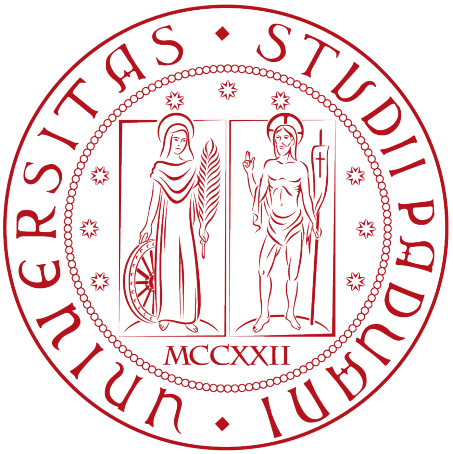
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Vouchy: validazione di dati senza terze parti con Ethereum

Tesi di laurea

Relatore

Dott. Michele Scquizzato

Laureando

Matteo Galvagni

Anno Accademico 2021-2022

Sommario

Ethereum è una piattaforma decentralizzata basata su tecnologia blockchain per lo sviluppo e la pubblicazione di smart contract creati con il linguaggio di programmazione nativo Solidity. La natura di Ethereum garantisce che tali contratti digitali siano eseguiti esattamente come originariamente scritti, consentendo l'esecuzione di codice di backend senza la necessità di dover prestare fiducia a terze parti. Il flusso d'esecuzione degli smart contract pubblicati su Ethereum è pubblicamente verificabile e chiunque ha la possibilità di sviluppare un'applicazione web per interfacciarsi ad un contratto o invocare i suoi metodi direttamente da terminale tramite uno dei molti client disponibili, volesse testarne il funzionamento o implementare un'interfaccia ad esso diversa da quelle già esistenti. Vouchy, la piattaforma discussa in questa tesi e realizzata nel periodo di stage della durata di 320 ore presso l'azienda Sync Lab S.r.l., è stata sviluppata con lo scopo di consentire la validazione di documenti (garantendo informazioni utili come data, ora e autore del caricamento) in modo trasparente e senza la necessità di dover affidarsi a servizi di terze parti, sfruttando l'immutabilità degli smart contract su Ethereum. Questo documento ne approfondirà i casi d'uso, il funzionamento e le scelte pratiche effettuate in fase di sviluppo.

"When life gives you lemons? Don't make lemonade. Make life take the lemons back!"

— Cave Johnson

Ringraziamenti

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

Desidero ringraziare con affetto la mia famiglia per essermi stati vicini in ogni momento durante gli anni di studio.

Ringrazio i miei amici, in particolar modo Alessandro, Sebastiano, Alex e Angelo, per il grande sostegno fornitomi durante il mio percorso di studi.

Infine, un ringraziamento speciale a Carlotta, per avermi supportato e sopportato nel corso di tutti questi anni.

Padova, Settembre 2022 Matteo Galvagni

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

In questo capitolo viene fornita una breve introduzione al progetto.

1.1 L'azienda

Sync Lab è un'azienda che da più di vent'anni produce soluzioni *IT*[g]in diversi settori quali sanità, finanza, logistica, telecomunicazioni e trasporti[1].

Dalla sua nascita (Napoli, 2002) è in continua espansione e può vantare ad oggi sei sedi in tutta Italia e oltre trecento dipendenti. Sync Lab collabora inoltre con diverse università italiane dal 2003.

1.2 Background

Al fine di una migliore comprensione è stato ritenuto necessario includere una sezione dedicata interamente a quello che è l'ecosistema *blockchain*[g].

Una blockchain è una struttura dati immutabile e condivisa, organizzata a blocchi di istruzioni (dette anche transazioni) concatenati in ordine cronologico, la cui validità è garantita da un meccanismo di consenso. Esistono più algoritmi di consenso e per la comprensione di questo documento non è ritenuto necessario approfondirli singolarmente; è sufficiente comprendere che un algoritmo di consenso in questo contesto consente ai nodi della rete di competere per includere nella catena i blocchi prodotti in cambio della generazione di moneta digitale nativa della rete, assicurandosi che blocchi contenenti istruzioni non consentite (come ad esempio invio di una quantità di moneta superiore a quella posseduta o invio di moneta da parte di un portafogli non posseduto) non vengano inclusi nella catena.

Con la nascita della blockchain Ethereum nel 2015 è stato introdotto un nuovo concetto in questo mondo: gli *smart contract*[g]. Uno smart contract non è altro che un software che, sfruttando l'immutabilità della blockchain, garantisce agli utenti che interagiscono con esso che l'esecuzione del codice sarà sempre uguale a come è stato originariamente implementato. Esso viene eseguito su una macchina virtuale appositamente sviluppata,

[1] *Sito web di Sync Lab.* URL: https://www.synclab.it/about.php.

detta Ethereum Virtual Machine. Una blockchain che utilizza la Ethereum Virtual Machine o un'estensione di essa per eseguire gli smart contract è detta *Ethereum-compatibile*.

Gli smart contract aprono così le porte ad un nuovo mondo di possibilità circa cosa è possibile sviluppare su una blockchain, passando dal "semplice" invio di moneta digitale senza intermediari allo sviluppo di complessi protocolli *trustless*[g]. Le implementazioni più significative di questa nuova tecnologia riguardano la finanza decentralizzata: diversi protocolli consentono di scambiare, dare o prendere in prestito token senza la presenza di terze parti ad arbitrare le operazioni ma affidandosi interamente a del codice pubblicamente verificabile ed immutabile. Questi token possono avere funzionalità particolari all'interno di alcuni protocolli o avere diverse rappresentazioni tra cui per esempio azioni di aziende o, come si vedrà in questa tesi, nomi di dominio.

Le transazioni nei blocchi possono essere quindi invii della moneta nativa della rete ma anche chiamate a metodi di contratti digitali o pubblicazioni degli stessi. Una transazione deve essere firmata utilizzando la coppia formata da chiave pubblica e chiave privata che rappresenta un portafogli; questa coppia è generabile tramite un algoritmo e di conseguenza non è legata all'identità di un utente.

Per facilitare la chiamata dei metodi nei contratti digitali è possibile sviluppare un'interfaccia grafica che dialoghi con gli stessi. Spesso quest'interfaccia è una webapp ed è sviluppata seguendo il "pattern" di una *dApp*[g](*Decentralized Application*), ovvero dialogando direttamente con lo smart contract senza ulteriore *backend*[g]che ne comprometterebbe l'effettiva decentralizzazione. Un pregio fondamentale di seguire questo pattern di sviluppo è che un'interfaccia grafica ad un contratto digitale è implementabile da chiunque, dunque possono esistere più interfacce allo stesso contratto e nel caso di problemi con un'interfaccia è sempre sviluppabile una nuova.

1.3 Introduzione al progetto

Il progetto di stage nasce dall'idea di implementare un servizio che consenta di accertare che un utente fosse in possesso di un certo documento in un dato momento senza ricorrere all'utilizzo di servizi di terze parti. In particolare, il progetto prevede l'utilizzo di una blockchain Ethereum-compatibile per la validazione trustless dei documenti. La soluzione scelta dovrà includere un'interfaccia di caricamento documento e una di validazione, nonché un modulo che consenta l'embedding di una pagina di validazione all'interno di altre pagine web.

La dApp prodotta sarà, per sua natura, interagibile sia direttamente da qualsiasi utente tramite un *frontend*[g], sia a basso livello da altri protocolli o smart contract che lo desiderano. Il risultato finale, tra le altre cose, consentirà a qualsiasi dApp di accertarsi e dimostrare che un utente (identificato dal suo indirizzo Ethereum) abbia ricevuto documenti a lui indirizzati, quali per esempio termini e condizioni di utilizzo della dApp stessa; il prodotto sarà dunque paragonabile ad un servizio di firma digitale completamente trasparente, senza terze parti che verifichino l'identità degli utenti e utilizzabile pubblicamente sia graficamente che programmaticamente.

1.4 Soluzione adottata

L'architettura ideale individuata per questo progetto è quella di una tipica dApp: la parte di frontend si occuperà di gestire le interazioni con gli utenti fornendo un'interfaccia grafica, mentre uno smart contract rappresenterà la parte di occupandosi delle verifiche di sicurezza, della profilazione e della memorizzazione dei dati necessari.

Si è scelto di sviluppare la parte di frontend tramite una WebApp realizzata con il *framework*[g] Angular, creato da Google, che consente uno sviluppo "a componenti" utile per questo progetto vista la necessità di riutilizzare in più pagine gli stessi elementi. La scelta di questo framework è stata dettata in gran parte dal grande utilizzo di esso all'interno dell'azienda. Si è inoltre deciso di consentire l'embedding della pagina di validazione tramite *iframe*[g], modificando tale pagina opportunamente.

È stato scelto di sviluppare lo smart contract su una testnet di Ethereum, Rinkeby, per compatibilità con qualsiasi blockchain Ethereum-compatibile e per la gratuità del servizio. Di conseguenza, il contratto è stato sviluppato utilizzando il linguaggio di programmazione nativo di Ethereum Solidity.

È stata posta particolare attenzione al metodo di comunicazione con la blockchain da parte della WebApp: al fine di garantire la maggior trasparenza possibile si è scelto di implementare più modalità di comunicazione, sceglibili dall'utente. All'interno della WebApp la comunicazione con la blockchain può quindi avvenire tramite *RPC*[g] a nodi della rete pubblici o privati, oppure tramite chiamate *API*[g] *REST*[g] ad un server appositamente sviluppato che si occuperà di comunicare con la blockchain tramite RPC ad un nodo privato. È stato scelto di sviluppare tale server utilizzando Spring Boot, un framework Java molto utilizzato dall'azienda.

1.5 Strumenti e tecnologie utilizzati

LaTeX: LaTeXè un linguaggio per la redazione di documenti ed è stato utilizzato per la stesura di ogni documento prodotto durante lo stage;

Git: Git è il più utilizzato sistema di versionamento ed è distribuito, gratuito e open source;

Typescript: Typescript è un linguaggio di programmazione che deriva da Javascript al quale è stata aggiunta la tipizzazione, viene compilato in normale codice Javascript;

Angular: Angular è un framework Typescript sviluppato da Google che permette la realizzazione di applicazioni web a "componenti", rendendo il riutilizzo di codice estremamente efficiente;

Java: Java è un linguaggio di programmazione ad oggetti ampiamente utilizzato che si appoggia sulla propria macchina virtuale per l'esecuzione;

Spring: Spring è un framework Java open source che consente il facile sviluppo di server in Java;

Spring Boot: Spring Boot è un'estensione del framework Spring che consente l'avvio della propria applicazione con un web server integrato;

Solidity: È il linguaggio di programmazione fortemente ispirato a Javascript nativo di Ethereum e funzionante su tutte le blockchain che condividono la macchina virtuale di Ethereum;

Truffle: Truffle è un ambiente di sviluppo per smart contract che offre diversi strumenti d'utilità quale per esempio la possibilità di avviare una blockchain locale;

1.6 Organizzazione del testo

Il secondo capitolo descrive la struttura dello stage e i suoi obiettivi prefissati.

Il terzo capitolo approfondisce l'analisi dei requisiti del progetto.

Il quarto capitolo illustra la fase di progettazione e codifica del progetto.

Il quinto capitolo descrive la fase di verifica e validazione del prodotto finale

Nel sesto capitolo si trovano le conclusioni circa l'esperienza di stage.

Sono state rispettate le seguenti convenzioni tipografiche circa la stesura del presente documento:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*[g];

Capitolo 2

Struttura dello stage

Il presente capitolo descrive la struttura dello stage e gli obiettivi prefissati.

2.1 Obiettivi prefissati

Gli obiettivi verranno d'ora in poi accompagnati da un codice nel formato *tipo-numero*. I possibili tipi degli obiettivi sono elencati di seguito:

* **O** - Obbligatori, ovvero necessari e richiesti dal committente per la corretta conclusione dell'esperienza;
* **D** - Desiderabili, cioè non strettamente necessari ma dal valore riconosciuto;
* **F** - Facoltativi, ovvero non necessari e dal poco valore aggiunto;

Obiettivi obbligatori

* *O-01*: Studio dei framework Java Spring e Angular, necessari allo sviluppo del prodotto;
* *O-02*: Sviluppo contratto digitale che memorizzi prove crittografiche di documenti;
* *O-03*: Sviluppo webapp che consenta inserimento di un documento salvandone un hash su contratto digitale;
* *O-04*: Sviluppo modulo inseribile in pagine web che consenta la verifica di un documento reperendo il relativo hash dal contratto digitale;

Obiettivi desiderabili

* *D-01*: Studio di fattibilità circa la possibilità di rendere il prodotto un'applicazione web immutabile e resistente a downtime tramite il deploy su servizio di hosting distribuito quale IPFS, Skynet o simili;

Obiettivi facoltativi

* *F-01*: Deploy effettivo della webapp su servizio di hosting distribuito scelto dallo studio di fattibilità dell'obiettivo D-01;
* *F-02*: Utilizzo di un risolutore di nomi di dominio in blockchain (ENS, UnstoppableDomains, ecc) per raggiungere la webapp da browser compatibili;

2.2 Pianificazione settimanale dello stage

* **Prima Settimana (40 ore)**
  - Incontro con stakeholders per discutere i requisiti e le richieste relative al sistema da sviluppare;
  - Formazione sul framework Java Spring;
* **Seconda Settimana (40 ore)**
  - Continuo formazione su Java Spring;
  - Integrazione formazione Java Spring mediante parziale Proof of Concept del prodotto da sviluppare.
* **Terza Settimana (40 ore)**
  - Formazione sul framework Angular;
* **Quarta Settimana (40 ore)**
  - Continuo formazione su Angular;
  - Integrazione formazione Angular mediante completamento del Proof of Concept del prodotto.
* **Quinta Settimana (40 ore)**
  - Eventuali correzioni del PoC se richieste;
  - Sviluppo contratto digitale e scrittura test di unità corrispondenti;
  - Inizio sviluppo backend prodotto finale utilizzando frontend del PoC e scrittura test di unità corrispondenti;
* **Sesta Settimana (40 ore)**
  - Continuazione e fine sviluppo backend;
  - Inizio sviluppo frontend prodotto finale e scrittura test di unità corrispondenti;
* **Settima Settimana (40 ore)**
  - Continuo sviluppo frontend;
* **Ottava Settimana (40 ore)**
  - Fine sviluppo frontend;
  - Verifica copertura test prodotto finale ed eventuali correzioni;
  - Validazione prodotto con stakeholders;

2.3 Analisi dei potenziali rischi

Come misura preventiva sono stati analizzati i possibili rischi riscontrabili durante l'esperienza, individuando già una soluzione. Di seguito un estratto di quest'analisi.

* **Nessuna conoscenza dei framework da utilizzare**: al fine di ovviare al problema sono state appositamente incluse nel piano di lavoro diverse ore dedicate alla formazione;
* **Documentazione di *Metamask*[g]non aggiornata**: è stato rilevato che esistono sufficienti esempi aggiornati per quanto riguarda le parti necessarie per il progetto, dunque quest'ultimo problema non sussiste;

2.4 Organizzazione dell'esperienza

Incontro settimanale
Almeno una volta alla settimana è stato svolto un incontro con gli stakeholders per aggiornarsi sullo stato di avanzamento del prodotto.

Smart working
La maggior parte del lavoro è stato svolto da remoto, tuttavia l'incontro settimanale si è sempre svolto in presenza.

Comunicazione
L'azienda ha fin da subito messo a disposizione un server su Discord per facilitare la comunicazione con dipendenti ed altri stagisti. Questo strumento si è rivelato particolarmente utile favorendo il confronto e velocizzando la risoluzione di problemi.

Capitolo 3

Analisi dei requisiti

In questo capitolo viene illustrata la fase di analisi dei requisiti effettuata.

3.1 Definizione casi d'uso

Come prima attività è stato effettuato un incontro con gli stakeholders per definire chiaramente quali fossero i casi d'uso circa il prodotto da sviluppare. Al fine di descrivere al meglio i casi d'uso si sono divisi gli utenti in due categorie, di seguito illustrate.

- **Utenti collegati con Metamask:** questi utenti possono sia inserire un documento per la sua futura validazione tramite apposita sezione, sia effettuare la validazione di un documento;
- **Utenti non collegati con Metamask:** questi utenti possono effettuare la validazione di un documento senza ulteriori azioni, tuttavia per inserire un documento viene loro richiesto di connettersi alla webapp con Metamask;

Queste categorie rappresentano i due casi particolari della più grande categoria "utente generico", e insieme a Metamask questi quattro saranno gli attori che compariranno nelle descrizioni dei casi d'uso di seguito riportate.

3.2 Casi d'uso

UC1: Accesso alla webapp

Descrizione: La webapp deve consentire agli utenti di collegarsi utilizzando Metamask nella sezione dedicata all'inserimento di un documento, poichè tale inserimento richiede l'invio di una transazione in blockchain.

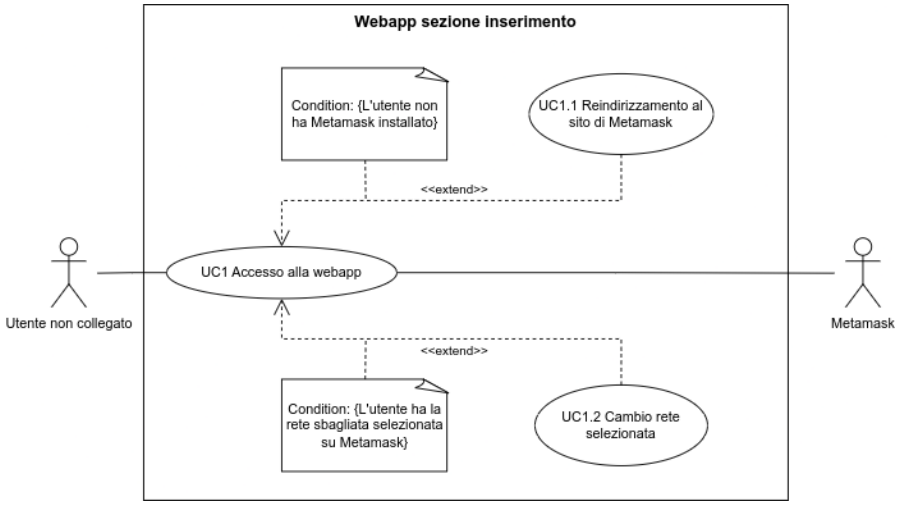


Figura 3.1: Use Case - UC1: Accesso alla webapp

Attori Principali: Utente non collegato, Metamask.

Precondizioni: L'utente accede alla sezione di inserimento documento.

Scenario principale: L'utente collega Metamask alla webapp tramite l'estensione stessa.

Postcondizioni: L'utente è ora collegato alla webapp con Metamask.

Scenario alternativo: *UC1.1* Reindirizzamento al sito di Metamask.

1. La webapp mostra un errore e un link per il reindirizzamento al sito di Metamask;
2. La webapp non permette l'inserimento di un documento;

Scenario alternativo: *UC1.2* Cambio rete selezionata.

1. La webapp mostra un errore e invita al cambio di rete tramite Metamask:
2. La webapp non permette l'inserimento di un documento;

UC2: Inserimento documento

Descrizione: La webapp deve consentire agli utenti collegati con Metamask di inserire un documento e ne deve poi caricare l'hash in blockchain per consentirne la futura validazione.

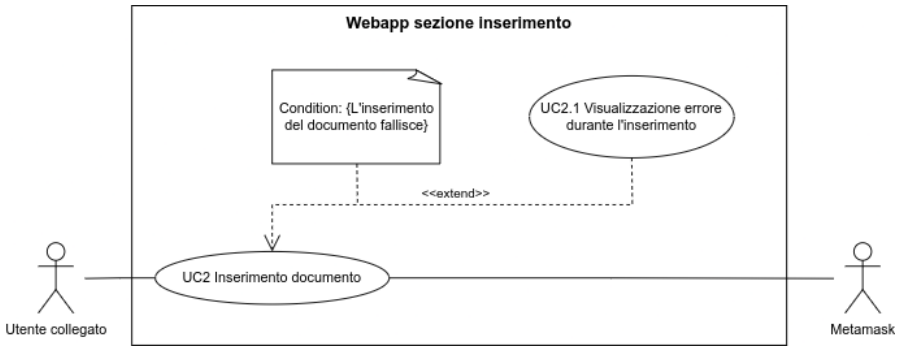


Figura 3.2: Use Case - UC2: Inserimento documento

Attori Principali: Utente collegato, Metamask.

Precondizioni: L'utente accede alla sezione di inserimento documento.

Scenario principale: L'utente inserisce un documento nella webapp, che ne carica l'hash in blockchain.

Postcondizioni: L'utente visualizza un riepilogo dell'inserimento, incluso il codice HTML per l'embedding.

Scenario alternativo: *UC2.1* Visualizzazione errore durante l'inserimento.

1. La webapp mostra un errore all'utente;
2. L'inserimento dell'hash non viene effettuato;

UC3: Validazione documento

Descrizione: La webapp deve consentire a tutti gli utenti di validare un documento inserendolo nella sezione apposita, restituendo le informazioni relative ad esso recuperate dallo smart contract.

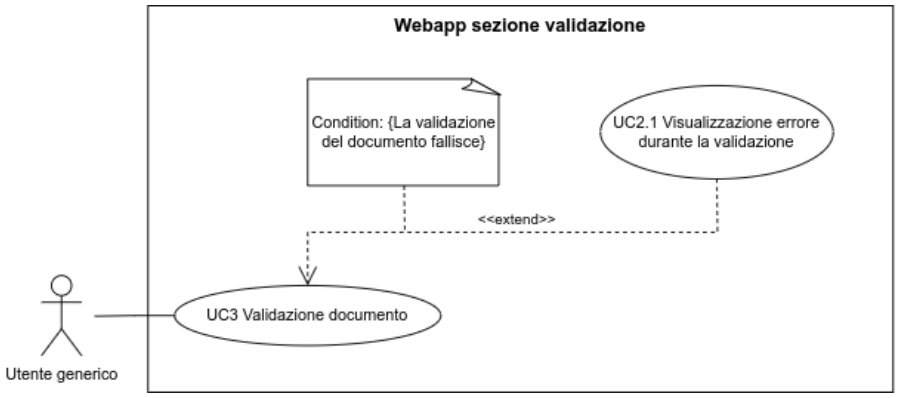


Figura 3.3: Use Case - UC3: Validazione documento

Attori Principali: Utente generico.

Precondizioni: L'utente accede alla sezione di validazione documento.

Scenario principale: L'utente inserisce un documento nella webapp, che ne ricerca l'hash in blockchain.

Postcondizioni: L'utente visualizza autore e timestamp formattato relativi al caricamento originale.

Scenario alternativo: *UC3.1* Visualizzazione errore durante la validazione.

1. La webapp mostra un errore all'utente;
2. La validazione del documento non viene effettuata;

3.3 Tracciamento dei requisiti

In seguito all'analisi dei casi d'uso si è proceduto alla fase di tracciamento dei requisiti. Per identificare i requisiti con dei codici si è utilizzata la seguente convenzione:

* la prima lettera è sempre **R** da "requisito";
* la seconda lettera rappresenta il tipo di requisito e può quindi essere:
  - **F** per requisito funzionale;
  - **Q** per requisito qualitativo;
  - **V** per requisito di vincolo;
* un numero progressivo che insieme al tipo identifica univocamente il requisito;

Nelle tabelle 3.1, 3.2 e 3.3 sono riassunti i requisiti e il loro tracciamento con i casi d'uso evidenziati in fase di analisi.

Tabella 3.1: Tabella del tracciamento dei requisiti funzionali

Requisito	Descrizione	Rilevanza	Fonte
RF1	Il sistema deve consentire il collegamento con Metamask	Obbligatorio	UC1
RF2	Il sistema deve reindirizzare l'utente sul sito di Metamask se lo stesso non fosse installato	Obbligatorio	UC1.1
RF3	Il sistema deve, utilizzando Metamask, consentire all'utente di cambiare rete selezionata	Obbligatorio	UC1.2
RF4	Il sistema deve consentire l'inserimento di un documento	Obbligatorio	UC2, UC3
RF5	L'inserimento del documento dovrebbe poter avvenire anche con drag&drop	Desiderabile	Committente
RF6	L'inserimento del documento dovrebbe poter avvenire anche tramite URL	Desiderabile	Committente
RF7	Il sistema deve essere in grado di calcolare l'hash del documento con una funzione scelta	Obbligatorio	UC2, UC3

Requisito	Descrizione	Rilevanza	Fonte
RF8	Il sistema deve essere in grado di far firmare transazioni all'utente tramite Metamask	Obbligatorio	UC2
RF9	Il sistema deve mostrare chiaramente all'utente gli eventuali errori in fase di caricamento del documento	Obbligatorio	UC2.1
RF10	Il sistema deve mostrare chiaramente all'utente gli eventuali errori in fase di validazione del documento	Obbligatorio	UC3.1
RF11	La webapp deve utilizzare delle RPC per reperire informazioni dalla blockchain	Obbligatorio	Committente
RF12	Lo smart contract deve consentire la memorizzazione di un hash sottoforma di stringa	Obbligatorio	UC2, UC3
RF13	Lo smart contract deve memorizzare autore e timestamp relativi al caricamento di un hash	Obbligatorio	UC3
RF14	Lo smart contract deve essere pubblicamente accessibile	Obbligatorio	Committente
RF15	La webapp deve restituire all'utente del codice HTML per l'embedding in altri siti web	Obbligatorio	UC2
RF16	La webapp deve consentire il passaggio di un documento anche tramite URL nel parametro *GET*[g]	Obbligatorio	Committente

Tabella 3.2: Tabella del tracciamento dei requisiti qualitativi

Requisito	Descrizione	Rilevanza	Fonte
RQ1	Devono essere presenti test di unità per lo smart contract	Obbligatorio	Committente
RQ2	Devono essere presenti test di unità per il backend	Desiderabile	Committente
RQ3	La webapp deve essere accessibile a tutte le categorie di utenti	Obbligatorio	Committente
RQ4	La webapp deve essere responsive e adattandosi al dispositivo	Obbligatorio	Committente
RQ5	La webapp deve funzionare correttamente anche con il browser integrato dell'app Metamask per Android	Obbligatorio	Committente

Tabella 3.3: Tabella del tracciamento dei requisiti di vincolo

Requisito	Descrizione	Rilevanza	Fonte
RV1	Deve essere utilizzato Angular per lo sviluppo del frontend	Obbligatorio	Committente
RV2	Deve essere utilizzato Java Spring per lo sviluppo del backend	Desiderabile	Committente
RV3	Deve essere utilizzato Metamask come portafoglio	Obbligatorio	Committente
RV4	Deve essere utilizzata una blockchain Ethereum-compatibile	Obbligatorio	Committente

Capitolo 4

Progettazione e codifica

Questo capitolo tratta la progettazione e lo sviluppo del prodotto finale.

4.1 Formazione sulle tecnologie

Di seguito una panoramica delle tecnologie e degli strumenti utilizzati non precedentemente conosciuti.

Typescript

Linguaggio di programmazione che aggiunge i tipi al linguaggio già conosciuto Javascript. Data la grande mole di documentazione esistente e la previa conoscenza di Javascript non ha rappresentato un problema nella fase di formazione.

Angular

Un framework completo e molto complesso, ricco di concetti mai affrontati. Grazie alla documentazione eccellente e alla presenza di diversi tutorial sul sito di Angular, la fase di formazione circa il si è svolta senza particolari problemi. Tuttavia, la fase di studio di Angular ha occupato la maggior parte del tempo dedicato alla formazione.

Solidity

Linguaggio di programmazione nativo di Ethereum, è fortemente ispirato a Javascript e per questo non ha rappresentato un grande ostacolo.

Truffle

È la suite di utilità scelta per lo sviluppo dello smart contract. Nonostante l'ottimo ambiente di sviluppo che fornisce alcune funzionalità tra cui in particolare la scrittura di test di unità non sono ben documentate e richiedono ricerche di esempi aggiuntivi.

Spring Boot

Estensione del framework Java Spring è molto ben documentato e, visto anche i pochi argomenti cui fosse necessario lo studio per questo progetto, non ha rappresentato un problema in fase di formazione.

Metamask

Il funzionamento particolare di Metamask, basato sull'injection di codice Javascript all'interno di una pagina web, ha richiesto tempo per dello studio dedicato. La documentazione di questo software è risultata incompleta di sufficienti esempi; tuttavia quella presente indica chiaramente che l'utilizzo di una libreria terza di alto livello per gestire la comunicazione con esso è preferibile.

Web3js e Web3j

Librerie di alto livello (rispettivamente Javascript e Java) per l'interazione con blockchain compatibili con Ethereum, forniscono metodi molto comodi rispetto a quelli per esempio offerti da Metamask che si basano su chiamate dirette a metodi della blockchain.

4.2 Progettazione interfacce

Vista la semplicità grafica delle interfacce da produrre il committente non ha ritenuto necessario sviluppare un mockup vero e proprio, si è piuttosto optato per una descrizione informale basata su quello che sarebbe stato il normale flusso di operazioni degli utenti, illustrato di seguito. Decisi i componenti grafici fondamentali e la loro posizione approssimativa, è poi iniziata la fase di sviluppo vera e propria.

4.2.1 Flusso inserimento documento

1. l'utente entra nella sezione di inserimento documento;
2. l'utente si collega al sito con Metamask;
3. l'utente inserisce un documento scegliendo una delle seguenti opzioni:
   * drag&drop;
   * selezione file dalla memoria del dispositivo;
   * inserimento URL;
4. l'utente conferma l'inserimento tramite apposito pulsante;
5. l'utente firma la transazione proposta da Metamask;
6. viene visualizzato un riepilogo dell'inserimento;

4.2.2 Flusso validazione documento

1. l'utente entra nella sezione di validazione documento;
2. l'utente inserisce un documento scegliendo una delle seguenti opzioni:
   * drag&drop;
   * selezione file dalla memoria del dispositivo;
   * inserimento URL;
3. l'utente conferma la validazione tramite apposito pulsante;
4. viene visualizzato un riepilogo comprensivo di autore e timestamp;

4.3 Struttura smart contract

Lo smart contract sviluppato è molto semplice ed è composto da due soli metodi che consentono rispettivamente di inserire un hash e di leggere le informazioni relative all'inserimento di un hash. Le informazioni che vengono memorizzate (autore e timestamp del caricamento) sono organizzate in una *struct*, struttura dati che consente di raggruppare più variabili insieme. Tali coppie di dati sono memorizzate all'interno di una mappa che utilizza l'hash del documento come chiave per accedervi: questa scelta è stata effettuata per una serie di motivi:

* in Solidity le mappe sono implementate come hash table, e hanno quindi un tempo di accesso di O(1) al contrario di un normale array che avrebbe O(N);
* nel contesto della blockchain esiste un limite di "gas" (unità di moneta digitale pagate come commissione di rete per inviare una transazione) per blocco che non può essere superato da protocollo. Dato che ogni operazione ha un costo in gas, scorrere un array troppo grande potrebbe costare di più del limite esistente, rendendo il contratto essenzialmente inutilizzabile;

Per queste ragioni è stato deciso di non utilizzare array, con il possibile svantaggio che un documento potrà essere caricato da un solo utente. Questo è però facilmente raggirabile, poichè è sufficiente "personalizzare" un documento (ad esempio aggiungendo l'indirizzo del wallet dell'utente) per generare un hash diverso e consentirne l'inserimento. Inoltre, risulta più intuitivo validare un documento inserendo il solo documento ed ottenendo le informazioni relative; se un documento potesse essere inserito da più utenti in fase di validazione sarebbe necessario fornire anche l'indirizzo di uno dei tanti autori del caricamento per ottenere il timestamp, idea giudicata scomoda e controproducente in accordo con il committente.

4.4 Struttura backend

La struttura del backend realizzato è anch'essa molto semplice: implementando il design pattern MVC, un controller si occupa di rispondere alle richieste ricevute dal server con i dati reperiti dal model. Più precisamente, utilizzando la libreria Web3j il modello che funge da wrapper dello smart contract si occupa di interagire con lo stesso, mentre il controller si occupa di rispondere alle richieste con i risultati reperiti in formato standard JSON; per le necessità di questo progetto è stata implementata una sola chiamata API che, dato un hash passato come parametro GET, restituisce autore e timestamp ottenuti chiamando l'apposito metodo presente nel contratto.

4.5 Struttura frontend

La struttura del frontend è la più complessa ed occuperà la maggior parte di questo capitolo.

4.5.1 Componenti

Come prima cosa si sono individuati i componenti in cui suddividere la webapp. Lo sviluppo a componenti non solo favorisce la manutenibilità del prodotto ma consente anche il riutilizzo degli stessi, estremamente utile in questo progetto. In Angular i componenti sono moduli composti da:

* HTML, che detta la struttura del componente nella pagina web;
* CSS, che gestisce la grafica del componente;
* Typescript, che gestisce il comportamento del componente;

È importante sottolineare che questi componenti possono essere composti da altri componenti figli, dunque esiste una gerarchia tra di essi e sono presenti delle metodologie per farli comunicare.

I componenti individuati sono i seguenti:

* **Bottone Metamask:** è il bottone che consente all'utente di connettersi alla webapp con Metamask e sarà presente solo nella sezione dedicata all'inserimento del documento, visto che non è necessario essere collegati in fase di validazione;
* **Header:** è il componente in testata a tutte le pagine che include i link di navigazione nella webapp;
* **Preview:** gestisce la preview del file inserito;
* **Drag&drop:** gestisce il drag&drop di file;
* **Caricamento file:** gestisce il caricamento di un file e include al suo interno i componenti "drag&drop" e "preview";
* **Componente inserimento:** gestisce l'inserimento dell'hash in blockchain previo caricamento del documento stesso, ed comprende i componenti "caricamento file" e "bottone Metamask";

* **Componente validazione:** gestisce la validazione di un documento previo il suo caricamento, ed comprende il componente "caricamento file";
* **Router:** è un componente proprio di Angular che funge da placeholder per "componente inserimento" o "componente validazione", ed ha lo scopo di sostituirsi con uno di questi due componenti in base alla posizione di navigazione dell'utente;
* **App component:** è il componente principale di Angular ed è richiesto dal framework, è composto dal componente "header" e dal componente "router";

Nota. Il routing in Angular è una modalità standard per mostrare i componenti corretti in base alla pagina in cui l'utente si trova. Funziona molto comodamente creando un componente "router" che funge da placeholder per i possibili componenti da visualizzare, per poi impostare nella configurazione del progetto le varie routes e a cosa puntano.

4.5.2 Services

In Angular i servizi sono classi particolari che si occupano di fornire utilità ai componenti come reperire e restituire dati, interfacciarsi con servizi esterni o semplicemente far comunicare due o più componenti non parenti stretti. Essi vengono iniettati nelle classi che li utilizzano tramite il costruttore grazie al dependency injector di Angular, che ne mantiene una sola istanza senza la necessità di implementare il design pattern singleton. Di seguito vengono descritti i servizi utilizzati in questo progetto:

* **Metamask:** gestisce tutte le comunicazioni con Metamask e quindi le interazioni con la blockchain;
* **Keccak:** è il servizio che si occupa di calcolare e restituire l'hash di un documento tramite la funzione keccak256;
* **Backend:** si occupa di effettuare le chiamate al backend Java e restituirne il risultato ai componenti;
* **Download file:** dato un URL, si occupa di scaricare tale file per il calcolo dell'hash e la preview;
* **Estensioni accettate:** servizio di utilità, fornisce le estensioni dei file supportati a tutti i componenti che lo richiedono;

Nota. Si è scelto di utilizzare la funzione keccak256 i seguenti motivi:

* è la funzione di hash nativa di Ethereum, dunque la documentazione è molto buona;
* è utilizzabile nativamente anche all'interno di smart contract, rendendo possibili espansioni future più facili;
* questa funzione ha uno spazio degli input potenzialmente infinito, dunque non impone limiti di grandezza dei file o la frammentazione di essi;

4.5.3 Design pattern implementati

Sono stati utilizzati più design pattern, principalmente integrati in Angular, per la realizzazione della webapp. Di seguito un riassunto dei più significativi.

* **Decorator**
  È un pattern che aggiunge funzionalità ad un oggetto dinamicamente. In Angular viene utilizzato per aggiungere metadati alle classi, per esempio i decoratori @Output e @Input consentono a delle variabili di essere comunicate tra componenti padre-figlio.

* **Dependency injection**
  Questo pattern si basa sull'inserimento automatico delle dipendenze di un oggetto in fase di costruzione di un altro oggetto. La dichiarazione degli oggetti da iniettare è eseguita nel costruttore, rendendo facile il tracciamento delle dipendenze.

* **Singleton**
  È un pattern che garantisce l'esistenza di una sola istanza di un oggetto, condivisa per tutto il programma.

* **Service**
  Pattern utilizzato molto spesso all'interno del progetto, consente l'accesso alle classi services da parte di tutti i componenti. Iniettati all'interno delle classi che li utilizzano tramite il costruttore.

* **Obeserver**
  Questo pattern consente di rimanere in ascolto di eventi emessi da un'altra classe ed è uno dei pattern principali per la programmazione asincrona. Angular gestisce tutte le richieste HTTP con questo pattern.

4.6 Codifica frontend

4.6.1 Introduzione

La webapp è sviluppata per fornire la pagina di inserimento come pagina di default; tale pagina, come la pagina di validazione, include l'header con i link per la navigazione all'interno della webapp. Tutte le pagine rispettano la stessa impostazione grafica, e riutilizzano i componenti di Angular dove possibile. La webapp supporta tutte le estensioni di file maggiormente utilizzate, tra cui anche estensioni audio e video.

4.6.2 Pagina di inserimento

Al primo accesso l'utente non sarà collegato con Metamask dunque, assumendo che esso sia installato, viene visualizzato il pulsante che ne consente il collegamento. Viene visualizzato il componente di inserimento del documento, ma finchè l'utente non si collega con Metamask il pulsante di inserimento non viene attivato.

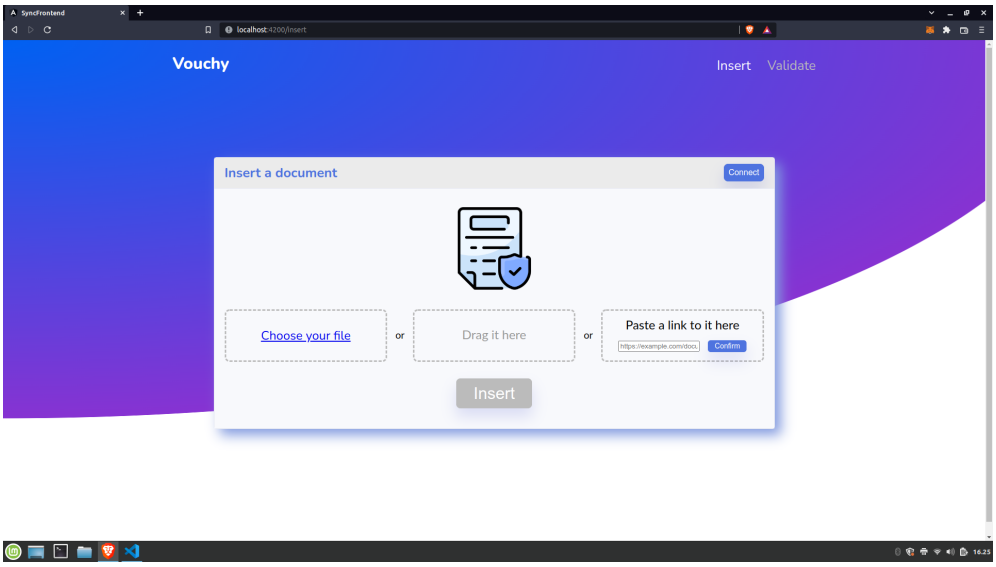


Figura 4.1: Pagina di inserimento - utente non connesso

L'inserimento del documento può avvenire tramite selezione del documento da memoria del dispositivo, drag&drop o inserimento URL. La preview del documento viene effettuata fino al limite massimo di 1.5MB per non rallentare l'esperienza dell'utente. Diversi errori possono venire mostrati in fase di inserimento; il più significativo tuttavia può essere visualizzato solo utilizzando l'inserimento tramite URL e riguarda l'impossibilità di raggiungere il documento a causa delle politiche CORS del dominio di appartenenza del documento. Infatti, in questo caso è necessario che il server che fornisce il documento permetta il raggiungimento da un altro dominio.

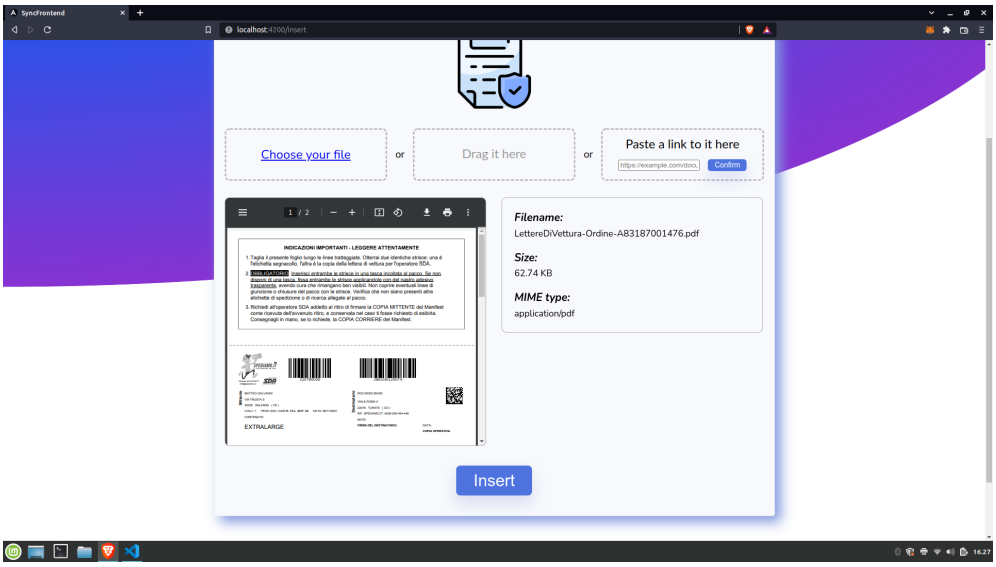


Figura 4.2: Pagina di inserimento - preview del file

La preview include anche un riassunto sulle informazioni del file caricato, tra cui nome e peso di quest'ultimo. Essa viene visualizzata ovviamente solo dopo il corretto inserimento di un file. A questo punto, assumendo un utente collegato con Metamask, il pulsante di inserimento è cliccabile e se cliccato (previa firma della transazione generata con Metamask) la webapp mostra nel caso di successo un riepilogo dell'inserimento, includendo il codice HTML richiesto per l'embedding in altre pagine web.

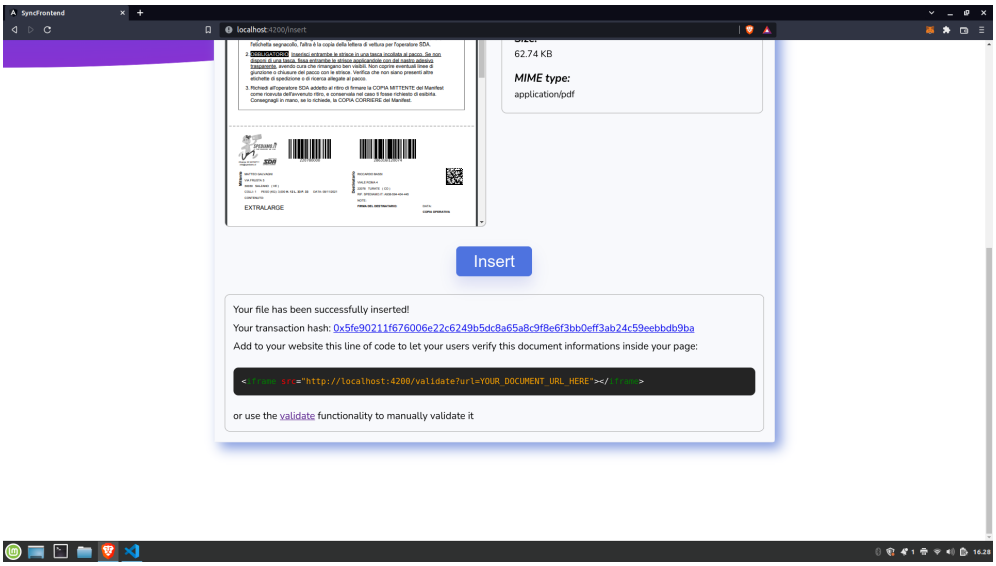


Figura 4.3: Pagina di inserimento - inserimento avvenuto

4.6.3 Pagina di validazione

In questa pagina non è necessario che l'utente sia collegato con Metamask, dunque il pulsante relativo è rimosso. Viene invece riutilizzato il componente di inserimento documento presente nella pagina di inserimento, insieme alla preview.

In seguito alla pressione del pulsante di validazione, autore e timestamp formattato dell'inserimento originale vengono restituiti. Queste informazioni vengono reperite dal contratto digitale tramite RPC oppure tramite chiamata API al backend in Java.

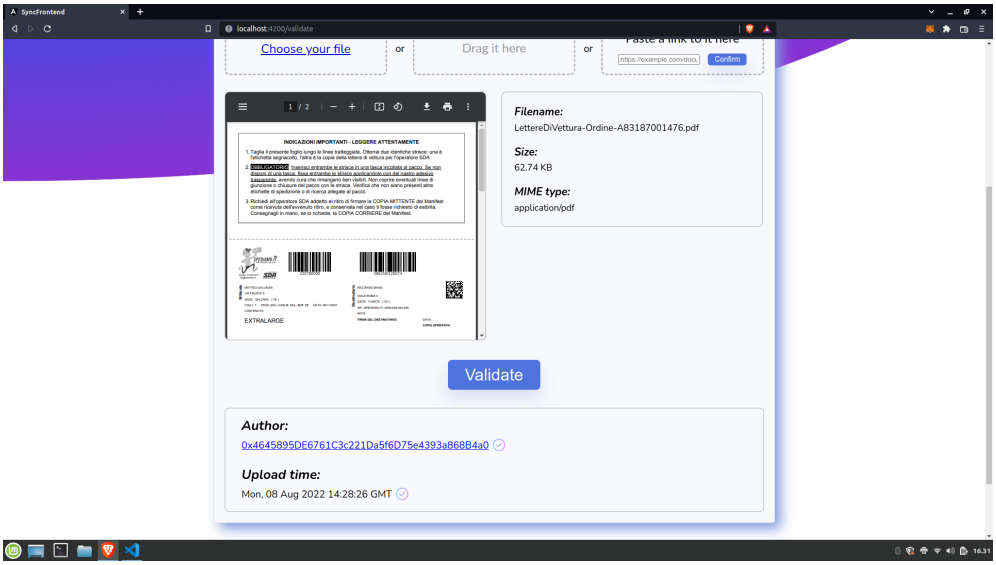


Figura 4.4: Pagina di validazione - documento validato

4.6.4 Versione mobile

Infine, di seguito un'immagine della webapp adattata su dispositivi mobile. La webapp è sviluppata per essere responsive ed adattarsi a tutti gli schermi dei possibili dispositivi.

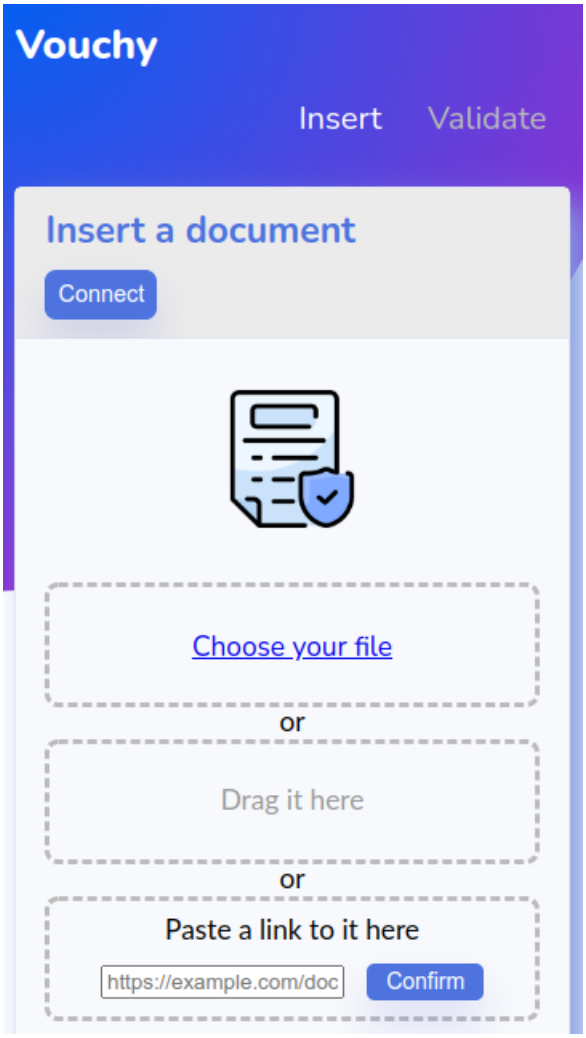


Figura 4.5: Pagina di inserimento - vista da mobile

4.7 Hosting decentralizzato

Tra gli obiettivi non vincolanti ma desiderabili del progetto vi era uno studio di fattibilità con conseguente implementazione circa la possibilità di rendere fruibile la webapp da un servizio di hosting decentralizzato per pagine statiche. Tra le motivazioni per questo obiettivo vi sono il desiderio che la webapp sia sempre online e che non sia facilmente censurabile. Sono state studiate più possibili soluzioni, di seguito descritte:

4.7.1 IPFS

IPFS è un protocollo che consente l'hosting distribuito di files e pagine statiche che si basa sulla frammentazione dei file e la distribuzione di essi tra più nodi. Utilizzato senza alcuna estensione questo protocollo aumenta la ridondanza dei files nei nodi in base alla loro richiesta, dunque ha lo svantaggio che se una pagina ha un basso numero di richieste essa non sarà presente nella cache di molti nodi e di conseguenza gli utenti subiranno gli effetti dell'alta latenza.

Per evitare che i nodi della rete cancellino la dApp dalla loro cache vista la bassissima richiesta è possibile utilizzare un meccanismo proprio di IPFS denominato "pinning", che si basa sul fornire l'identificativo dei file da mantenere al nodo IPFS che non li cancellerà nonostante la loro bassa richiesta; tuttavia questa operazione si può eseguire solo avendo un proprio nodo (soluzione giudicata eccessiva per il progetto) o utilizzando un servizio esterno.
L'operazione di pinning ha comunque il grave problema che, data la poca richiesta, la dApp sarebbe comunque mantenuta su un solo nodo (personale o del servizio esterno che sia), risultando inutile dal punto di vista della decentralizzazione.

Una caratteristica interessante di IPFS è che permette l'accesso ai file memorizzati nella rete anche senza il bisogno di utilizzare un proprio nodo (tuttavia, alcuni browser supportano già l'accesso diretto con protocollo IPFS mediante nodo locale): l'accesso può avvenire infatti tramite "gateway" accessibili come normali siti web che restituiscono i file ricercati; questa non è la soluzione ideale a livello di decentralizzazione ma aiuta sicuramente utenti con browser che non supportano nativamente IPFS (la maggior parte) ad accedere ai file su di esso, e in ogni caso l'ampio numero di gateway pubblici disponibili diminuisce la possibilità di centralizzazione verso uno solo di loro.

4.7.2 Skynet

Skynet è un protocollo basato sulla blockchain "Sia" per molti versi simile ad IPFS, ma presenta un'interessante caratteristica aggiuntiva che ad IPFS manca: con Skynet è possibile instaurare veri e propri contratti (ovviamente digitali) con i nodi della rete, pagando una determinata cifra per garantire l'uptime e la ridondanza dei propri file.

Tuttavia Skynet è un protocollo relativamente giovane e poco usato, dunque sia la documentazione che i servizi che lo utilizzano sono insufficienti a renderlo utilizzabile per il progetto in esame.

4.7.3 IPFS con Filecoin

Sviluppato dagli stessi creatori di IPFS, Filecoin è un protocollo basato sull'omonima blockchain che aggiunge la possibilità di instaurare contratti con i nodi della rete su IPFS per garantire l'uptime e la ridondanza dei file in cambio di un pagamento. Questi contratti garantiscono che i nodi della rete mantengano online i file tramite un meccanismo di penalità economiche in caso di downtime.

Purtroppo l'unica rete di test di Filecoin che rende gratuiti i test su di essa è molto recente e manca di conseguenza della documentazione necessaria ad utilizzarla, e per questo progetto è stata ritenuta sufficiente la ricerca svolta fino a questo punto.

4.7.4 Conclusioni

In conclusione, si è optato per l'utilizzo di un servizio esterno per il pinning dei file su IPFS, ricordando comunque che questo è accettabile in quanto non si tratta di un ambiente di produzione. Per quanto riguarda un possibile deployment in ambiente di produzione sarà bene utilizzare IPFS insieme a Filecoin per garantire la corretta decentralizzazione e dunque resistenza a downtime, ritenendo tale combinazione di tecnologie la migliore.

4.8 ENS

L'ultimo obiettivo facoltativo riguardava l'implementazione dell'accesso alla dApp tramite un nome di dominio basato su Ethereum (ENS). Questo desiderio nasce dalla volontà che la dApp risulti completamente decentralizzata, quindi anche in assenza di server che risolvano un nome di dominio. La webapp sarebbe accessibile anche tramite link diretto con IPFS, ma questo comporta le stesse problematiche di memorizzazione e di riconoscimento che avrebbe navigare il web solamente tramite indirizzi IP.

Di conseguenza, dato che il protocollo ENS (che altro non è che uno smart contract) è utilizzabile anche sulla rete di test Rinkeby, si è proceduto con l'acquisto fittizio del nome di dominio "vouchy.eth", facendolo puntare poi all'indirizzo IPFS della dApp. In questo modo, tutti i browser compatibili con questa tecnologia potranno accedere alla webapp tramite il nome di dominio basato su Ethereum.

Capitolo 5

Verifica e validazione

In questo capitolo vengono illustrate le fasi di verifica e validazione del prodotto.

5.1 Accessibilità

In questa fase è stata verificata la corretta implementazione delle best practices nel campo dell'accessibilità.

5.1.1 Ordine di tabulazione

Utilizzando il browser Firefox si è controllato l'ordine di tabulazione, cioè l'ordine con la quale gli utenti che navigano la pagina con la tastiera visualizzerebbero gli elementi cliccabili.

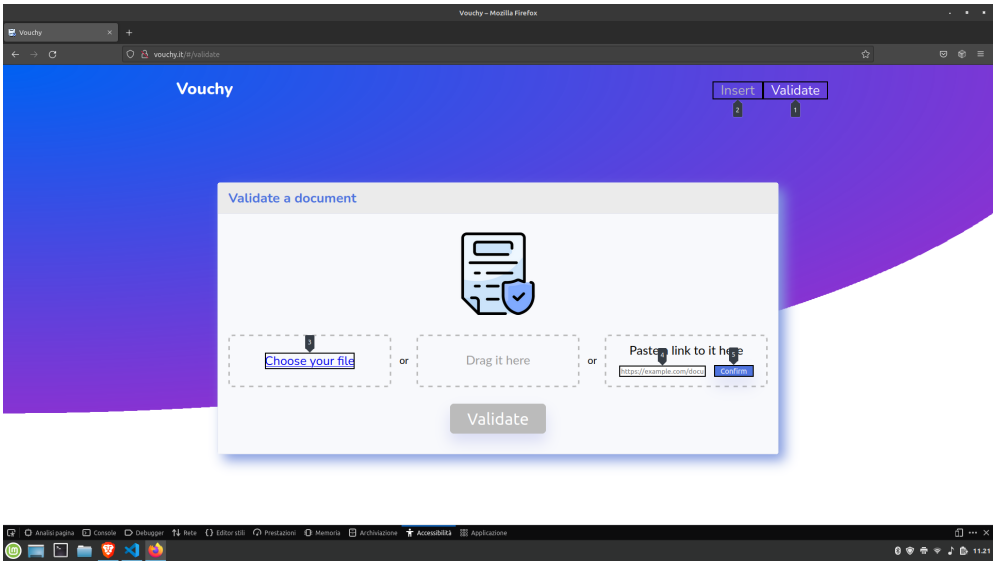


Figura 5.1: Ordine di tabulazione

5.1.2 Colori

Successivamente, utilizzando lo strumento di verifica accessibilità fornito da Firefox, si è verificato che nessuna informazione importante fosse derivata dal colore al fine di consentire una buona esperienza di navigazione anche ad utenti affetti da disturbi alla vista. Di seguito le immagini di due dei casi più significativi, tritanopia (cecità del blu) e perdita di contrasto.

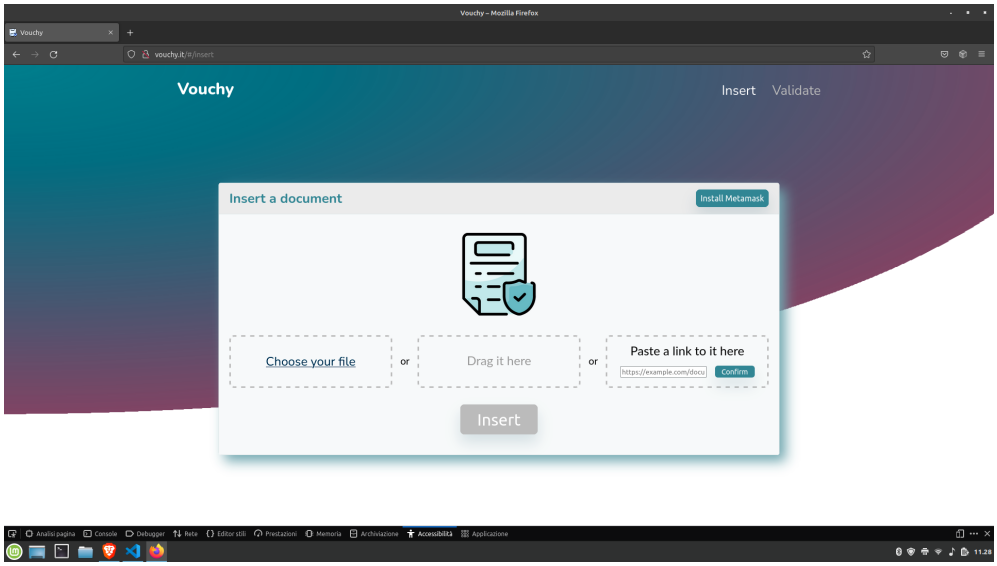


Figura 5.2: Tritanopia

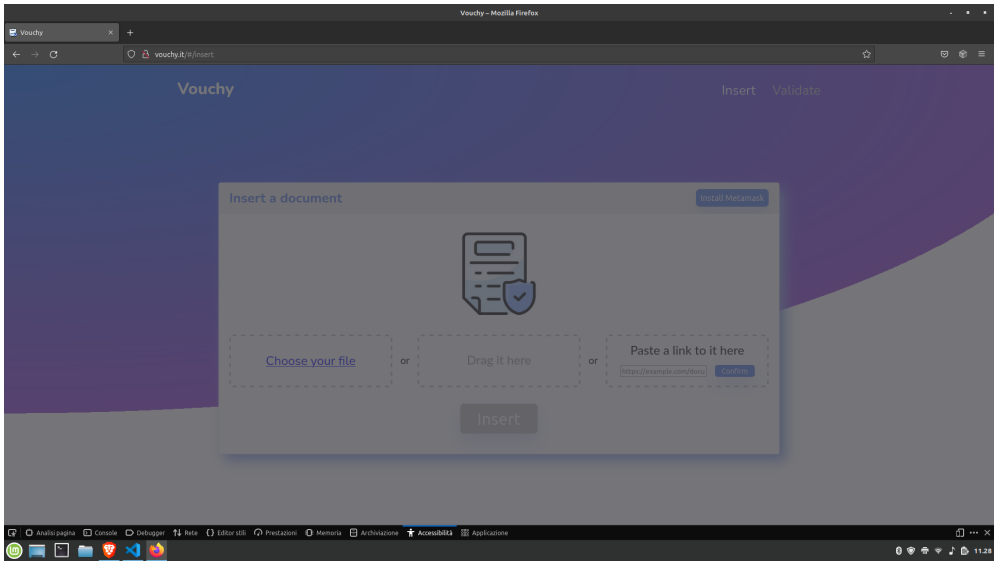


Figura 5.3: Perdita di contrasto

5.1.3 Lighthouse

È stato utilizzato Lighthouse (strumento di Google Chrome) per effettuare dei test automatici circa performance, accessibilità e indicizzazione dai motori di ricerca. I report generati da Lighthouse evidenziano delle performance scarse, nonostante tutti gli altri argomenti di test siano molto buoni.

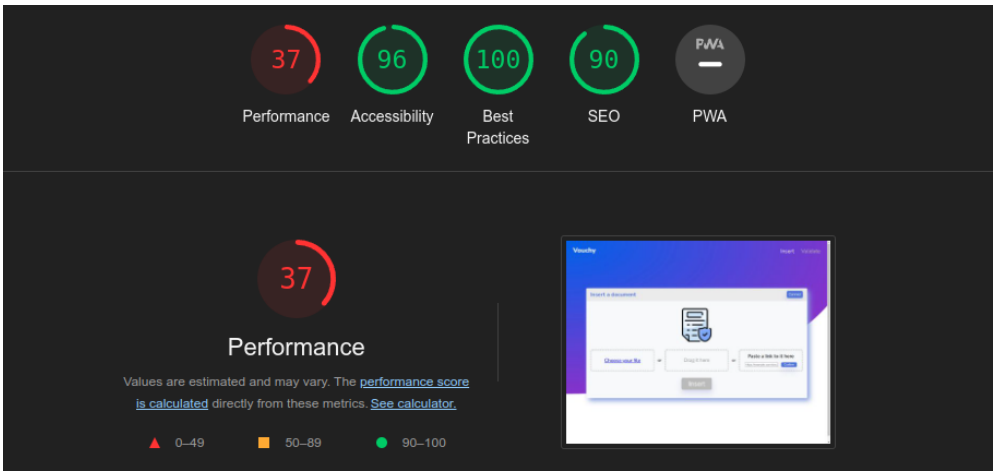


Figura 5.4: Report di Lighthouse

Questo è da imputare principalmente all'utilizzo estensivo di Javascript dovuto allo sviluppo tramite Angular. I tempi di caricamento sono infatti molto rallentati per colpa del codice Javascript eseguito per generare gli elementi della pagina.

Nonostante esistano dei metodi per mitigare il problema (per esempio il rendering lato server), non sono state implementate in questo progetto in quanto ritenute non immediatamente necessarie.

5.1.4 Problemi

Il principale problema di accessibilità riguarda Metamask, poichè alla sua apertura appare come pop-up e può disorientare alcuni screen reader. Purtroppo l'utilizzo di questo software è stato obbligatorio, e le sue caratteristiche di accessibilità sono fuori dal controllo degli sviluppatori web.

5.2 Validazione

Terminata l'esperienza di stage, si è verificata la copertura dei requisiti individuati nel capitolo 3. Tale copertura risulta totale e il prodotto è stato giudicato più che soddisfacente da parte del committente.

Tabella 5.1: Validazione dei requisiti

Tipo	**Coperti**	**Totali**	**Copertura percentuale**
Funzionali	16	16	100%
Qualitativi	5	5	100%
Di vincolo	4	4	100%

Capitolo 6

Conclusioni

Questo capitolo contiene le conclusioni sull'esperienza svolta.

6.1 Raggiungimento degli obiettivi

Tutti gli obiettivi sono stati raggiunti nelle tempistiche previste. Sia l'analisi dei requisiti che la soluzione individuata sono stati ritenuti più che soddisfacenti dal committente. Anche i test e la documentazione prodotti sono stati giudicati come apprezzabili.

Obiettivi obbligatori

* *O-01*: Soddisfatto;
* *O-02*: Soddisfatto;
* *O-03*: Soddisfatto;
* *O-04*: Soddisfatto;

Obiettivi desiderabili

* *D-01*: Soddisfatto;

Obiettivi facoltativi

* *F-01*: Soddisfatto;
* *F-02*: Soddisfatto;

6.2 Conoscenze acquisite

Nel corso di questo stage ho avuto modo di sviluppare diverse conoscenze circa le tecnologie utilizzate per il progetto. In particolar modo ho imparato ad utilizzare Angular e ho studiato i suoi concetti base, che penso saranno molto utili nel mio futuro in quanto Angular è un framework estremamente utilizzato nel mondo del lavoro.

Ho inoltre maturato una discreta esperienza con Java Spring che ritengo importante perchè anche quest'ultimo è molto utilizzato nello sviluppo di backend.

6.3 Valutazione personale

Ho apprezzato molto l'esperienza di stage poichè mi ha dato modo di applicare nella pratica buona parte delle conoscenze teoriche maturate durante i miei anni di studio. L'ambiente di lavoro si è rivelato eccellente, con un tutor aziendale cordiale e disponibile.

Nel complesso sono più che soddisfatto dell'esperienza svolta, sia come introduzione al mondo del lavoro sia per il risultato finale prodotto.

Acronimi e abbreviazioni

API Application Program Interface. 37, 38

CORS Cross-Origin Resource Sharing. 37

dApp Decentralized Application. 37

ENS Ethereum Name System. 37

HTTP Hypertext Transfer Protocol. 38

IPFS InterPlanetary File System. 38

IT Information Technology. 38

REST Representational State Transfer. 38

RPC Remote procedure call. 38

Glossario

API in informatica con il termine *API, Application Programming Interface* (ing. interfaccia di programmazione di un'applicazione) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. 3, 20, 25, 35

backend in informatica con il termine *Backend* si indica la parte di codice che esegue tipicamente su un server, nascosta dalla vista dell'utente, e che deve comunicare con la parte di frontend restituendo i risultati delle sue elaborazioni. 2, 15, 20, 34, 37, 38

blockchain in informatica con *Blockchain* si intende una struttura dati condivisa e "immutabile" organizzata per blocchi di operazioni, dove ogni blocco è approvato o scartato in base ad un definito meccanismo di consenso; sebbene sia possibile aggiungere nuovi blocchi a questa struttura dati, non è possibile rimuoverne senza invalidare l'intera catena da quel blocco in poi. 1–4, 6, 10–12, 14, 15, 18–21, 27, 28, 37, 38

CORS con *CORS, Cross-Origin Resource Sharing* si intende un meccanismo per richieste HTTP che di default blocca l'accesso da browser alle proprie risorse da parte di domini diversi dal proprio. 23, 35

dApp nel contesto della tecnologia blockchain con *dApp, Decentralized Application* (ing. applicazione decentralizzata) si intende un software composto da un frontend e da uno smart contract che assume la funzione di backend delle tradizionali architetture software; tale frontend si occuperà di stabilire un dialogo diretto con lo smart contract fornendone un'interfaccia grafica per gli utenti. 2, 3, 27, 28, 35

ENS è un protocollo per la risoluzione dei nomi di dominio basata su Ethereum. Tali nomi di dominio sono rappresentati come token in blockchain, sono liberamente scambiabili e possono puntare a wallet/indirizzi web ed essere risolti dai browser compatibili. 28, 35

Ethereum-compatibile nel contesto della blockchain con *Ethereum-compatibile* si intende una qualsiasi blockchain che utilizza per l'esecuzione dei suoi smart contract una macchina virtuale compatibile al 100% con quella utilizzata dalla blockchain di Ethereum; questa compatibilità garantisce che gli smart contract

scritti per Ethereum funzionino senza alcuna modifica anche su un'altra blockchain compatibile. 2, 3, 15

framework un *framework* in informatica è un'architettura logica di supporto sulla quale un software può essere progettato e realizzato. 3–7, 17, 18, 21, 34

frontend in informatica con il termine *Frontend* si indica la parte di codice che esegue sull'elaboratore dell'utente finale, visibile e spesso con elementi grafici, che si occupa generalmente di presentare i risultati delle elaborazioni svolte dal backend all'utente. 3, 15, 20, 23, 37

GET il metodo *GET* è un metodo per la trasmissione in chiaro di dati tramite passaggio nell'URL. 14, 20

HTTP *HTTP, Hypertext Transfer Protocol* è un protocollo di comunicazione sul web estremamente popolare. 22, 35, 37, 38

iframe l'*iframe* è un elemento HTML che consente l'inclusione di una pagina web all'interno di un'altra pagina web tramite URL. 3

IPFS è un protocollo per la condivisione di file in maniera distribuita. 27, 28, 35

IT in informatica con *IT, Information Technology* (ing. tecnologie dell'informazione) si intende l'utilizzo di qualsiasi elaboratore, dispositivo di memoria, rete, infrastruttura o processo per creare, elaborare o memorizzare qualsiasi forma di dati elettronici. 1, 35

Metamask *Metamask* è un software (estensione per browser ed applicazione nativa) che consente agli utenti di firmare transazioni ed emetterle in blockchain. 7, 9–11, 13–15, 18, 20, 21, 23–25, 31

REST in informatica con *REST, Representational State Transfer* si intende un particolare tipo di Application Program Interface (API) che rispetta i seguenti vincoli architetturali: architettura client-server, richieste tramite HTTP, assenza di stato nelle richieste, informazioni restituite in formato standard. 3, 35

RPC in informatica con *RPC, Remote procedure call* (ing. chiamata di procedura remota) si intende l'avvio di una procedura su un computer diverso da quello sul quale il programma che la utilizza viene eseguito. 3, 14, 25, 35

smart contract nel contesto della blockchain con *Smart contract* (ing. contratto intelligente/digitale) si intende un protocollo, pubblicato su una blockchain, che verifica e fa rispettare delle clausole scritte sottoforma di codice. In una definizione più ampia, un contratto digitale non è altro che un software che sfrutta l'immutabilità della blockchain sulla quale è pubblicato per garantire che il suo codice venga eseguito sempre allo stesso modo, sia consultabile pubblicamente e non sia facilmente censurabile. 1–4, 12, 14, 15, 17, 19–21, 28, 37

trustless in informatica con *Trustless* si intende un sistema all'interno della quale i partecipanti non hanno necessità di fidarsi l'uno dell'altro perchè le operazioni possibili al suo interno sono definite dal sistema stesso, e tale sistema protegge automaticamente i partecipanti da qualsiasi azione non sia definita come valida. 2, 3

Bibliografia

Siti web consultati

Documentazione Angular. URL: https://angular.io/docs.

Documentazione Metamask. URL: https://docs.metamask.io/guide/.

Documentazione Typescript. URL: https://www.typescriptlang.org/docs.

Documentazione Web3j. URL: https://docs.web3j.io/4.8.7/.

Documentazione Web3js. URL: https://web3js.readthedocs.io/en/v1.7.5/.

Sito web di Sync Lab. URL: https://www.synclab.it/about.php.