venga eliminata al termine, altrimenti potrebbe essere utilizzata per compromettere l'intero sistema, generando prove false. Le *zk-STARKs*, al contrario, non necessitano di tale fase, eliminando quindi questo potenziale rischio. Nonostante ciò, a causa delle loro maggiori esigenze computazionali rispetto alle *zk-SNARKs* e alla minore disponibilità di librerie e protocolli che le supportano, essendo una tecnologia più recente, ho deciso di adottare le *zk-SNARKs* per il mio progetto.

La scelta del protocollo *zk-SNARK* ha aperto un panorama di tecnologie e librerie consolidate che la implementano e che si sono rivelate fondamentali per il successo del progetto.

- **Snarkjs**: Questa libreria, scritta in *Javascript*, rappresenta una delle implementazioni fondamentali di *zk-SNARK* nell'ambito delle *Zero-Knowledge Proofs*. Numerose altre applicazioni, come *Tornado Cash*, sfruttano *Snarkjs* come pilastro per gestire le *ZKPs*, testimoniando la sua affidabilità e versatilità. È stata essenziale nella nostra implementazione, in quanto abbiamo utilizzato *Snarkjs* per generare, gestire e verificare le prove *zk-SNARK*.

- **Circom**: Uno strumento specializzato per la creazione di circuiti *zk-SNARKs*. Abbiamo adoperato la sua libreria associata, *circomlibjs*, per gestire funzioni crittografiche avanzate, in particolare basate su *miMC*. Questo algoritmo di crittografia ci ha permesso di assicurare sia l'anonimato degli utenti che la sicurezza delle prove, risultando importante per la nostra implementazione

- **Tornado Cash**: Una delle principali soluzioni nel panorama di *Ethereum* per la realizzazione di transazioni anonime attraverso l'utilizzo delle *ZKPs*. Fondamentalmente, *Tornado Cash* si avvale delle tecnologie *zk-SNARK*, e fa uso specifico della libreria *snarkjs* come parte centrale del suo funzionamento. Al

cuore di *Tornado Cash* c'è *Tornado Core*, una libreria che rappresenta l'insieme delle componenti fondamentali del sistema. Da *Tornado Core*, ho selezionato e integrato specifiche classi per costruire il *Merkle Tree*. La decisione di sfruttare questi componenti deriva dalla comprovata efficienza e solidità di *Tornado Cash* in termini di sicurezza. Adottando queste classi, sono riuscito a implementare una struttura dati sia robusta che performante

### 3.2.6 Accesso alla Rete Ethereum

Per poter interagire e operare all'interno della rete *Ethereum*, è necessario avere accesso a un nodo *Ethereum*. Tuttavia, mantenere e gestire un nodo personale può essere rischioso e richiede risorse computazionali significative. Qui entrano in gioco servizi come *Infura* e *Alchemy*. Questi non sono solo intermediari, ma veri e propri ponti che facilitano l'interazione con la *blockchain Ethereum*, permettendo agli sviluppatori di focalizzarsi sullo sviluppo delle loro applicazioni piuttosto che sulla manutenzione di un'infrastruttura.

- **Alchemy**: Una piattaforma che si posiziona come un punto di riferimento per lo sviluppo di applicazioni *blockchain*. La sua principale forza risiede nell'alta affidabilità: ha un'*uptime* estremamente elevato, minimizzando così il rischio di interruzioni del servizio. Inoltre, fornisce strumenti di analisi avanzati che aiutano gli sviluppatori a monitorare e ottimizzare le loro applicazioni in tempo reale. Tuttavia, il *set* ricco di funzionalità di *Alchemy* comporta costi più elevati rispetto ad altre soluzioni. Inoltre, per chi è nuovo al suo ecosistema, la vasta gamma di opzioni e configurazioni può risultare complessa, richiedendo un periodo più ampio per comprenderne appieno tutte le potenzialità.

- **Infura**: D'altra parte, abbiamo *Infura*, un servizio consolidato nel panorama *Ethereum* e uno dei primi del suo genere. La sua longevità nel mercato ha permesso di guadagnarsi una solida reputazione e fiducia nella comunità. È progettato per la scalabilità, essendo in grado di gestire una vasta quantità di richieste, il che lo rende ideale per progetti che prevedono un'ampia base di utenti. Però, benché *Infura* offra un piano gratuito, presenta delle limitazioni che potrebbero non soddisfare le esigenze di progetti più ambiziosi. Tuttavia, tenendo conto delle mie esigenze specifiche, e considerando che offre un servizio gratuito di qualità, ho concluso che *Infura* fosse la soluzione più adatta per il mio progetto e ho deciso di adottarlo.

### 3.2.7 Metamask

Nel corso della progettazione del nostro sistema di votazione, una delle decisioni fondamentali riguardava l'interfaccia utente per la gestione dei portafogli digitali. Tale scelta ha un impatto diretto sull'esperienza dell'utente, sulla sicurezza e sull'efficienza dell'intero sistema. Su suggerimento del mio *tutor*, ho optato per l'adozione di *Metamask*, uno strumento che, nel panorama delle *criptovalute*, ha guadagnato notorietà come soluzione principale per la gestione dei portafogli.

*Metamask* funziona come un'estensione per browser e offre un'interfaccia intuitiva per gli utenti, permettendo loro di gestire facilmente i propri fondi in criptovalute e le relative *DApps*. Agendo come un ponte tra i *browser* e le *blockchain*, *Metamask* consente agli utenti di eseguire transazioni e di interagire con *smart contracts* in maniera semplice e sicura. L'utente può anche controllare le sue chiavi private direttamente dal suo dispositivo, garantendo così una protezione ottimale dei propri fondi.

Scegliendo di integrare *Metamask* nel nostro sistema, ho potuto ridurre notevolmente la complessità associata alla creazione e gestione di portafogli e indirizzi all'interno dell'applicazione. Questa scelta ha anche ridotto i potenziali rischi associati alla gestione diretta dei fondi e ha fornito agli utenti un'interfaccia familiare e affidabile con cui interagire.

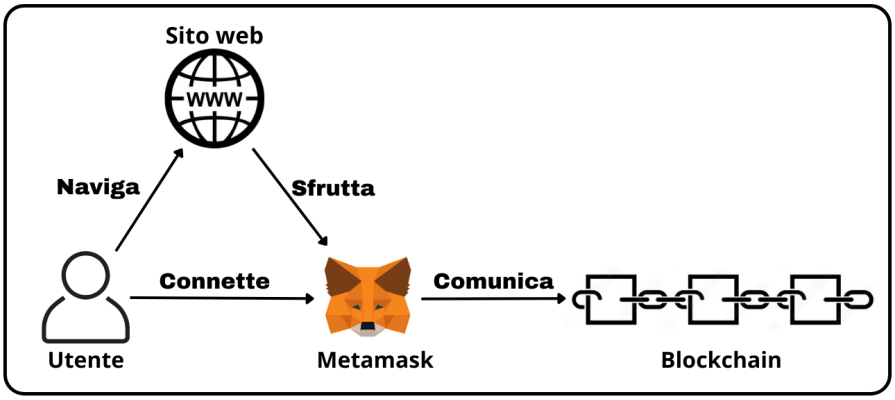



**Figura 3.4:** Metamask: Ponte verso la Blockchain

## 3.3 Sviluppo

### 3.3.1 Smart contract

Nel contesto del nostro sistema di voto basato sulla *blockchain*, lo sviluppo del contratto intelligente rappresentava l'elemento principale.

*Solidity*, essendo il linguaggio di programmazione per la creazione di *smart contract* su *Ethereum*, è stato naturalmente il nostro linguaggio di riferimento. Prima di addentrarci nello sviluppo specifico e nelle funzionalità dei contratti, è importante menzionare quali siano i contratti utilizzati all'interno del nostro sistema. Di seguito:

- **MerkleTreeWithHistory**
  Questo contratto è fornito da Tornado-Core, rappresenta una struttura di dati dell'albero di *Merkle* con la capacità di tenere traccia della sua storia. L'albero di *Merkle* è una struttura dati crittografica che consente di verificare la presenza di dati specifici in un set di dati senza rivelare l'intero set. Nella sua essenza, ogni foglia dell'albero è una *hash* di dati e ogni nodo padre è una *hash* delle sue foglie figlie. In questo caso, l'*hashing* è realizzato attraverso una funzione specifica, *MiMCSponge*, fornita dall'interfaccia *IHasher*. Inoltre, per tenere traccia della sua storia, memorizza e conserva la radici dell'albero ogni volta che viene modificata.

- **ZKTree**
  Questo contratto, fornito da *Tornado-Core*, estende le funzionalità di *MerkleTreeWithHistory* introducendo due concetti fondamentali per ogni sistema basato su *ZKP*: i *commitments* e i *nullifiers*.

  – Un **commitment** in questo contesto è come una promessa di un determinato valore (in questo caso, un voto) senza rivelare quel valore. È simile a una scatola chiusa: qualcosa viene posto al suo interno e sigillato, e mentre tutti possono vedere la scatola, il suo contenuto rimane nascosto. In un secondo momento, però, è possibile "aprire" questa scatola e mostrare il suo contenuto, confermando ciò a cui ci si era impegnati. Nel caso del sistema di voto, permette di "impegnarsi" a un voto senza rivelarlo immediatamente.
  – Un **nullifier**, invece, serve come una sorta di marchio o etichetta unica, garantendo che un voto specifico (o un *commitment*) non venga utilizzato più di una volta.

  Insieme, *commitment* e *nullifier* garantiscono che ogni individuo voti una sola volta e che il suo voto rimanga privato fino a quando non sceglie di rivelarlo.

- **ZKVote**
  Il contratto *ZKVote* effettua l'implementazione di un sistema di voto *ZKP*, ereditando da *ZKTree* per beneficiare delle strutture e delle funzionalità di base relative agli alberi *Merkle* e alle prove *ZKP*.

  Caratteristiche principali del contratto:

  – **Gestione dei Ruoli**: Il contratto stabilisce chiaramente i ruoli. C'è un proprietario (o creatore) del contratto, quindi della votazione, e ci sono validatori. I validatori sono responsabili della registrazione dei *commitments* e della gestione della *whitelist* degli elettori.
  – **Commitment**: Un aspetto centrale del contratto è la registrazione di *commitments* attraverso la funzione *registerCommitment*. Qui, un validatore

impegna un *hash* complesso (il *commitment*) associato a un elettore, senza rivelare informazioni specifiche sull'input originale dell'elettore. Questo *commitment* viene utilizzato successivamente quando l'elettore decide di votare, garantendo l'anonimato. In contemporanea, viene creato il *nullifier* e una prova, che si basano su quel *commitment*. Vediamo in dettaglio come vengono generati *commitment*, *nullifier* e prova, in sezione §.

- **Votazione**: La funzione *vote* consente a un elettore di sottoporre il suo voto in modo anonimo utilizzando *ZKP*. L'elettore invia il *nullifier* (che assicura che l'elettore non voti più di una volta) e le relative prove *ZKP* per dimostrare la validità del voto senza rivelare il contenuto effettivo del voto.
- **Interrogazione**: Il contratto fornisce metodi per ottenere informazioni sulla votazione in corso. Le funzioni come **getVoti, getOptions e getTitle** offrono dettagli sui voti attuali, le opzioni di voto e il titolo della votazione, rispettivamente.

- **ZKMapVote**
  Il contratto *ZKMapVote* funge da mappatura e gestore di diverse votazioni basate su *ZKVote*. Piuttosto che avere un singolo processo di votazione, questo contratto consente di gestire molteplici votazioni, ciascuna rappresentata da un'istanza separata di *ZKVote*.

  In pratica, *ZKMapVote* agisce come un registro o una *directory*, consentendo agli utenti di creare nuove votazioni, accedere alle votazioni esistenti e ottenere una visione complessiva dello stato delle votazioni in corso. Grazie a questa struttura centralizzata, gli utenti possono facilmente navigare tra diverse votazioni, mantenendo la trasparenza e la sicurezza garantite dal protocollo *ZKP*.

### 3.3.2 Angular

Il ***frontend***, nell'architettura di un sistema informatico, rappresenta il punto di interazione diretta tra l'utente e le complesse operazioni che avvengono "dietro le quinte". Nel contesto del nostro progetto, la creazione di un'interfaccia utente coerente, intuitiva e funzionale era imperativa per consentire una corretta interazione con le funzionalità offerte dal sistema. Tuttavia, essendo il progetto un *proof of concept*, il *focus* principale era la dimostrazione della fattibilità del sistema proposto, piuttosto che l'attrattiva estetica o la sofisticazione dell'interfaccia.

*Angular*, sviluppato e mantenuto da *Google*, è uno dei principali *framework* per lo sviluppo di applicazioni *web single-page* ed è stato scelto per lo sviluppo del nostro *frontend*. Si caratterizza per la sua modularità, permettendo di suddividere l'applicazione in componenti riutilizzabili, facilitando così sia lo sviluppo che la manutenzione. Inoltre, offre una serie di strumenti che accelerano e semplificano operazioni complesse, come la gestione dello stato, il *routing* e l'integrazione con API esterne.

Un altro aspetto fondamentale nella scelta di *Angular* è stato il suo supporto per *TypeScript*, che essendo un'estensione tipizzata di *JavaScript*, fornisce vantaggi in termini di sicurezza del codice, migliorando la leggibilità e prevenendo potenziali errori durante la fase di sviluppo.

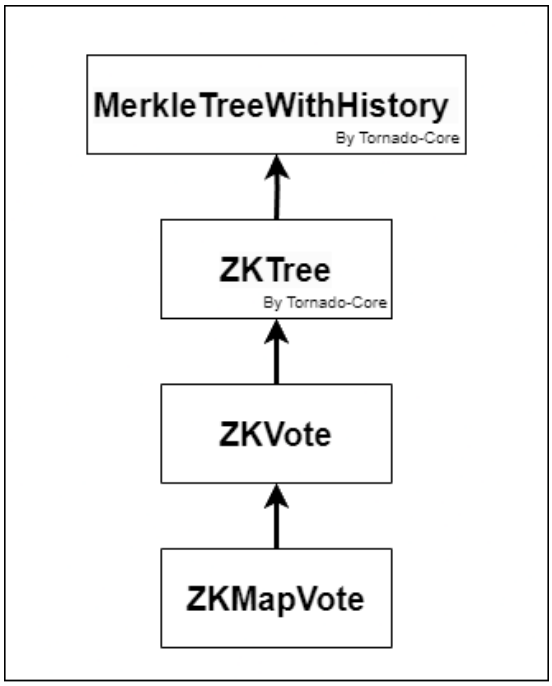



**Figura 3.5:** Gerarchia delle classi

Per quanto riguarda l'aspetto grafico dell'applicazione, si è optato per *Angular Material*. Questa libreria offre una vasta gamma di componenti predefiniti che aderiscono alle linee guida del *Material Design* di *Google*. La scelta di *Angular Material* ha garantito coerenza e uniformità nell'interfaccia, riducendo contemporaneamente il tempo necessario per definire e implementare dettagli stilistici. Tuttavia, è stato mantenuto un approccio minimalista al *design*, focalizzandosi sulla funzionalità e chiarezza piuttosto che sull'estetica avanzata.

### Struttura e componenti dell'applicazione

L'adozione di un approccio modulare nell'architettura del *frontend* è fondamentale per garantire sia una chiara distinzione delle responsabilità sia una facilità di manutenzione e aggiornamento. In questo contesto, l'interfaccia è stata articolata attraverso la suddivisione in diversi componenti principali. Questa segmentazione non solo ha permesso di isolare e gestire singolarmente le funzionalità distinte dell'applicazione, ma ha anche fornito agli utenti un'esperienza di navigazione intuitiva e logicamente strutturata.

Ogni componente è stato progettato e sviluppato con un obiettivo preciso: gestire un aspetto specifico del processo di voto, assicurando che ogni fase del processo, dalla creazione al conteggio dei voti, fosse rappresentata in modo chiaro e accessibile.

- **CreateVote**: Come suggerisce il nome, questo componente consente agli utenti di creare un nuovo voto. Questo modulo guida l'utente attraverso i passaggi

necessari per definire i dettagli del voto, come le opzioni disponibili e i parametri associati.

- **Vote**: Questo componente gestisce effettivamente il processo di voto. Permette agli utenti di selezionare un'opzione di voto e di inviare il loro voto in modo sicuro e anonimo.

- **ViewVote**: Questo modulo è stato progettato per consentire agli utenti di visualizzare i risultati di un voto. Offre una panoramica chiara e comprensibile delle opzioni di voto e del conteggio corrente dei voti.

- **Register**: Infine, il componente "Register" si occupa della registrazione degli utenti. È un passo essenziale per garantire che solo gli utenti autorizzati possano partecipare al processo di voto.

Questi componenti rappresentano la spina dorsale dell'interfaccia utente, garantendo un flusso di lavoro logico e coerente.

#### Comunicazione con la blockchain

Al di sotto di questi componenti, ci sono due servizi fondamentali che gestiscono la comunicazione e l'integrazione con la *blockchain* e il *backend*: **wallet.service e web3.service**.

- **wallet.service**: Questo servizio gestisce tutte le operazioni relative al portafoglio dell'utente. Dalla creazione di un nuovo portafoglio alla gestione delle chiavi private e delle transazioni, questo servizio assicura che tutte le operazioni relative al portafoglio siano sicure e trasparenti per l'utente finale.

- **web3.service**: Il servizio web3.service costituisce un elemento cruciale dell'architettura *frontend*, agendo come l'intermediario tra l'interfaccia utente e la *blockchain Ethereum*. Si avvale della libreria Web3.js, una scelta prevalente tra gli sviluppatori *Ethereum*, per instaurare e gestire la comunicazione con gli *smart contract*, consentendo l'esecuzione fluida di transazioni e l'interrogazione dei dati.

  L'essenza di questo servizio risiede quindi nella sua capacità di interagire con la *blockchain*. Inizialmente, definisce l'indirizzo dello *smart contract* e stabilisce una connessione *HTTP* con *Infura*, il servizio di nodo *Ethereum* utilizzato, utilizzando un *endpoint* designato. Questa connessione assicura che l'applicazione possa comunicare con la rete *Ethereum* senza ostacoli.

  Parallelamente, *web3.service* integra il *WalletService*, fondamentale per interfacciare l'applicazione con i portafogli digitali degli utenti. Questa sinergia consente agli utenti di collegare i loro portafogli digitali all'applicazione, facilitando l'autorizzazione e l'esecuzione delle transazioni sulla piattaforma. Attraverso questo servizio, l'utente può, ad esempio, creare nuove votazioni, visualizzare dettagli specifici delle votazioni esistenti, registrare validatori e *whitelist* per votazioni specifiche e, soprattutto, registrare i propri *commitment*, garantendo un processo di voto sicuro, privato e verificabile.

```
async newVotation(title : string, numOptions: number, opzioni : string[]){
  if (this.walletConnected) {
      let abi= require('contracts/ZKMapVote.sol/ZkMapVote.json')
      console.log("nuova");
      const contract = new this.web3WalletProvider.eth.Contract(abi.abi, this.contractAddress);
      await contract.methods.newVotation(
          20,
          "0x3034cD9FDE929139399743430dF5fe340E77308d",
          "0xE549DD626C1D1D50d9De5A9c2A452544aB978E4b",
          title,
          numOptions,
          opzioni)
      .send({from: this.address[0]})
      .on('transactionHash', function (hash: any) {
          console.log(hash);
      })
      .on('receipt', function (receipt: any) {
          console.log(receipt+"Done!");
      })
      .on('error', console.error)
  }else console.log("wallet not connected");
}
```

**Figura 3.6:** Funzione transazione per creare nuova votazione da smart contract

Il design di questi servizi è stato attentamente strutturato per garantire che ogni interazione con la *blockchain* sia ottimizzata e sicura
Le funzioni che gestiscono le transazioni inviate alla *blockchain*, come *newVotation* o *registerCommitment*, sono esempi notevoli di questa ottimizzazione. Queste non si limitano semplicemente a invocare metodi dello *smart contract*; piuttosto, incorporano un modello di programmazione asincrona attraverso l'utilizzo di promesse. Ciò garantisce che le funzioni non blocchino il flusso di esecuzione dell'applicazione, permettendo agli utenti di continuare ad interagire con l'interfaccia anche durante la pendente esecuzione di transazioni complesse.

Ancora più cruciale è l'implementazione di eventi in queste funzioni. Questi eventi sono progettati per fornire *feedback* in tempo reale agli utenti, tenendoli informati sullo stato delle loro transazioni. Ad esempio, quando un utente invia una transazione, viene notificato di un '*transactionHash*', che rappresenta un identificatore univoco per la transazione sulla rete *Ethereum*. Successivamente, alla conferma della transazione, riceve una ricevuta, assicurandogli che la transazione sia stata completata con successo. In caso di problemi o anomalie, gli eventi di errore sono pronti a catturare e comunicare tali incidenti all'utente, garantendo trasparenza e consapevolezza in ogni fase del processo.

### Generazione commitment e prove

Procediamo all'analisi di alcune funzioni fondamentali che rivestono un ruolo chiave nella gestione della *privacy* e dell'integrità dei dati. Queste funzioni non operano direttamente all'interno dello *smart contract* sulla *blockchain*, ma piuttosto nel *frontend* dell'applicazione.

Questo può sembrare controintuitivo, ma se avessi implementato queste funzioni direttamente all'interno del contratto sulla *blockchain*, ciò avrebbe comportato la necessità di effettuare ulteriori transazioni sulla *blockchain* stessa. Questo avrebbe potenzialmente aumentato notevolmente il tempo necessario per completare le operazioni e i costi associati, soprattutto considerando le variabili delle reti *blockchain* in termini di congestione e tariffe di transazione.

Pertanto, l'implementazione di alcune funzioni chiave nel *frontend* dell'applicazione consente una gestione più efficiente e rapida delle operazioni, senza compromettere la sicurezza complessiva del sistema. Va sottolineato che la *blockchain* rimane il punto centrale della sicurezza, in quanto tutte le operazioni critiche sono validate e garantite dalla logica dello *smart contract* sulla *blockchain* stessa. Il *frontend* svolge un ruolo complementare nell'ottimizzazione delle operazioni e nell'esperienza utente complessiva.

**Generazione del commitment**

La funzione *generateCommitment* è un componente cruciale all'interno del progetto, in quanto si occupa della generazione di un *commitment*. Tuttavia, essa svolge un ruolo ancora più ampio all'interno del protocollo, in quanto produce non solo il *commitment* stesso, ma anche tre elementi critici aggiuntivi: il *nullifier*, il *secret*, e il *nullifierHash*.

```
export async function generateCommitment() {
    const mimc = await circomlibjs.buildMimcSponge();
    const nullifier = BigNumber.from(crypto.randomBytes(31)).toString()
    const secret = BigNumber.from(crypto.randomBytes(31)).toString()
    const commitment = mimc.F.toString(mimc.multiHash([nullifier, secret]))
    const nullifierHash = mimc.F.toString(mimc.multiHash([nullifier]))
    return {
        nullifier: nullifier,
        secret: secret,
        commitment: commitment,
        nullifierHash: nullifierHash
    }
}
```

**Figura 3.7:** Codice generazione commitment

È importante notare come inizialmente vengano generati (pseudo)casualmente il *nullifier* e *secret*, mentre il *commitment* e *nullifierHash*, vengono generati tramite funzioni di *hashing*.

Innanzitutto, il *nullifierHash* non è altro che un *hashing* del *nullifier*.
Mentre il *commitment* viene creato a partire dal *nullifier* e un secret attraverso un processo di *hash*. Quindi, se si modifica il *nullifier*, il *commitment* cambierà di conseguenza, il che significa che un singolo *nullifier* può essere utilizzato solamente per un *commitment*.
Questo è possibile grazie alla natura unidirezionale dell'*hashing*, il che significa che non è possibile stabilire un collegamento diretto tra il *commitment* e il *nullifier*.

Tuttavia, possiamo generare una prova *ZKP* per garantire la validità dell'associazione tra il *nullifier* e il *commitment* senza rivelare le informazioni sottostanti.

**Generazione di una prova**

La fase successiva nel processo è la generazione della prova di validità per il *commitment* e la sua associazione con il *nullifier*. Questo passaggio è implementato attraverso una serie di funzioni interne, ma focalizzeremo la nostra attenzione sulla funzione principale denominata *calculateMerkleRootAndZKProof*.

Questa funzione svolge un ruolo critico e complesso all'interno del nostro sistema. Per comprendere meglio il suo scopo e il suo funzionamento, possiamo suddividerla in due parti fondamentali:

- **Recupero dei Commitment e Calcolo dell'Albero di Merkle** : La prima parte della funzione è responsabile del recupero dei *commitment* precedentemente registrati in un contratto specifico.

  Questi commitment, rappresentano degli hash di specifiche informazioni che sono state conservate in precedenza. Una volta ottenuti, è fondamentale assicurarsi della loro integrità, verificando che non siano corrotti o mancanti Successivamente, si procede con la creazione dell'albero di Merkle. Questa struttura dati organizza le informazioni in modo gerarchico, con ogni nodo rappresentato dall'hash della combinazione dei suoi nodi figli. I commitment vengono inseriti come foglie di questo albero. A partire da questi, viene calcolato l'hash di ciascun nodo superiore combinando gli hash dei nodi sottostanti. Questa operazione si ripete iterativamente fino a raggiungere la radice dell'albero.

  Una caratteristica essenziale dell'albero di Merkle è che la radice, o "Merkle root", funge da riassunto crittografico di tutti i commitment. Questa radice permette di verificare rapidamente l'esistenza di un determinato commitment nell'albero, senza la necessità di esaminare ogni singolo elemento. Al termine del processo quindi, ciò che si ottiene è non solo la Merkle root, ma anche una serie di percorsi crittografici che confermano la presenza dei singoli commitment nella struttura.

- **Generazione della Prova ZKP** : La seconda parte della funzione è ancor più interessante. Utilizzando la libreria *snarkjs*, genera una prova di conoscenza zero (*Zero-Knowledge Proof*) basata su diversi parametri cruciali:

  - **Nullifier**: Questo è il nostro segreto crittografico che vogliamo mantenere nascosto. Nel contesto delle Prove di Conoscenza Zero, il *nullifier* è cruciale poiché, pur essendo unico per ogni transazione o dato, non rivela direttamente nulla su di esso. Questo lo rende ideale per attuare procedure che necessitano di anonimato e segretezza, permettendo di dimostrare l'esistenza di una specifica transazione senza svelarne i dettagli
  - **Secret**: Si tratta di una chiave crittografica o un valore segreto associato specificamente al *commitment* originale. Mentre il *commitment* potrebbe rappresentare un dato o una transazione registrata e visibile, il *secret* rimane nascosto, garantendo la *privacy* dell'utente o della transazione. Nel contesto della generazione della Prova *ZKP*, il *secret* è uno dei parametri chiave, poiché fornisce la prova che l'entità conosce le specifiche dell'originale *commitment*, senza doverlo però rivelare
  - **Path**: Questo si riferisce al percorso specifico all'interno dell'albero di *Merkle*, delineando la sequenza di nodi da una determinata foglia (il nostro *commitment*) fino alla radice dell'albero. Questo percorso è essenziale in quanto consente la verifica dell'esistenza del *commitment* all'interno dell'albero senza doverlo esaminare in toto.
  - **Radice**: La radice dell'albero di *Merkle*, agisce come una sorta di "impronta digitale" crittografica per l'intero albero. Ogni cambiamento, anche minimo, in uno qualsiasi dei *commitment* all'interno dell'albero altererebbe questa radice, garantendo così l'integrità di tutte le informazioni conservate

La prova *ZKP* è quindi una sorta di "certificato crittografico" che dimostra la conoscenza del *nullifier* e del *secret* senza mai rivelarli direttamente. Questo è possibile grazie alla capacità di generare prove senza dover utilizzare il *commitment* stesso.

Inoltre, la funzione utilizza una chiave privata del sistema per garantire la validità della prova.

### 3.3.3 Analisi quantitativa

Nel vasto mondo dello sviluppo *software*, la dimensione e la complessità di un progetto possono variare notevolmente. Da piccoli *script* a complessi sistemi *enterprise*, ogni progetto porta con sé una sua unica serie di sfide e obiettivi. Una delle modalità con cui possiamo avvicinarci alla comprensione dell'enormità e della profondità di un progetto è attraverso un'analisi quantitativa. Questa analisi fornisce una panoramica numerica del lavoro svolto e può essere utilizzata come un termometro per valutare l'ampiezza e l'approfondimento del progetto in termini di sforzo di sviluppo.

Non solo le metriche quantitative offrono una misura tangibile dello sforzo e dell'impegno impiegato, ma possono anche fornire informazioni chiave sulla struttura e l'organizzazione del codice. Inoltre, mettendo in evidenza specifiche parti del sistema, ci permettono di comprendere meglio dove si sono concentrati gli sforzi e come le varie componenti interagiscono tra loro.

A tal proposito, di seguito sono mostrate due tabelle che contengono le principali metriche, suddivise tra *backend* e *frontend*. Questa suddivisione mira a fornire una chiara rappresentazione del lavoro svolto in ciascun segmento del sistema.

| Backend | |
|---|---|
| **Metrica** | **Valore** |
| Righe di codice | 586 |
| Numero di file sorgenti | 6 |
| Metodi implementati | 36 |

**Tabella 3.2:** Dati relativi all'implementazione del backend.

| Frontend | |
|---|---|
| **Metrica** | **Valore** |
| Righe di codice | 446 |
| Numero di file sorgenti | 10 |
| Metodi implementati | 35 |

**Tabella 3.3:** Dati relativi all'implementazione del frontend.

## 3.4 Verifica

### 3.4.1 Metodologie

L'attività di verifica, all'interno di questo progetto, ha avuto indubbiamente un'importanza cardine. L'evoluzione costante del panorama tecnologico, combinata con le particolarità e le complessità del settore della *blockchain* e degli *smart contracts*, ha posto l'accento sull'essenzialità di conformarsi rigorosamente ai requisiti delineati.

In un contesto in cui la correttezza dei dati, la protezione della *privacy* degli utenti e l'assicurazione di transazioni sicure sono prioritari, è imperativo che ogni elemento del sistema sia sottoposto a un'analisi accurata e meticolosa. Ogni funzione implementata e ogni comunicazione con la *blockchain* rappresentano elementi critici, ed è importante garantire la loro sicurezza. Mentre l'innovazione tecnologica è fondamentale per realizzare una soluzione di alto livello, è solo attraverso una rigorosa fase di verifica che si possono identificare e prevenire potenziali vulnerabilità.

La verifica assicura che il sistema operi come progettato, mantenendo gli *standard* di qualità stabiliti. Questo implica l'implementazione di meccanismi che assicurino la trasparenza dove necessario, la tutela della *privacy* degli utenti e la robustezza del sistema di fronte a possibili anomalie o attacchi. Sebbene la *blockchain*, per sua natura, fornisca molte di queste garanzie, la responsabilità di assicurare che ogni interazione, *smart contract* e interfaccia utente rispetti questi *standard*, ricade sugli sviluppatori e architetti del sistema.

**Metodologie di verifica:**

- **Approccio iterativo:** Mantenendo coerenza con la metodologia agile, abbiamo adottato una strategia di verifica iterativa. Dopo ogni *sprint* di sviluppo, un *set* di *test* mirati veniva eseguito per assicurarsi che le nuove implementazioni e modifiche fossero all'altezza degli *standard* richiesti.
- **Test di unità:** Usando la libreria *ethers*, offerta dal *framework Hardhat*, abbiamo potuto assicurarci che ogni parte dello *smart contract* fosse verificata in profondità, garantendo che ogni funzione e variabile rispondesse perfettamente alle aspettative. Questi *test* vengono trattati approfonditamente nella sezione §3.7.
- **Feedback continuo con lo stakeholder:** Il *tutor*, essendo uno degli *stakeholder* principali, è stato coinvolto in ogni fase del processo attraverso incontri settimanali. Questi incontri non erano solo occasioni di aggiornamento, ma anche momenti fondamentali per ricevere un *feedback* diretto sul lavoro svolto. Questo approccio ha permesso di ricevere *feedback* tempestivi, facilitando e rendendo più efficace la fase di verifica.
- **Test manuali e simulazione d'uso reale:** La combinazione di *test* automatizzati e manuali ha offerto una panoramica completa del comportamento del sistema, da un punto di vista sia tecnico che pratico.

### 3.4.2 Test di unità backend

La fase di *test* rappresenta un punto cardine in qualsiasi ciclo di sviluppo *software*. Questa assicura non solo che il *software* funzioni come previsto, ma anche che soddisfi tutti

i requisiti e le specifiche stabilite, fornendo così un prodotto robusto e affidabile. Nel contesto delle applicazioni basate su *blockchain*, la sua importanza è amplificata a causa delle caratteristiche intrinseche della tecnologia *blockchain*: l'immutabilità dei dati e le possibili conseguenze economiche o legali associate a qualsiasi errore o vulnerabilità.

Il *backend*, essendo il cuore dell'intera applicazione, è stato sottoposto a una serie di *test* approfonditi, specialmente riguardo alle operazioni crittografiche e ai meccanismi di interazione con la *blockchain*.

- **Generazione di Commitment**: I *commitment* rappresentano promesse crittografiche nel nostro sistema. È stato quindi essenziale verificare che questi fossero generati correttamente e in maniera sicura, mantenendo l'anonimato dell'utente e la validità del voto.

- **Generazione e Validazione delle Prove**: Dopo aver creato un *commitment*, il sistema è stato testato per assicurarsi che potesse generare prove a zero conoscenza valide e che potesse, al contempo, validare queste prove correttamente, garantendo la segretezza delle informazioni sottostanti.
  Sono stati anche simulati tentativi di creare e inviare prove false o alterate. Come atteso, il sistema ha rifiutato tali tentativi, confermando la sua sicurezza.

- **Interazioni con Smart Contracts e Gestione Voti**: Gli *smart contract* sono stati meticolosamente sottoposti a *test* per assicurare una risposta accurata alle invocazioni e una gestione idonea delle transazioni. Questo ha incluso l'invio di transazioni, la creazione di nuove votazioni, il processo di voto, la visualizzazione dei risultati e una dettagliata interazione con i *validators* e le *whitelist*.
  Durante queste verifiche, si è prestata particolare attenzione al fatto che solo entità autorizzate potessero interagire con le funzioni dedicate dello *smart contract*. L'invio di voti da parte di vari utenti è stato simulato, con l'obiettivo di confermare che i voti e i corrispondenti impegni fossero registrati e gestiti correttamente. Concluso il processo di voto, si è proceduto alla validazione dei risultati, garantendo la loro accuratezza e coerenza rispetto alle operazioni effettuate.

- **Resilienza e Gestione degli Errori**: In situazioni reali, gli errori possono accadere. Per questo motivo, il *backend* è stato testato anche per la sua capacità di gestire situazioni anomale o inattese, come interruzioni di connessione o dati non validi, garantendo una corretta gestione degli errori e la resilienza dell'intero sistema.

Durante il processo di sviluppo del *software*, ho dedicato una particolare attenzione alla fase di *testing*. Ho eseguito 7 *test* di unità per mettere alla prova il codice e verificare la sua correttezza e robustezza. È importante sottolineare che tutti questi *test* sono stati portati a termine con successo, confermando l'efficacia delle implementazioni e che il codice è stato testato completamente.

All'interno di questi 7 *test* di unità, alcuni di essi hanno incluso la valutazione di più funzionalità contemporaneamente, consentendo di esaminare in modo completo l'interazione tra le diverse componenti del *software*. Questo approccio mi ha permesso di individuare eventuali interazioni indesiderate e correggerle tempestivamente.

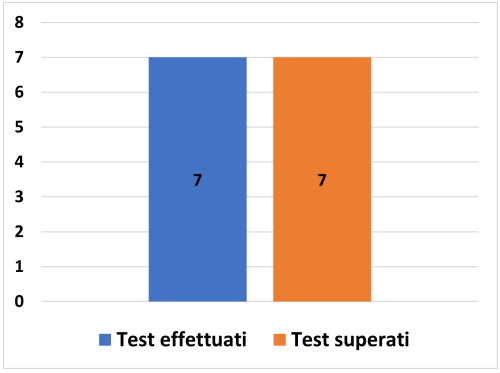



**Figura 3.8:** Istogramma relativo ai *test* implementati e superati

Durante l'esecuzione dei *test*, ho pianificato specificamente alcuni scenari che avrebbero dovuto produrre esiti positivi, confermando il funzionamento corretto delle funzionalità previste. Tuttavia, ho anche progettato alcuni *test* che avrebbero dovuto produrre esiti negativi, simulando situazioni limite o tentativi di aggirare le funzionalità del *software*. Pur essendo esiti negativi, questi *test* sono stati eseguiti per verificare la stabilità del sistema e garantire la sua resistenza ad eventuali tentativi di violazione o utilizzo improprio. Il fatto che questi *test* abbiano riportato esiti negativi non rappresenta un fallimento, ma una conferma che il *software* sta svolgendo correttamente il suo ruolo di protezione e sicurezza.

In conclusione, possiamo affermare con fiducia che tutti i *test* di unità hanno raggiunto gli esiti desiderati. Ciò significa che il codice è stato sottoposto a un'attenta verifica e ha dimostrato di essere affidabile, efficiente e resiliente. Questo processo di *testing* mi ha consentito di sviluppare un *software* solido e di qualità, pronto per affrontare le sfide del mondo reale.

### 3.4.3 Test frontend

A causa delle limitazioni temporali, non è stato possibile effettuare test automatizzati utilizzando strumenti specifici per Angular. Tuttavia, è importante sottolineare che la verifica manuale delle funzionalità è stata condotta in modo accurato.

- **Navigazione**: Verifica della corretta navigazione tra le diverse pagine e sezioni dell'applicazione.
- **Interattività**: Controllo delle funzioni interattive, come pulsanti, *form* di inserimento e *link*.
- **Comunicazione con la Blockchain**: Ogni volta che veniva effettuata una transazione o si interagiva con lo *smart contract*, si controllava che la comunicazione tra *frontend* e *blockchain* avvenisse senza intoppi e che i dati visualizzati corrispondessero alle attese.
- **Feedback Visivo**: Si è assicurato che l'utente ricevesse sempre un *feedback* visivo appropriato in risposta alle proprie azioni, come messaggi di conferma, errori o notifiche.

- **Performance**: Anche se non è stato effettuato un *test* di carico formale, durante la validazione manuale si è prestata attenzione alle *performance* dell'interfaccia, in particolare in relazione ai tempi di caricamento e di risposta.

### 3.4.4 Risultati

#### Test backend

Posso riferire che, dopo una dettagliata fase di *test*, tutti i casi più critici e rilevanti hanno prodotto l'esito desiderato. Ciò non significa che ogni possibile scenario sia stato esplorato, ma sicuramente ho coperto quelle situazioni e quei contesti che, per importanza e frequenza, avevano la priorità.

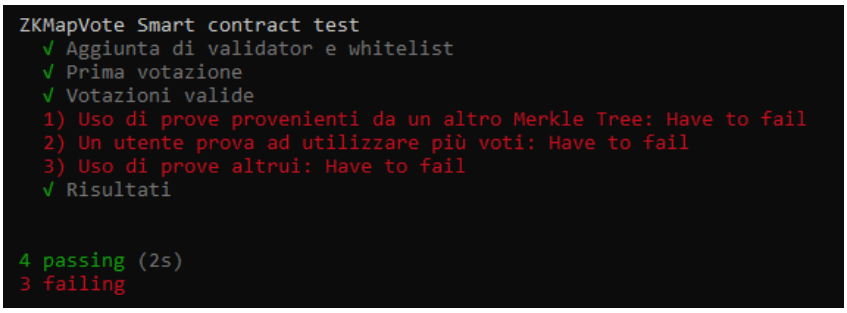



**Figura 3.9:** Risultati test unità degli smart contract usando Hardhat

Nonostante non abbia sondato ogni possibile scenario o circostanza immaginabile, mi sono concentrato con determinazione su garantire che le funzionalità principali e le operazioni più critiche fossero testate e funzionassero come atteso. Questo approccio ha permesso una valida e rigorosa verifica della qualità e della sicurezza del codice. È importante sottolineare che, sebbene la copertura dei *test* effettuati non raggiunga la totalità, i test condotti sono solidi e affidabili, offrendo una buona sicurezza nella gestione delle operazioni fondamentali dello *smart contract*.

Tuttavia, come in ogni sistema complesso, esiste sempre una minima possibilità di errori o comportamenti inattesi, specialmente in scenari rari o estremi. Questo non compromette l'efficacia complessiva del lavoro, ma piuttosto enfatizza l'importanza della continua verifica e miglioramento.

Va inoltre ricordato che questo progetto è stato concepito come un *PoC*. L'obiettivo primordiale non era la perfezione assoluta, ma piuttosto dimostrare la fattibilità del concetto proposto. La nostra focalizzazione sui casi più rilevanti e l'accuratezza con cui questi sono stati testati conferma la solidità e l'efficacia di questo *PoC*.

#### Test frontend

È importante sottolineare che, data l'assenza di *test* automatizzati sul *frontend*, non disponiamo di risultati quantitativi o metriche precise che attestino la performance o la robustezza dell'interfaccia. Gli unici "risultati" ottenuti derivano dalla validazione manuale, che, sebbene possa fornire un *feedback* immediato sull'esperienza utente e sulla funzionalità generale, non può quantificare con precisione eventuali limiti o potenziali aree di miglioramento.

Senza *test* strutturati e oggettivi, siamo privi di *benchmarks* o parametri di riferimento che possano essere utilizzati per future ottimizzazioni o revisioni. Tuttavia, le verifiche manuali effettuate durante lo sviluppo hanno garantito che le principali funzionalità dell'applicazione fossero operative e che l'interfaccia rispondesse alle aspettative in termini di interazione con l'utente.

## 3.5 Conclusioni

Nell'ambito di questo progetto, il nostro obiettivo primario era dimostrare la fattibilità della soluzione proposta, e possiamo affermare con sicurezza di averlo raggiunto. Il prototipo sviluppato, benché non completamente funzionante in tutte le sue parti, serve come solida base dimostrativa del potenziale del sistema.

Abbiamo identificato e superato numerose sfide durante lo sviluppo, ma la principale è stata relativa all'integrazione di certe librerie con Angular. La mancanza di file di definizione di tipo (.d.ts) per alcune delle librerie ha rappresentato un ostacolo significativo. Tuttavia, è importante sottolineare che ciò non mette in discussione la validità delle funzioni stesse. Infatti, pur avendo riscontrato problemi nell'integrarle nel *frontend*, abbiamo verificato la loro corretta esecuzione in altri contesti.

Tra i risultati principali del nostro sviluppo, possiamo delineare le seguenti funzionalità implementate con successo nel contesto del progetto:

- **Creazione di una votazione**
  Questa funzione consente a chiunque di creare una nuova votazione all'interno del sistema.
- **Gestione dei *validators*** L'"*owner*", ossia chi ha creato la votazione, può selezionare dei "*validators*" che hanno due compiti principali. In primo luogo, controllano il processo di votazione validando le prove presentate dai partecipanti. In secondo luogo, hanno l'autorità di aggiungere utenti alla *whitelist*, un elemento chiave del sistema.
- **Gestione della *whitelist*** *Validators* e *owner* possono aggiungere utenti alla *whitelist*. Questo è fondamentale per definire chi può partecipare alle votazioni e assicurare che il processo rimanga sicuro e controllato.
- **Votazione** Chiunque si trovi nella *whitelist* può accedere alla votazione e votare.
- **Visualizzazione dei risultati** Questa funzione permette a chiunque di accedere ai risultati delle votazioni. Offre trasparenza e accessibilità ai dati risultanti dal processo di votazione.

# Capitolo 4

# Conclusioni

## 4.1 Sviluppo e pianificazione a confronto

La gestione del tempo in un progetto, soprattutto di natura tecnologica e innovativa, rappresenta una delle sfide principali per ogni *team* di sviluppo. La pianificazione iniziale si basa su stime che cercano di prevedere il carico di lavoro necessario per ogni attività. Tuttavia, la natura dinamica dello sviluppo *software* e la poca esperienza sul campo, possono portare a variazioni significative tra le ore pianificate e quelle effettivamente impiegate.

Nel corso del nostro progetto, ho incontrato diverse sfide, imprevisti, ma anche momenti in cui sono riusciti ad essere più efficienti del previsto. Ad esempio, la formazione, ha richiesto più tempo del previsto, in particolare per approfondire argomenti come Angular. D'altro canto, la stesura della relazione ha richiesto meno tempo del previsto, grazie alla chiarezza degli obiettivi e alla struttura predefinita del documento.

Un altro esempio significativo riguarda lo sviluppo del *frontend* in Angular. Anche se avevo pianificato 35 ore, in realtà ne sono state necessarie 45. Gran parte di questo tempo aggiuntivo è stato causato da complicazioni nell'integrare le funzioni di generazione e verifica delle prove. Come precedentemente discusso, l'implementazione e l'interfacciamento di queste funzioni si sono rivelati più impegnativi del previsto.

Di seguito, presentiamo una tabella dettagliata che mostra il confronto tra le ore pianificate e quelle effettive per ogni attività svolta.

| Attività | Ore pianificate | Ore effetive |
|---|---|---|
| **Incontri con stakeholder e analisi** | 30 | 30 |
| **Stesura relazione** | 45 | 15 |
| **Formazione** | 75 | 100 |
| Blockchain | | |
| Solidity | | |
| Angular | | |
| **Sviluppo** | 170 | 165 |
| Progettazione e programmazione smart contract | 80 | 75 |
| Realizzazione del frontend in Angular | 35 | 50 |
| Sviluppo test di unità | 35 | 30 |
| Verifica e Validazione | 20 | 20 |
| **Totale ore** | **320** | **320** |

## 4.2 Copertura obiettivi

Quali obiettivi sono stati raggiunti e quali non.
Causa del perché alcuni non sono stati raggiunti.

## 4.3 Importanza delle tecnologie blockchain e zero knowledge proof per la votazione elettronica

Riflessioni riguardanti l'uso della Blockchain e ZKP per lo svolgimento di una votazione elettronica. Pro e contro.

Considerazioni sia personali che oggettive, riguardo le tecnologie Blockchain e ZKP. Sia utilizzate insieme, che prese singolarmente.

## 4.4 Conoscenze acquisite

Quali conoscenze considero acquisite.

## 4.5 Valutazione personale

# Appendice A

# Appendice A

Citazione

Autore della citazione

# Bibliografia