

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

CORSO DI LAUREA IN INFORMATICA



**Implementazione di funzionalità di front-end
per il progetto ShopChain, tramite il
framework Angular**

Tesi di laurea

Relatore

Prof. Massimiliano de Leoni

Laureando

Domenico Casazza

ANNO ACCADEMICO 2021-2022

Vivere è la cosa più rara al mondo. La maggior parte della gente esiste e nulla più.

— Oscar Wilde

Dedicato ai miei genitori

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecento ore, dal laureando Domenico Casazza presso l'azienda Sync Lab S.r.l. Il percorso di stage si inserisce nel progetto ShopChain, ovvero nello sviluppo di un'applicazione web prototipale per la gestione di pagamenti online tramite criptovalute. Gli obbiettivi da raggiungere erano molteplici.

In primo luogo era richiesto lo studio delle tecnologie necessarie per l'implementazione del front end del progetto ShopChain, in particolare il linguaggio TypeScript, il framework [Angular](#)^[g] e la libreria grafica Angular Material.

In secondo luogo era richiesta la progettazione delle maschere di front end per l'applicazione web, più precisamente la maschera per il pagamento in criptovalute e la maschera per la gestione degli ordini da parte di un utente compratore. Infine era richiesta l'implementazione di tali maschere.

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

— Confucius

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Massimiliano de Leoni, relatore della mia tesi, per l'aiuto fornitomi durante la stesura del lavoro.

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

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

Padova, Luglio 2022

Domenico Casazza

Indice

1	Introduzione	1
1.1	L'azienda	1
1.2	L'idea	1
1.3	Architettura del progetto	3
1.3.1	Architettura nel dettaglio	3
1.4	Strumenti utilizzati	4
1.5	Organizzazione del testo	5
1.5.1	Struttura del documento	5
1.5.2	Convenzioni tipografiche	5
2	Descrizione dello stage	7
2.1	Analisi preventiva dei rischi	7
2.2	Requisiti e obiettivi	7
2.3	Pianificazione del lavoro	8
3	Analisi dei requisiti	11
3.1	Casi d'uso	11
3.1.1	Attori principali	11
3.1.2	Attori secondari	11
3.1.3	Ordini	11
3.1.4	Elenco casi d'uso	12
3.1.5	Landing Page	12
3.1.6	Applicazione Web	14
3.2	Tracciamento dei requisiti	20
4	Progettazione e codifica	23
4.1	Tecnologie	23
4.2	Progettazione	24
4.2.1	Architettura di Angular	24
4.2.2	Architettura dell'applicazione sviluppata	25
4.2.3	Progettazione delle maschere	25
4.2.4	Progettazione dello smart contract	26
4.3	Organizzazione dei componenti	26
4.4	Design Pattern implementati	28
4.5	Codifica	29
4.5.1	Maschere	29
4.5.2	Componenti	34
4.5.3	Servizi	39

4.6	Problematiche riscontrate	41
5	Verifica e validazione	43
5.1	Accessibilità	43
5.1.1	Attributi HTML	43
5.1.2	Colori	43
5.2	Validazione e collaudo	44
6	Conclusioni	45
6.1	Raggiungimento degli obiettivi	45
6.2	Conoscenze acquisite	45
6.3	Valutazione personale	46
	Glossary	49
	Acronyms	51
	Bibliografia	53

Elenco delle figure

1.1	Interazione tra Wallet - Frontend - IPFS - Smart Contract [4]	3
1.2	DApp Pattern B - Self-Confirmed Transactions	4
1.3	Architettura progetto ShopChain	4
3.1	Use Cases - Landing Page	12
3.2	Use Cases - Applicazione Web	14
3.3	Use Case 4 - Visualizzazione Lista Ordini	14
3.4	Use Case 4.1 - Visualizzazione Singolo Ordine nella Lista	15
3.5	Use Case 5 -Filtra Ordini	17
3.6	Use Case 7 - Visualizzazione Dettagli Ordine	19
4.1	Schema logico dell'architettura di un applicazione web basata su Angular	24
4.2	Mock-up della pagina principale dell'applicazione web	25
4.3	Diagramma dei componenti della landing page	26
4.4	Diagramma dei componenti della pagina principale	27
4.5	Pagina di autenticazione	29
4.6	Pagina di pagamento e creazione ordine	30
4.7	Home page di ShopChain	31
4.8	Pagina dei dettagli di un ordine	32
4.9	Pagina di errore: connessione a blockchain sbagliata	33
4.10	Componente AccessComponent	34
4.11	Componente BuyerComponent	35
4.12	Componente HeaderComponent	35
4.13	Componente LandingPageComponent	36
4.14	Componente LogComponent	37
4.15	Componente OrderInfoComponent	37
4.16	Componente OrdersComponent	38
4.17	Componente SwitchNetworkComponent	39
4.18	Esempio soluzione 1	41
4.19	Esempio soluzione 2	41

Elenco delle tabelle

3.2	Tabella del tracciamento dei requisiti funzionali	20
3.2	Tabella del tracciamento dei requisiti funzionali	21
3.3	Tabella del tracciamento dei requisiti qualitativi	21
5.1	Rapporto contrasto testo-sfondo	43
5.2	Test di accettazione	44
6.1	Riepilogo obiettivi tirocinio	45

Capitolo 1

Introduzione

1.1 L'azienda

Sync Lab nasce a Napoli nel 2002 come software house ed è rapidamente cresciuta nel mercato dell'Information and Communications Technology (ICT). A seguito di una maturazione delle competenze tecnologiche, metodologiche ed applicative nel dominio del software, l'azienda è riuscita rapidamente a trasformarsi in System Integrator conquistando significative fette di mercato nei settori mobile, videosorveglianza e sicurezza delle infrastrutture informatiche aziendali. Attualmente, Sync Lab ha più di 150 clienti diretti e finali, con un organico aziendale di 300 dipendenti distribuiti tra le 6 sedi dislocate in tutta Italia a Napoli, Roma, Milano, Padova, Verona e Como.

Sync Lab si pone come obiettivo principale quello di supportare il cliente nella realizzazione, messa in opera e governance di soluzione IT, sia dal punto di vista tecnologico, sia nel governo del cambiamento organizzativo.

1.2 L'idea

L'avvento delle tecnologie su Blockchain ha portato in questi ultimi anni e porterà in futuro moltissimi cambiamenti sulla società. Indubbiamente il settore dove questa tecnologia trova il campo più fertile è quello economico/finanziario. L'avvento della criptomoneta Bitcoin, nel 'lontano' 2009, ha contribuito a porre le basi di una nuova 'era finanziaria' in cui non esiste più un regolatore, un ente che regola e governa le politiche economico-finanziarie dei beni di scambio. Per questo si sono sviluppate presto molte altre cripto-monete diventate più o meno famose (Ethereum, Ripple, Litecoin ecc.) accessibili a tutti senza bisogno di aprire un conto in banca.

Successivamente sono nati gli smart contract, contratti (algoritmi) che vengono 'minati' nella catena ed eseguiti in autonomia (senza nessuna possibilità di modifica del codice o di controllo dall'esterno) al verificarsi di eventi interni alla piattaforma. Questo ha permesso l'implementazione di meccanismi di 'tokenizzazione' degli asset digitali (e non), e la definizione dello standard [ERC20](#)^[8] dei token su catena Ethereum (e di altri standard su altre catene). Da qui poi è nata quella che viene spesso definita la 'DeFi' (ossia Finanza Decentralizzata), una finanza fatta di asset digitali (token) che rimangono solo nei ledger delle blockchain.

In questi ultimi mesi quindi per andare incontro a questa esigenza stanno nascendo delle piattaforme di e-commerce attraverso le quali è possibile acquistare beni/servizi usando

direttamente criptomonete (per esempio <https://cryptoemporium.eu>). Grandi aziende come Tesla hanno annunciato che accetteranno il Bitcoin come forma di pagamento per gli acquisti dei loro prodotti.

L'offerta di venditori che accettano pagamenti in criptomonete si sta sviluppando, dove manca però ancora una tutela dell'acquirente. Infatti, pagando con questo sistema il cripto denaro viene versato nel wallet un destinatario senza garanzia di affidabilità nel fornire i beni o i servizi offerti.

Non esiste ancora un sistema simile a PayPal che faccia da garante tra l'acquirente ed il venditore affinché vengano portate a termine in modo corretto da ambo le parti le operazioni di acquisto/ consegna/ ricezione del bene o servizio.

La soluzione prospettata in questo progetto consiste nel realizzare un prototipo di una piattaforma in grado di 'affiancare' un cripto-e-commerce dalle fasi di pagamento fino alla consegna.

Si propone la realizzazione di una piattaforma su blockchain che si incarichi di ricevere l'ammontare in criptovaluta e che lo trattienga e lo consegni al venditore solo quando il pacco viene recapitato all'acquirente. L'evento che scatena l'avvio della procedura sarà il trasferimento dei dati dell'ordine alla blockchain da parte dell'acquirente. La piattaforma a questo punto starà quindi in attesa di ricevere i soldi (ovviamente criptomonete da versare nel proprio wallet) dall'acquirente, quando li riceverà notificherà l'evento al venditore (che modificherà lo stato dell'ordine e spedirà l'articolo). Nel momento della consegna del pacco l'acquirente dovrà necessariamente inquadrare il QR code applicato sul collo che certifica l'avvenuta consegna. A questo punto quindi verrà effettuato il passaggio della criptovaluta dal wallet della piattaforma al wallet del venditore.

1.3 Architettura del progetto

L'architettura di ShopChain, rappresentata in Figura 1.1, è di tipo Fully Decentralized (Pure DApp).

In una Pure DApp l'utente, dopo essersi connesso al proprio wallet, dal front-end può chiamare, tramite un'interfaccia, i metodi dello Smart Contract senza dover passare per un intermediario. Questo è possibile se il frontend viene ospitato su servizio distribuito come ad esempio IPFS (InterPlanetary File System).[7]

Vantaggi principali offerti dal pattern architetturale:

- Maggiore decentralizzazione (assenza di un server centralizzato);
- Maggiore sicurezza per l'utente (non ci sono intermediari tra di esso e la blockchain).

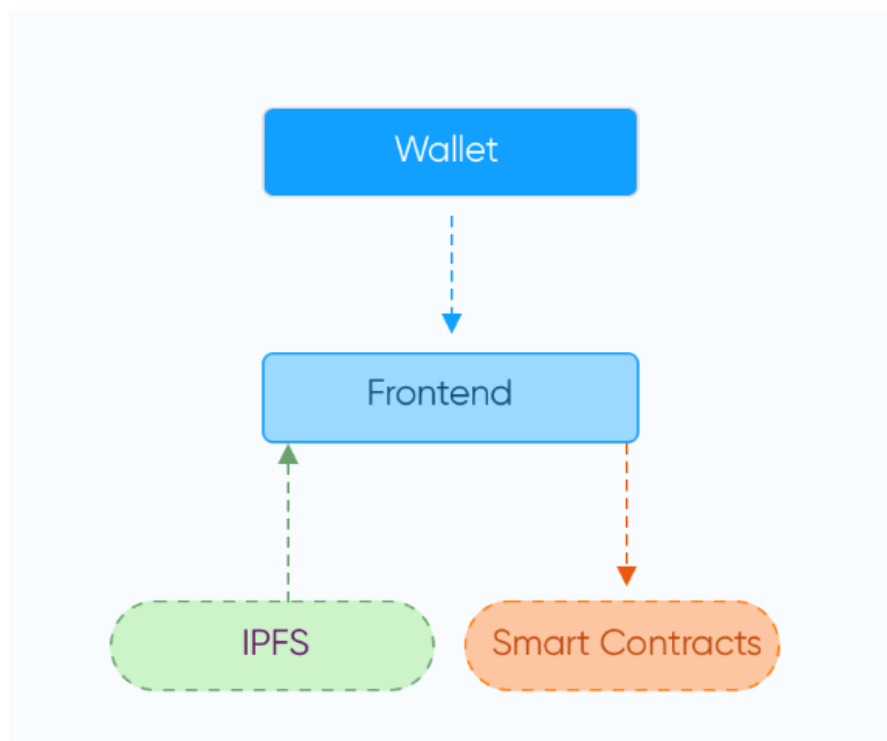


Figura 1.1: Interazione tra Wallet - Frontend - IPFS - Smart Contract [4]

1.3.1 Architettura nel dettaglio

L'architettura di ShopChain fa riferimento al *"Pattern B – Self-Confirmed Transactions"* descritto nel paper *"Engineering Software Architectures of Blockchain-Oriented Applications"* di F. Wessling e V. Gruhn dell'Università di Duisburg-Essen, raffigurata in Figura 1.2.[5]

In questo paper il Pattern B consiste nell'interazione dell'utente solo con un'applicazione web e/o un gestore di wallet (in questo caso [MetaMask](#)^[6]) per creare transazioni su

una blockchain: le transazioni non vengono create direttamente dall'utente ma vengono generate dall'applicazione web e poi mandate manualmente al nodo della blockchain a cui l'utente è collegato.

Questo pattern bilancia sicurezza e facilità nell'interazione con la webApp perché creare transazioni manualmente è un'operazione difficile e realizzabile solo da utenti esperti, ma questo implica che chi interagisce con l'applicativo si fidi degli sviluppatori dato che la generazione di una transazione non è completamente trasparente.

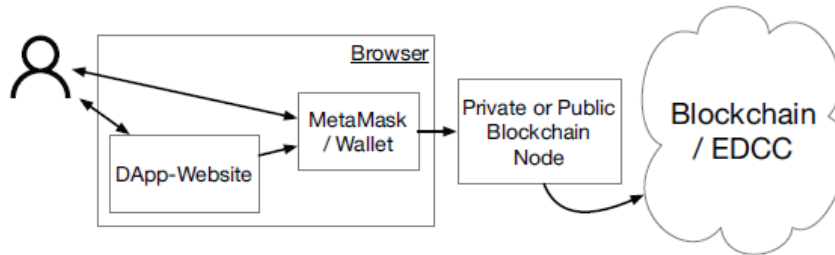


Figura 1.2: DApp Pattern B - Self-Confirmed Transactions

Applicando l'architettura appena descritta al progetto ShopChain si ottiene quindi:

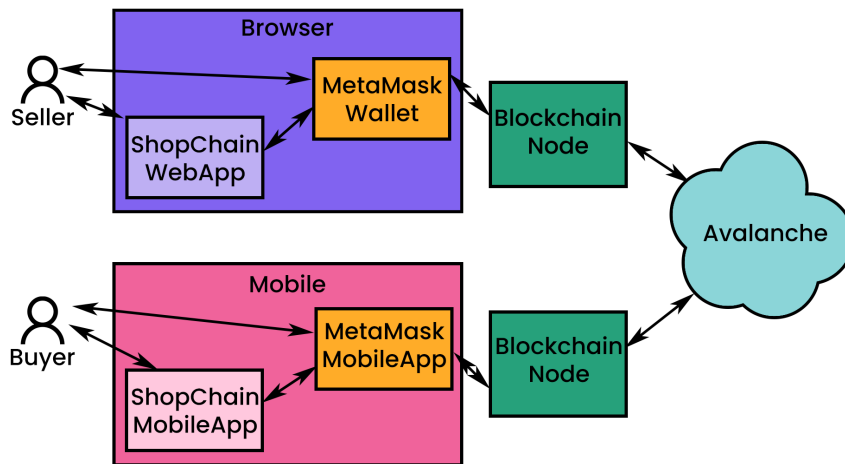


Figura 1.3: Architettura progetto ShopChain

1.4 Strumenti utilizzati

Visual Studio Code

Visual studio code è un editor di codice sorgente. Grazie alle numerose estensioni, che è possibile installare, si possono usare una vasta gamma di linguaggi e funzionalità di supporto alla scrittura del codice. Tra le estensioni utilizzate vi sono una per integrare Git, una per migliorare la leggibilità del codice e altre per velocizzare la scrittura di codice sorgente mentre si sviluppa in TypeScript e in particolare in Angular.

Git

Git è uno strumento per il controllo di versione. Utilizzato per collaborare con gli altri membri del gruppo e per controllare la versione del codice prodotto così da poter ritornare ad una versione stabile in caso di problemi.

Fuji Testnet

Fuji è la testnet della blockchain Avalanche in cui è possibile eseguire il deploy di smart contract per testarli e creare transazioni senza dover convertire soldi in criptovalute per eseguire test.

Draw.io

Software collaborativo, integrato con Google Drive, per la creazione dei mock-up delle maschere. Utilizzato in fase di progettazione per mostrare al committente come è stata ideata la grafica dell'applicazione e per ricevere dei feedback su eventuali modifiche prima di sviluppare l'applicazione vera e propria.

1.5 Organizzazione del testo

1.5.1 Struttura del documento

Il secondo capitolo descrive l'analisi preventiva dei rischi, gli obiettivi dello stage e la pianificazione delle ore di lavoro.

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

Il quarto capitolo approfondisce la progettazione e la codifica delle maschere e dei servizi.

Il quinto capitolo approfondisce la verifica e la validazione dell'applicativo.

Nel sesto capitolo si descrivono le conclusioni e le opinioni personali.

1.5.2 Convenzioni tipografiche

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

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

Capitolo 2

Descrizione dello stage

In questo capitolo è presente un'analisi preventiva dei rischi che potevano venire riscontrati durante lo svolgimento dello stage, la lista degli obiettivi da raggiungere e la pianificazione delle ore di lavoro.

2.1 Analisi preventiva dei rischi

Durante la fase di analisi iniziale sono stati individuati dei possibili rischi che si potranno incontrare durante il percorso di stage. Si è quindi proceduto a elaborare delle possibili soluzioni per far fronte a tali rischi.

1. **Inesperienza tecnologica:** alcune tecnologie incluse nel progetto sono parzialmente o completamente sconosciute.
Soluzione: studio delle tecnologie nelle prime settimane di stage tramite documentazione ufficiale e corsi forniti dall'azienda.
2. **Problematiche hardware:** le macchine utilizzate durante l'implementazione possono essere soggette a malfunzionamenti o guasti, con conseguenti perdite di dati.
Soluzione: condivisione giornaliera su una repository del lavoro svolto.
3. **Monitoraggio scadenze:** a causa di altri impegni lavorativi, non sarà sempre possibile confrontarsi con il tutor aziendale per aggiornarlo sullo svolgimento del lavoro.
Soluzione: creazione di un foglio Excel condiviso contenente un report giornaliero delle attività svolte.

2.2 Requisiti e obiettivi

Notazione

Si farà riferimento ai requisiti secondo le seguenti notazioni:

- *O* per i requisiti obbligatori, vincolanti in quanto obiettivo primario richiesto dal committente;

- *D* per i requisiti desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
- *F* per i requisiti facoltativi, rappresentanti valore aggiunto non strettamente competitivo.

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

Obiettivi fissati

Si prevede lo svolgimento dei seguenti obiettivi:

- Obbligatori
 - *O01*: Acquisizione competenze sulle tematiche sopra descritte;
 - *O02*: Capacità di raggiungere gli obiettivi richiesti in autonomia seguendo il cronoprogramma;
 - *O03*: Portare a termine le implementazioni previste con una percentuale di superamento pari al 80%.
- Desiderabili
 - *D01*: Portare a termine le implementazioni previste con una percentuale di superamento pari al 100%.
- Facoltativi
 - *F01*: Apportare un valore aggiunto al gruppo di lavoro durante le fasi di progettazione delle interfacce.

2.3 Pianificazione del lavoro

Lo stage ha una durata di 8 settimane e prevede lo svolgimento di 320 ore effettive di lavoro. È diviso in due parti, il primo mese di studio delle tecnologie e il secondo di progettazione e implementazione delle maschere.

La ripartizione settimanale delle attività è la seguente:

- **Prima Settimana (40 ore)**
 - Incontro con persone coinvolte nel progetto per discutere i requisiti e le richieste relativamente al sistema da sviluppare;
 - Presentazione strumenti di lavoro per la condivisione del materiale di studio e per la gestione dell'avanzamento;
 - Condivisione scaletta di argomenti;
 - Ripasso del linguaggio Java SE;
 - Ripasso concetti Web (Servlet, servizi Rest, Json ecc.).
- **Seconda Settimana (40 ore)**
 - Studio principi generali di Spring Core (IOC, Dependency Injection);
 - Studio SpringBoot;

- Studio Spring Data/DataRest.
- **Terza Settimana (40 ore)**
 - Ripasso linguaggio Javascript;
 - Studio del linguaggio TypeScript.
- **Quarta Settimana (40 ore)**
 - Studio piattaforma NodeJS e AngularCLI;
 - Studio framework Angular.
- **Quinta Settimana (40 ore)**
 - Analisi e studio del progetto ShopChain;
 - Progettazione ed implementazione della nuova maschera di accesso.
- **Sesta Settimana (40 ore)**
 - Progettazione ed implementazione nuova maschera "Gestione Profilo Venditore";
 - Scrittura dei service (su front-end) di chiamata al back-end.
- **Settima Settimana (40 ore)**
 - Progettazione ed implementazione nuova maschera "Gestione Profilo Acquirente";
 - Scrittura dei service (su front-end) di chiamata al back-end.
- **Ottava Settimana - Conclusione (40 ore)**
 - Termine integrazioni e collaudo finale.

Capitolo 3

Analisi dei requisiti

In questo capitolo vengono descritte le funzionalità che il prodotto deve offrire elencando i casi d'uso, gli attori del sistema e i requisiti individuati.

3.1 Casi d'uso

3.1.1 Attori principali

Durante la fase di analisi sono stati individuati i seguenti attori principali:

- Utente non riconosciuto: attore che indica un utente che non ha ancora eseguito l'autenticazione tramite [MetaMask](#);
- Utente riconosciuto: attore che indica un utente che ha eseguito l'autenticazione tramite [MetaMask](#) e può accedere alla creazione di un ordine o vedere le informazioni relative agli ordini.

3.1.2 Attori secondari

Durante la fase di analisi sono stati individuati i seguenti attori secondari:

- [MetaMask](#): plugin esterno al sistema che gestisce i wallet degli utenti e permette loro di essere riconosciuti e accedere all'applicazione web.

3.1.3 Ordini

Ogni ordine è caratterizzato da:

- un numero identificativo;
- l'indirizzo dell'acquirente;
- l'indirizzo del venditore;
- il prezzo;
- lo stato dell'ordine.

Di seguito è riportata la tabella che descrive i diversi tipi di stato che un ordine può assumere durante il suo ciclo di vita.

Stato	Descrizione	Precondizione
Created	L'acquirente ha versato la quantità di token richiesti nello smart contract e l'ordine è stato creato	Nessuna
Shipped	Il venditore ha inviato il pacco all'acquirente	Created
Confirmed	L'acquirente ha ricevuto il pacco e ha scansionato il QR code	Shipped
Deleted	Il venditore ha annullato l'ordine	Created/Shipped
Refund Asked	L'acquirente ha chiesto il rimborso dell'ordine	Created/Shipped/Confirmed
Refunded	Il venditore ha confermato la richiesta di rimborso	Refund Asked

3.1.4 Elenco casi d'uso

Per lo studio dei casi di utilizzo del prodotto sono stati creati dei diagrammi. I diagrammi dei casi d'uso (in inglese *Use Case Diagram*) sono diagrammi di tipo [Unified Modeling Language \(UML\)](#)^[6] dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso.

3.1.5 Landing Page

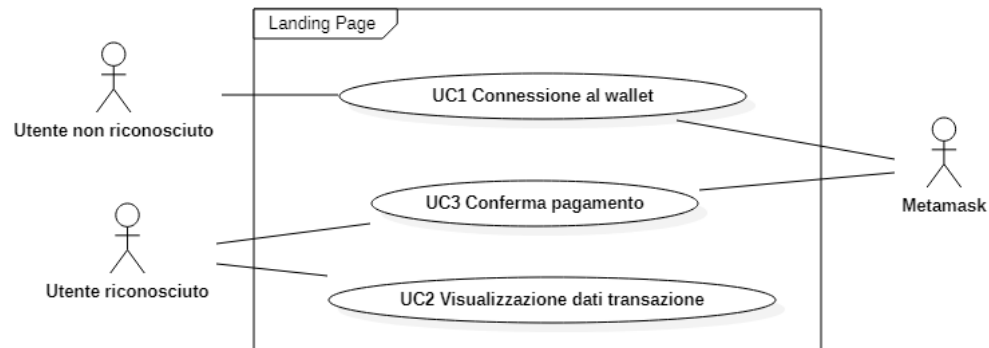


Figura 3.1: Use Cases - Landing Page

UC1: Connessione al wallet

Attori Principali: Utente non riconosciuto.

Attori Secondari: MetaMask.

Precondizioni: L'Utente non ha effettuato il collegamento al proprio wallet.

Postcondizioni: L'Utente si è identificato e può proseguire con il pagamento.

Descrizione: L'Utente vuole interagire con lo smart contract di ShopChain, quindi deve collegare il wallet ad esso per poter autorizzare successivamente le transazioni.

1. L'Utente clicca su "Connetti wallet";
2. Si apre il pop-up di MetaMask dove egli inserisce i dati;
3. L'utente dà il permesso allo smart contract di interagire con il proprio wallet.

UC2: Visualizzazione Dati Transazione

Attori Principali: Utente riconosciuto.

Precondizioni: L'Utente ha iniziato la fase di pagamento nella piattaforma e-commerce e ha eseguito l'accesso al wallet.

Postcondizioni: L'Utente ha visualizzato i dati della transazione.

Descrizione: L'Utente visualizza nella Landing Page:

1. l'importo corrispondente al/i prodotto/i selezionato/i;
2. eventuali fee stimate (medie);
3. l'indirizzo del venditore (e-commerce).

UC3: Conferma Pagamento

Attori Principali: Utente riconosciuto.

Attori Secondari: MetaMask.

Precondizioni: L'Utente è stato riconosciuto e ha visualizzato i dati della transazione.

Postcondizioni: La transazione è stata avviata e le criptovalute si trovano nello smart contract.

Descrizione:

1. L'Utente autorizza la transazione iniziata;
2. l'interfaccia di MetaMask permette la conferma;
3. l'importo viene trasferito dal suo wallet allo smart contract;
4. viene creato l'ordine nello smart contract in stato "Created".

3.1.6 Applicazione Web

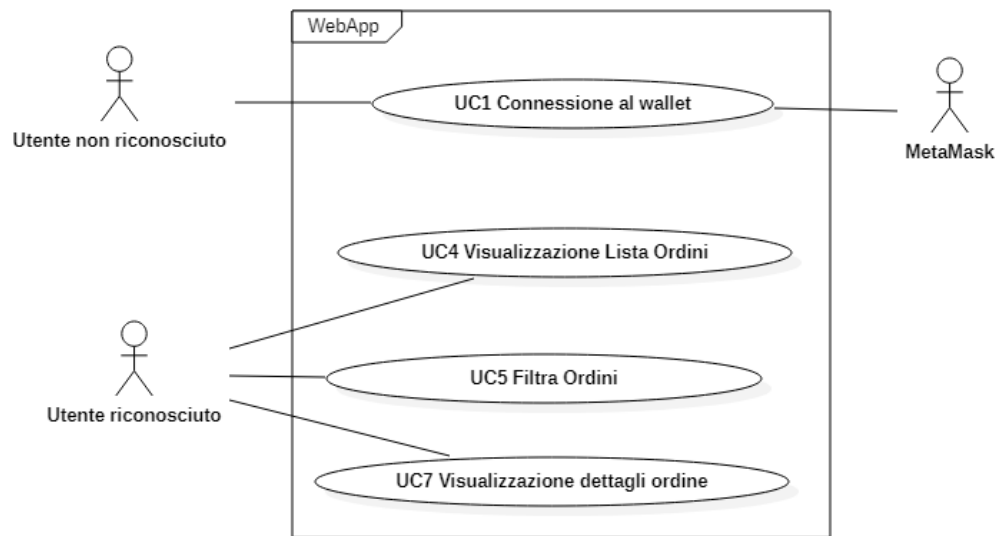


Figura 3.2: Use Cases - Applicazione Web

UC4: Visualizzazione Ordini

Attori Principali: Utente riconosciuto.

Precondizioni: L'Utente è stato riconosciuto tramite la connessione al suo wallet.

Postcondizioni: L'Utente ha visualizzato gli ordini relativi al wallet collegato.

Descrizione: L'Utente può visualizzare una tabella contenente gli ordini (in cui egli è uno degli attori) con identificativo, indirizzo dell'altro attore, costo, stato.

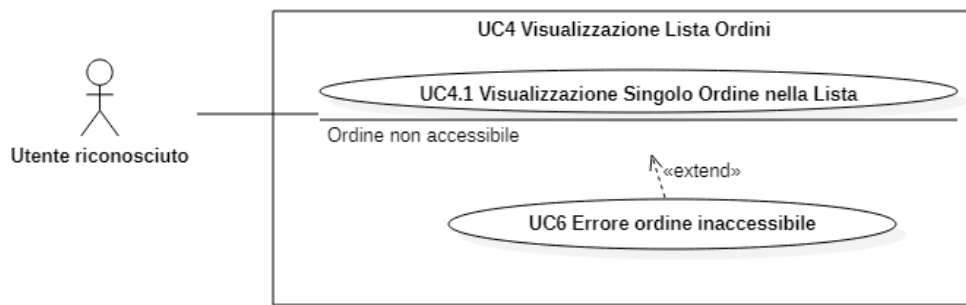


Figura 3.3: Use Case 4 - Visualizzazione Lista Ordini

UC4.1: Visualizzazione Singolo Ordine nella Lista

Attori Principali: Utente riconosciuto.

Precondizioni: L'Utente ha visualizzato la lista degli ordini relativa al suo indirizzo.

Postcondizioni: L'Utente ha visualizzato i dati specifici di un ordine.

Descrizione: L'Utente può visualizzare i dettagli di un ordine selezionato, inclusi il log degli stati e le eventuali operazioni disponibili.

Scenario Alternativo: Se l'Utente prova a visualizzare un ordine al quale non ha accesso, viene visualizzata una pagina di errore.

UC6: Errore Ordine Inaccessibile

Attori Principali: Utente riconosciuto.

Precondizioni: L'Utente è stato riconosciuto tramite la connessione al suo wallet.

Postcondizioni: L'Utente ha visualizzato la pagina d'errore.

Descrizione:

- L'Utente cerca di visualizzare i dettagli di un ordine inesistente o al quale non ha accesso;
- la webApp presenta una pagina d'errore che permette di tornare indietro.

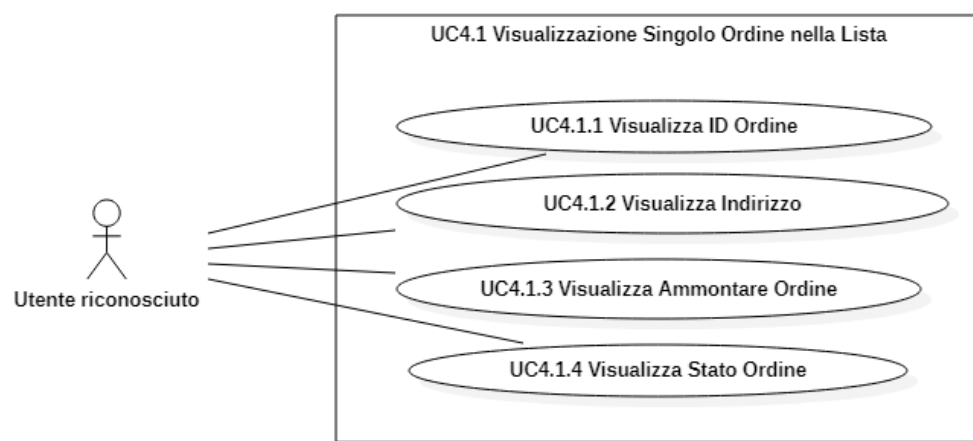


Figura 3.4: Use Case 4.1 - Visualizzazione Singolo Ordine nella Lista

UC4.1.1: Visualizzazione ID Ordine

Attori Principali: Utente riconosciuto.

Precondizioni: L'Utente si trova nella sezione "Visualizzazione Singolo Ordine nella Lista" (UC4.1) e desidera visualizzare l'id dell'ordine.

Postcondizioni: L'Utente ha visualizzato l'id dell'ordine.

Descrizione: L'Utente sta visualizzando i dettagli di un singolo ordine e tra i campi è presente l'id.

UC4.1.2: Visualizzazione Indirizzo

Attori Principali: Utente riconosciuto.

Precondizioni: L'Utente si trova nella sezione "Visualizzazione Singolo Ordine nella Lista" (UC4.1) e desidera visualizzare l'indirizzo dell'altro attore dell'ordine.

Postcondizioni: L'Utente ha visualizzato l'indirizzo dell'altro attore dell'ordine.

Descrizione: L'Utente sta visualizzando i dettagli di un singolo ordine e tra i campi è presente l'indirizzo dell'altro attore dell'ordine (venditore per il compratore e viceversa).

UC4.1.3: Visualizzazione Ammontare Ordine

Attori Principali: Utente riconosciuto.

Precondizioni: L'Utente si trova nella sezione "Visualizzazione Singolo Ordine nella Lista" (UC4.1) e desidera visualizzare l'ammontare dell'ordine.

Postcondizioni: L'Utente ha visualizzato l'ammontare dell'ordine.

Descrizione: L'Utente sta visualizzando i dettagli di un singolo ordine e tra i campi è presente l'ammontare dell'ordine.

UC4.1.4: Visualizzazione Stato Ordine

Attori Principali: Utente riconosciuto.

Precondizioni: L'Utente si trova nella sezione "Visualizzazione Singolo Ordine nella Lista" (UC4.1) e desidera visualizzare lo stato dell'ordine.

Postcondizioni: L'Utente ha visualizzato lo stato dell'ordine.

Descrizione: L'Utente sta visualizzando i dettagli di un singolo ordine e tra i campi è presente lo stato in cui si trova l'ordine.

UC5: Filtra Ordini

Attori Principali: Utente riconosciuto.

Precondizioni: L'Utente è stato riconosciuto tramite la connessione al suo wallet.

Postcondizioni: L'Utente ha visualizzato i dati relativi ad ordini che corrispondono ai requisiti di ricerca.

Descrizione: L'Utente può visualizzare in modo più ordinato le transazioni alle quali è interessato attraverso dei filtri.

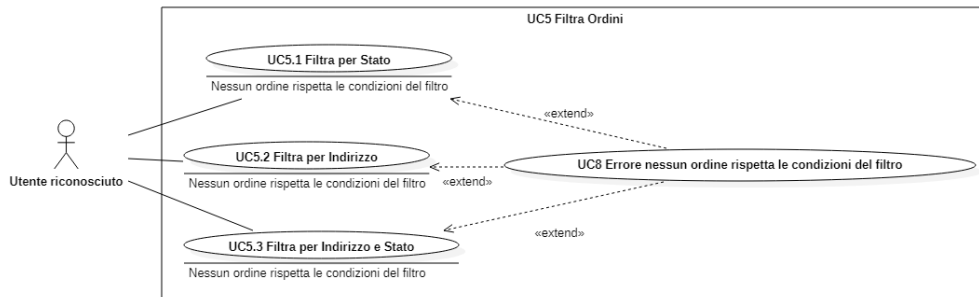


Figura 3.5: Use Case 5 -Filtra Ordini

UC5.1: Filtra per Stato

Attori Principali: Utente riconosciuto.

Precondizioni: L'Utente è stato riconosciuto tramite la connessione al suo wallet.

Postcondizioni: L'Utente ha visualizzato i dati relativi ad ordini che corrispondono ai requisiti di ricerca.

Descrizione:

- L'Utente seleziona lo stato degli ordini che vuole visualizzare;
- l'Utente clicca sul bottone "Applica Filtri".

Scenario Alternativo: Se l'Utente applica un filtro che non viene rispettato da nessun ordine, viene visualizzato un errore.

UC5.2: Filtra per Indirizzo

Attori Principali: Utente riconosciuto.

Precondizioni: L'Utente è stato riconosciuto tramite la connessione al suo wallet.

Postcondizioni: L'Utente ha visualizzato i dati relativi ad ordini che corrispondono ai requisiti di ricerca.

Descrizione:

- L'Utente inserisce l'indirizzo degli ordini che vuole visualizzare;
- l'Utente clicca sul bottone "Applica Filtri".

Scenario Alternativo: Se l'Utente applica un filtro che non viene rispettato da nessun ordine, viene visualizzato un errore.

UC5.3: Filtra per Indirizzo e Stato

Attori Principali: Utente riconosciuto.

Precondizioni: L'Utente è stato riconosciuto tramite la connessione al suo wallet.

Postcondizioni: L'Utente ha visualizzato i dati relativi ad ordini che corrispondono ai requisiti di ricerca.

Descrizione:

- L'Utente inserisce l'indirizzo e seleziona lo stato degli ordini che vuole visualizzare;
- l'Utente clicca sul bottone "Applica Filtri".

Scenario Alternativo: Se l'Utente applica un filtro che non viene rispettato da nessun ordine, viene visualizzato un errore.

UC8: Errore nessun ordine rispetta le condizioni del filtro

Attori Principali: Utente riconosciuto.

Precondizioni: L'Utente ha già applicato un filtro le cui condizioni non sono rispettate da nessun ordine.

Postcondizioni: L'Utente ha visualizzato l'errore e può provare ad applicare un altro filtro.

Descrizione:

- L'Utente applica un filtro a cui non corrisponde nessun ordine;
- viene mostrato un messaggio di errore e la lista di tutti gli ordini dell'Utente.

UC7: Visualizzazione Dettagli Ordine

Attori Principali: Utente riconosciuto.

Attori Secondari: MetaMask.

Precondizioni: L'Utente è stato riconosciuto tramite la connessione al suo wallet.

Postcondizioni: L'Utente si trova nella pagina di visualizzazione dettagli del singolo ordine.

Descrizione:

- L'Utente clicca sul bottone "Vedi dettagli";
- l'Utente viene reindirizzato alla pagina di visualizzazione dettagli del singolo ordine.

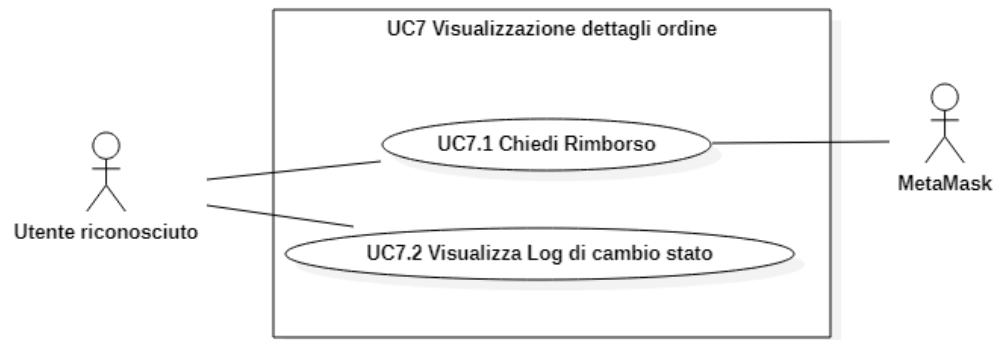


Figura 3.6: Use Case 7 - Visualizzazione Dettagli Ordine

UC7.1: Chiedi Rimborso

Attori Principali: Utente riconosciuto.

Attori Secondari: MetaMask.

Precondizioni: L'Utente si trova nella sezione "Visualizza Dettagli Singolo Ordine" (UC18) e desidera annullare uno degli acquisti sul proprio e-commerce.

Postcondizioni: L'Utente ha richiesto il reso.

Descrizione:

- L'Utente clicca sul bottone "Chiedi Reso";
- viene aperto il pop-up di MetaMask che richiede la conferma di una transazione con fee minima affinché venga emessa la richiesta di rimborso;
- lo stato dell'ordine passa da "Confirmed" a "Refund Asked".

UC7.2: Visualizza Log di Cambio Stato

Attori Principali: Utente riconosciuto.

Precondizioni: L'Utente si trova nella sezione "Visualizza Dettagli Singolo Ordine" (UC18) e desidera visualizzare il log di cambio stato dell'ordine.

Postcondizioni: L'Utente ha visualizzato il log di cambio stato dell'ordine.

Descrizione: L'Utente sta visualizzando i dettagli di un singolo ordine e sulla pagina è presente il log di cambio stati dell'ordine in cui è presente lo stato dell'ordine e il timestamp di quando l'ordine ha cambiato stato.

3.2 Tracciamento dei requisiti

Durante la fase di analisi dei requisiti e di individuazione degli use cases è stata stilata una tabella che tracci i requisiti in rapporto agli use cases. Sono stati individuati diversi tipi di requisiti e si è quindi fatto utilizzo di un codice identificativo per distinguerli. Il codice dei requisiti è così strutturato R[F/Q][N/D] dove:

R = Requisito;

F = Funzionale;

Q = Qualitativo;

N = Necessario (obbligatorio);

D = Desiderevole;

Le fonti dei requisiti possono essere le seguenti:

UC[numero]: indica un caso d'uso;

Committente: indica il capo del progetto.

Nelle tabelle 3.2 e 3.3 sono riassunti i requisiti e il loro tracciamento con gli use case delineati in fase di analisi.

Tabella 3.2: Tabella del tracciamento dei requisiti funzionali

Codice	Descrizione	Fonti
RFN 1	Landing page che permetta di completare il pagamento	UC1
RFN 1.1	L'utente deve potersi connettere al proprio wallet tramite MetaMask	UC1
RFN 1.2	L'utente deve poter visualizzare l'importo totale della transazione	UC2
RFN 1.3	L'utente deve poter visualizzare l'address del venditore	UC2
RFN 1.4	L'utente deve poter autorizzare la transazione, completando l'acquisto del prodotto	UC3
RFN 2	WebApp che permetta di visualizzare ed interagire con i propri ordini	UC4
RFN 2.1	L'utente deve potersi connettere al proprio wallet tramite MetaMask	UC1
RFN 2.2	L'utente deve poter visualizzare tutte le transazioni effettuate con ShopChain dai suoi indirizzi wallet collegati	UC4
RFN 2.3	L'utente deve poter filtrare le transazioni.	UC5
RFN 2.3.1	L'utente deve poter filtrare le transazioni in base al loro stato	UC5.1
RFN 2.3.2	L'utente deve poter filtrare le transazioni in base all'indirizzo	UC5.2
RFN 2.3.3	L'utente deve poter filtrare le transazioni in base al loro stato e all'indirizzo	UC5.3
RFN 2.4	L'utente deve poter visualizzare un singolo ordine della lista	UC4.1

Tabella 3.2: Tabella del tracciamento dei requisiti funzionali

Codice	Descrizione	Fonti
RFN 2.4.1	L'utente deve poter l'id di un singolo ordine	UC4.1.1
RFN 2.4.2	L'utente deve poter visualizzare l'indirizzo del venditore di un singolo ordine	UC4.1.2
RFN 2.4.3	L'utente deve poter visualizzare il prezzo di un singolo ordine	UC4.1.3
RFN 2.4.4	L'utente deve poter visualizzare lo stato di un singolo ordine	UC4.1.4
RFN 2.5	L'utente deve poter chiedere il rimborso di un ordine	UC7.1
RFN 2.6	L'utente deve poter visualizzare il log di cambio stati di un ordine	UC7.2

Tabella 3.3: Tabella del tracciamento dei requisiti qualitativi

Codice	Descrizione	Fonti
RQN 1	Deve essere fornito un documento tecnico che descriva le maschere	Committente
RQN 2	Il codice deve essere condiviso su una repository pubblica	Committente
RQD 3	L'applicazione web deve essere accessibile a utenti con disabilità	Committente
RQD 4	L'applicazione web deve essere responsive	Committente

Capitolo 4

Progettazione e codifica

In questo capitolo vengono descritte in primo luogo le tecnologie utilizzate durante il progetto e in secondo luogo la progettazione e l'implementazione delle maschere e dei servizi dell'applicazione.

4.1 Tecnologie

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

TypeScript

TypeScript è un linguaggio di programmazione open source che basa le sue caratteristiche su ECMAScript 6. Il linguaggio estende la sintassi del linguaggio JavaScript facendo sì che qualsiasi programma scritto in JavaScript possa funzionare anche con TypeScript. Tra le principali funzionalità aggiuntive di TypeScript ci sono la possibilità di tipizzare le variabili e di creare interfacce e classi.[9]

Angular

Angular è un framework JavaScript open source per creare applicazioni web dinamiche grazie a una serie di funzionalità e strumenti forniti dallo stesso. L'architettura modulare consente di strutturare al meglio un'applicazione web e di avere un elevato riutilizzo del codice. Permette inoltre un'elevata manutenibilità in quanto ogni componente è adibito ad un'unica funzione. Utilizzato per sviluppare l'applicazione web.[1]

Angular Material

Angular Material è una libreria grafica creata appositamente per Angular ed è basata sullo stile grafico di Google. Mette a disposizione componenti grafiche già implementate e testate anche dal punto di vista dell'accessibilità. Utilizzato per sviluppare la grafica delle pagine web.[3]

Node.js

Node.js è un framework utilizzato da Angular per gestire le dipendenze. Permette di dichiarare due insiemi di dipendenze: per gli sviluppatori e per far funzionare l'applicativo. In questo modo è possibile differenziare quali librerie si possono tralasciare in fase di deploy dell'applicazione perché, ad esempio, necessarie solo per effettuare i test. Permette inoltre, usando dei semplici comandi, di scaricare all'interno del progetto le librerie necessarie, funzionalità molto utile in fase di [Continuous Integration/Continuous Delivery \(CI/CD\)](#)^{[gl].[8]}

4.2 Progettazione

4.2.1 Architettura di Angular

Il componente principale di Angular è il modulo, che raggruppa un insieme di funzionalità legate tra loro. Ogni modulo contiene un component alla quale è associato un template.

Un component è una classe che si occupa di gestire la vista e definire la logica del codice, mentre il template è la vista stessa dove viene definito codice HTML per visualizzare i dati contenuti nel component. Questi due elementi lavorano a stretto contatto con altri due componenti di Angular rispettivamente: servizi e direttive. I servizi vengono richiamati dai component e svolgono compiti ben precisi come ad esempio la gestione dell'autenticazione utente.

Le direttive vengono richiamate dai template e permettono di personalizzare il codice HTML creando dei tag propri che è poi possibile riutilizzare in diverse viste.

Ogni component può contenere diversi component figli, in questo modo si può creare una maschera in modo modulare. Si può creare un component padre che rappresenta la pagina completa e poi diversi figli per ogni elemento contenuto in quella pagina, per esempio un component per il menu, un component per la sezione delle notizie e così via. Ognuno dei component figli si occuperà così di gestire la grafica di quella determinata funzionalità e di chiamare i servizi di cui necessita e sarà poi compito del padre mettere i figli insieme. Così facendo, si rende la struttura dell'applicazione più ordinata e il codice più mantenibile.

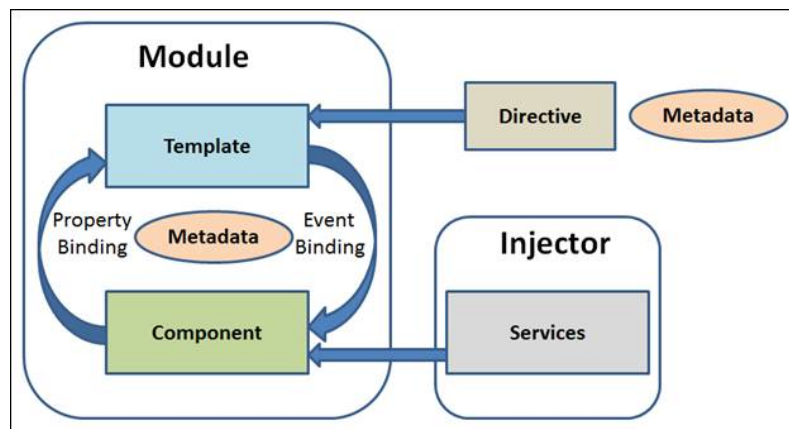


Figura 4.1: Schema logico dell'architettura di un'applicazione web basata su Angular [2]

4.2.2 Architettura dell'applicazione sviluppata

L'architettura dell'applicazione è stata progettata seguendo le best practice già consolidate nell'ambiente di Angular.

Ogni maschera dell'applicazione deve avere una propria cartella contenente il component, il template e il file di test ed eventualmente altri component figli.

Sempre in concordanza con le best practice sono state create due cartelle: Components e Services.

La cartella Components contiene tutte le cartelle dei componenti che costituiscono le maschere; la cartella Services contiene, invece, i service utilizzati dai componenti per dialogare con la blockchain.

4.2.3 Progettazione delle maschere

Successivamente allo studio e alla comprensione dei requisiti del progetto ShopChain, sono stati sviluppati i mock-up delle maschere delle pagine dell'applicazione web così da avere un feedback dal committente prima di iniziare l'implementazione.

Table					
Id ordine	Indirizzo venditore	Indirizzo acquirente	Ammontare ordine	Stato ordine	Dettagli ordine
value 1	value 2	value 3	value 4	value 5	value 6
...

Figura 4.2: Mock-up della pagina principale dell'applicazione web

In fase di progettazione delle maschere si è deciso che la struttura delle pagine deve essere simile ad altre applicazioni web di uso comune così da non disorientare l'utente. Tra le regole seguite:

- Link di navigazione nella parte superiore della pagina;
- I bottoni di eliminazione o di reset devono essere di colore differente rispetto agli altri bottoni, possibilmente di un colore tendente al rosso;
- I bottoni disabilitati devono avere un colore diverso dal loro stato normale così da far capire all'utente chiaramente quale sia il loro stato;
- I campi dei form che risultano errati devono essere segnalati in colore rosso e con un messaggio di errore per avvisare l'utente che per procedere devono essere corretti.

4.2.4 Progettazione dello smart contract

Il progetto ShopChain, essendo serverless, dialoga solamente con la blockchain Avalanche tramite lo smart contract del progetto stesso. Per questo percorso di tirocinio si è utilizzato lo smart contract del progetto ShopChain già esistente con delle piccole modifiche per togliere la funzionalità di conversione dei token ERC20 in stablecoin per rendere le transazioni più veloci da essere minate in blockchain.

4.3 Organizzazione dei componenti

Di seguito vengono riportati i diagrammi che raffigurano l'organizzazione dei componenti che costituiscono le viste principali dell'applicazione web, ovvero la landing page, in cui vengono effettuati i pagamenti, e la pagina principale, in cui è possibile visualizzare gli ordini dell'utente.

Landing page

Il diagramma dell'organizzazione dei componenti della landing page è rappresentato in Figura 4.3. Il componente `LandingPageComponent` renderizza sempre il componente figlio `HeaderComponent`, mentre viene renderizzato un secondo componente figlio in base ai seguenti casi:

- viene visualizzato il componente `AccessComponent` se l'utente non ha ancora connesso il proprio wallet a MetaMask;
- viene visualizzato il componente `SwitchNetworkComponent` se l'utente è collegato alla blockchain sbagliata;
- altrimenti viene visualizzato il template del componente `LandingPageComponent`.

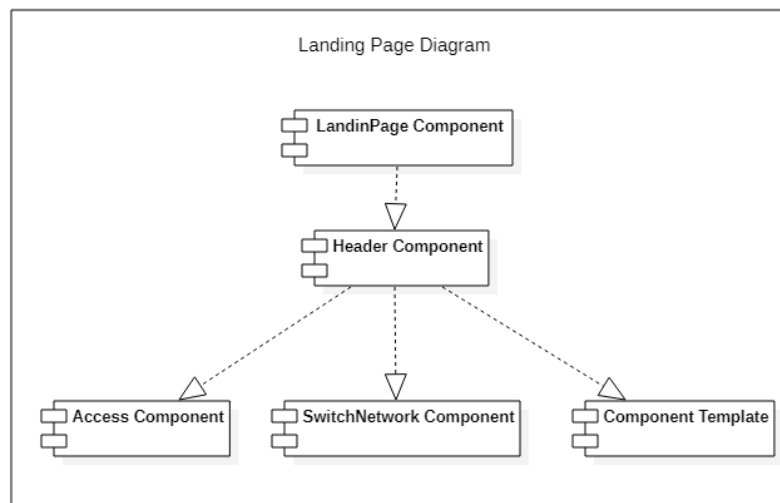


Figura 4.3: Diagramma dei componenti della landing page

Pagina principale

Il diagramma dell'organizzazione dei componenti della pagina principale è rappresentato in Figura 4.4. Il componente `BuyerComponent` renderizza sempre il componente figlio `HeaderComponent`, mentre viene renderizzato un secondo componente figlio in base ai seguenti casi:

- viene visualizzato il componente `AccessComponent` se l'utente non ha ancora connesso il proprio wallet a MetaMask;
- viene visualizzato il componente `SwitchNetworkComponent` se l'utente è collegato alla blockchain sbagliata;
- viene visualizzato il componente `OrderInfoComponent`, con al suo interno il componente `LogComponent`, se l'utente ha cliccato sul bottone di visualizzazione dei dettagli;
- altrimenti viene visualizzato il componente `OrdersComponent`.

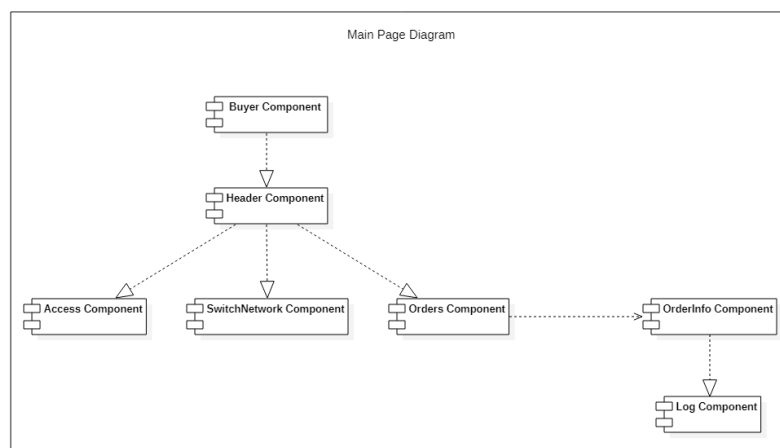


Figura 4.4: Diagramma dei componenti della pagina principale

4.4 Design Pattern implementati

Durante l'implementazione delle maschere di front-end sono stati utilizzati alcuni design pattern già integrati nel framework Angular per una maggiore efficienza e sicurezza dell'applicativo.

Dependency Injection

Il dependency injection è un pattern già integrato con Angular per migliorare l'efficienza e la modularità. Il dependency injection usato da Angular è di tipo constructor, questo prevede che in ogni componente vengano dichiarati nel costruttore di quali servizi ha bisogno e in fase di inizializzazione tali dipendenze gli vengono fornite dall'esterno.

Singleton

Il singleton è un pattern già integrato con Angular che ha lo scopo di garantire che di un determinato servizio venga creata una e una sola istanza. Un altro vantaggio è che i servizi, condivisi tra più componenti, posso modificare il proprio stato in seguito ad una richiesta di un componente e condividere tale cambiamento con il resto dei componenti.

Guard Check

Il guard check è un pattern per la verifica degli input inseriti dall'utente, sia dal punto di vista sintattico sia dal punto di vista semantico. Questo pattern permette di avvisare l'utente se ha compilato in modo errato un campo di un form.

Observer

L'observer è un pattern che permette di rimanere in 'ascolto' di determinati eventi. Questo pattern è stato utilizzato con le chiamate allo smart contract e la creazione di transazione perché possono richiedere un tempo variabile in base al traffico presente sulla blockchain.

4.5 Codifica

Di seguito vengono descritte le maschere implementate con le loro funzionalità, i componenti coinvolti e i servizi utilizzati.

4.5.1 Maschere

Accesso

Questa pagina permette all'utente di eseguire l'accesso al proprio wallet tramite MetaMask e di accedere alle funzionalità dell'applicazione web. Se MetaMask è già collegato ad un wallet all'utente basterà premere il pulsante "Connetti Wallet" per accedere all'applicazione web.

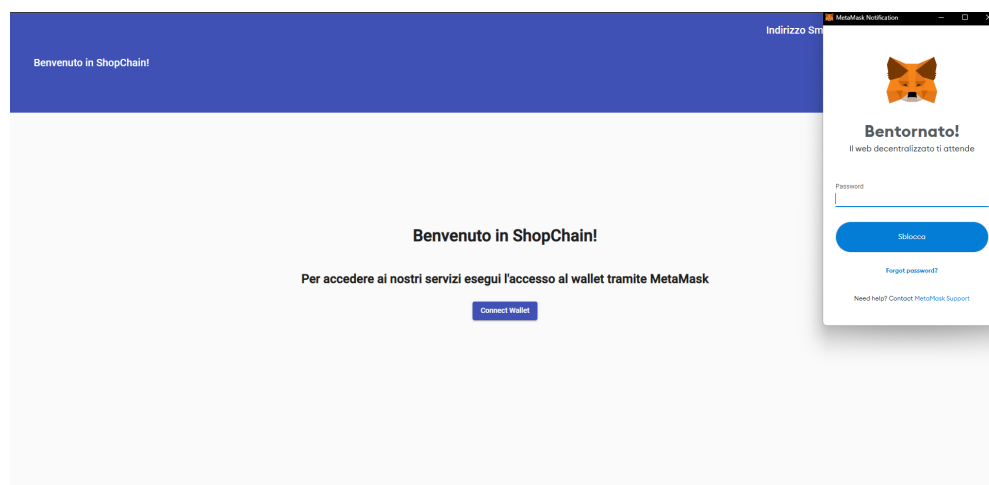


Figura 4.5: Pagina di autenticazione

Componenti coinvolti

- [HeaderComponent](#);
- [AccessComponent](#).

Servizi utilizzati

- [SmartContractService](#).

Landing Page

L'utente, dopo aver selezionato ShopChain come metodo di pagamento nell'e-commerce, viene reindirizzato in questa pagina. In questa pagina l'utente visualizza il riepilogo dell'ordine, cioè il prezzo da pagare e l'indirizzo del wallet del venditore. La pagina permette all'utente di creare la transazione, e di conseguenza l'ordine corrispondente, per effettuare il pagamento in criptovalute.

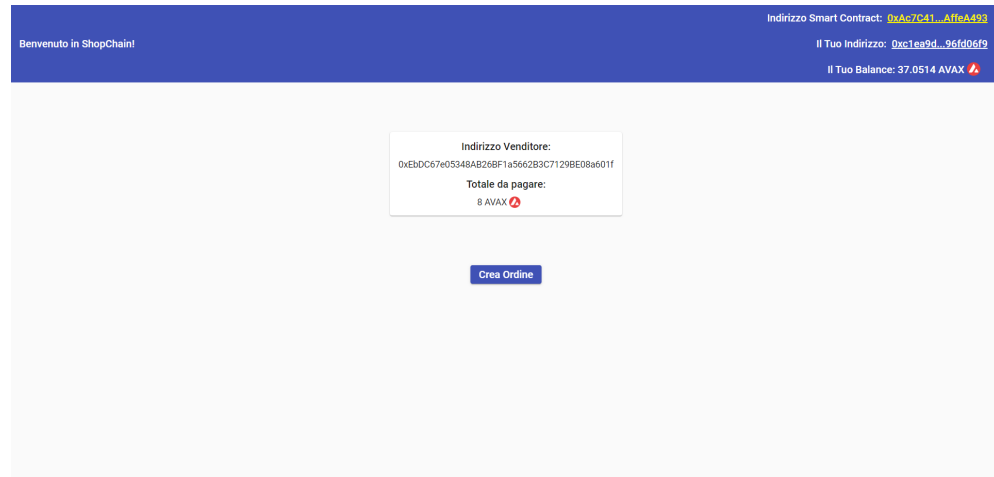


Figura 4.6: Pagina di pagamento e creazione ordine

Componenti coinvolti

- [HeaderComponent](#);
- [LandingPageComponent](#).

Servizi utilizzati

- [SmartContractService](#).

Home Page

In questa pagina l'utente può visualizzare la tabella con tutti gli ordini corrispondenti al wallet con cui è connesso a MetaMask. Di ogni ordine l'utente visualizza il suo numero univoco (Id), l'indirizzo del venditore del pacco collegato all'ordine, l'ammontare dell'ordine e lo stato in cui si trova; nella tabella è inoltre presente un pulsante per ogni ordine per visualizzare i dettagli dell'ordine corrispondente. Su questa lista di ordini l'utente può applicare i filtri per indirizzo e per stato, oltre che resettarli.

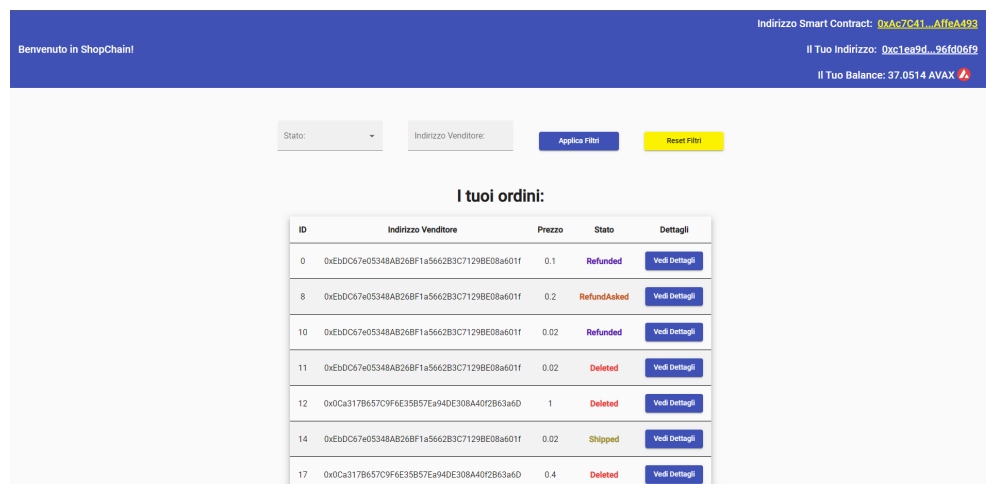


Figura 4.7: Home page di ShopChain

Componenti coinvolti

- [HeaderComponent](#);
- [BuyerComponent](#);
- [OrdersComponent](#).

Servizi utilizzati

- [SmartContractService](#).

Dettagli Ordine

In questa pagina l'utente può visualizzare la tabella con le informazioni dell'ordine selezionato e, se l'ordine è in stato *'Created'*, *'Shipped'* o *'Confirmed'*, è presente il bottone "Ask Refund" che permette all'utente di chiedere il reso al venditore dell'ordine selezionato. Nella parte inferiore della pagina è presente la tabella contenente i log di cambio stato dell'ordine contenente lo stato dell'ordine e il timestamp di quando l'ordine è entrato in quel preciso stato.

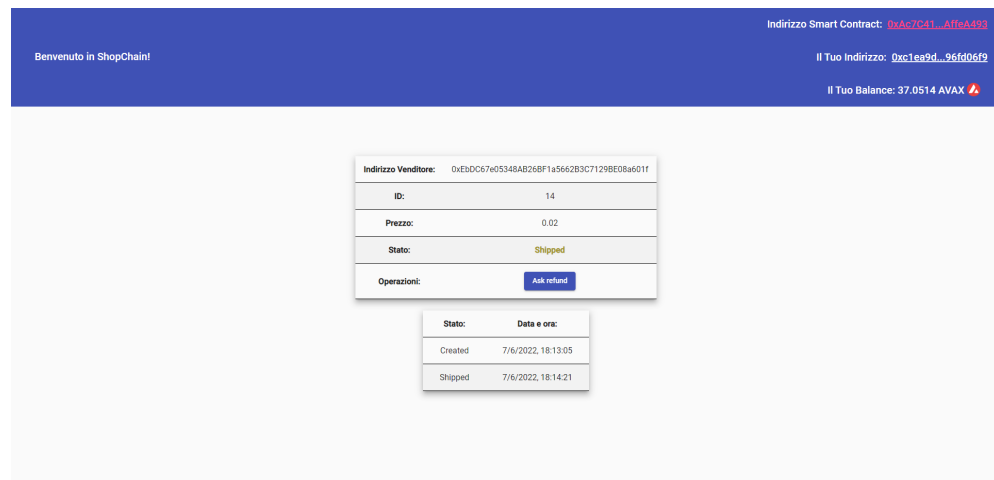


Figura 4.8: Pagina dei dettagli di un ordine

Componenti coinvolti

- [HeaderComponent](#);
- [OrderInfoComponent](#);
- [LogComponent](#).

Servizi utilizzati

- [SmartContractService](#).

Switch Network

Questa pagina di errore permette all'utente di collegarsi alla blockchain corretta tramite il bottone "Switch Network" tramite MetaMask.

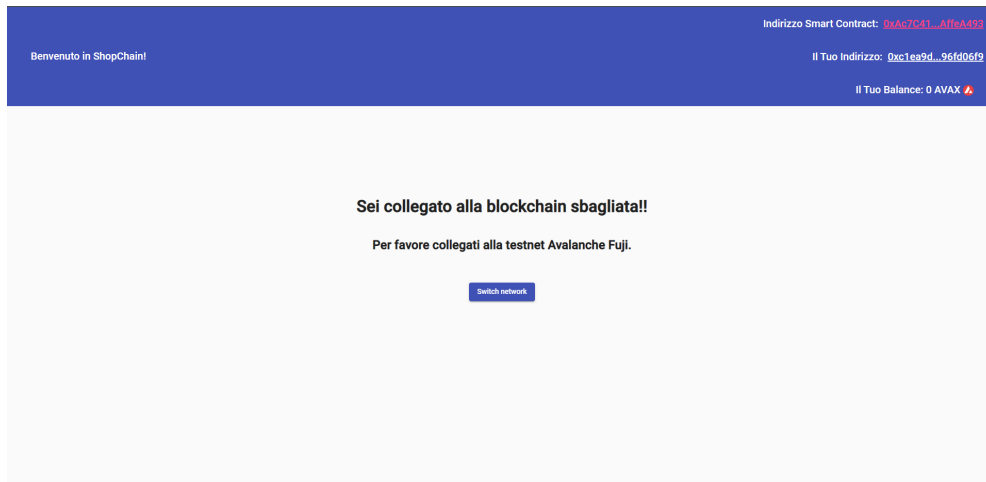


Figura 4.9: Pagina di errore: connessione a blockchain sbagliata

Componenti coinvolti

- [HeaderComponent](#);
- [SwitchNetworkComponent](#).

Servizi utilizzati

- [SmartContractService](#).

4.5.2 Componenti

AccessComponent

Questo componente permette all'utente **non** autenticato di autenticarsi tramite MetaMask e di collegare il proprio wallet all'applicazione web.

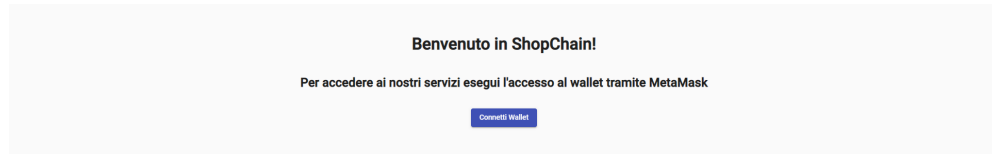


Figura 4.10: Componente AccessComponent

Servizi usati

- [SmartContractService](#).

Metodi

- **connectWallet:** effettua, tramite SmartContractService, la connessione al wallet dell'utente tramite MetaMask.

BuyerComponent

Questo componente fa da contenitore ad una serie di altri componenti. Le funzionalità fornite sono:

- Visualizzazione indirizzo smart contract di ShopChain, indirizzo wallet collegato e balance wallet collegato;
- Visualizzazione lista ordini correlati al wallet collegato;
- Filtraggio degli ordini per indirizzo venditore e stato dell'ordine.

Componenti

- [HeaderComponent](#);
- [OrdersComponent](#).

Servizi usati

- [SmartContractService](#).

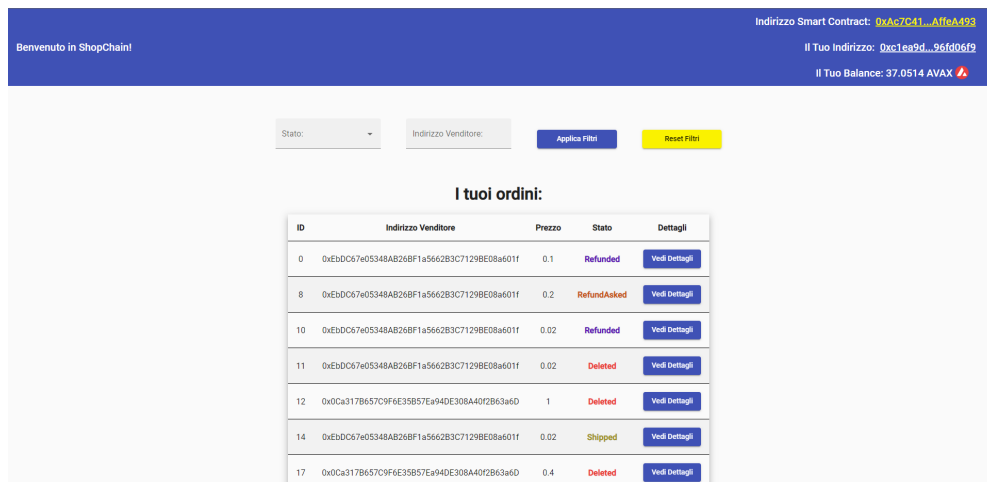


Figura 4.11: Componente BuyerComponent

Metodi

- **ngOnInit:** all'inizializzazione del componente viene controllato, mediante il servizio `SmartContractService`, se l'utente è già collegato tramite MetaMask e se si trova sulla blockchain giusta;
- **hasConnected:** event handler che risponde all'evento generato dal bottone "Connetti wallet" di `AccessComponent`.

HeaderComponent

Questo componente permette all'utente di visualizzare l'indirizzo e il balance del wallet collegato a MetaMask e fornisce anche un collegamento alla pagina web dell'explorer della blockchain corrispondente allo smart contract di ShopChain, per una maggiore trasparenza verso l'utente.



Figura 4.12: Componente HeaderComponent

Servizi usati

- `SmartContractService`.

Metodi

- **ngOnInit:** all'inizializzazione del componente vengono inizializzate le variabili contenenti l'indirizzo e il bilancio del wallet attualmente collegato a MetaMask tramite il servizio `SmartContractService`.

LandingPageComponent

Questo componente permette all'utente di visualizzare i dati dell'ordine di cui sta per effettuare il pagamento e permette all'utente di creare una transazione corrispondente all'ordine per completare il pagamento.

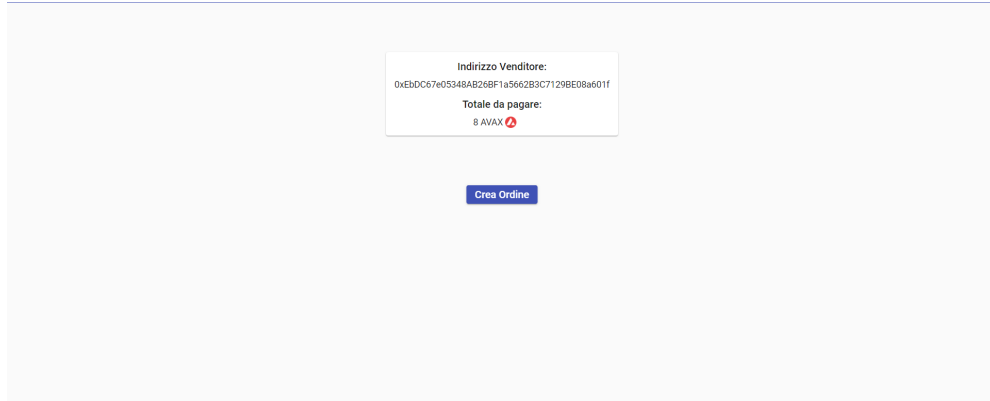


Figura 4.13: Componente LandingPageComponent

Servizi usati

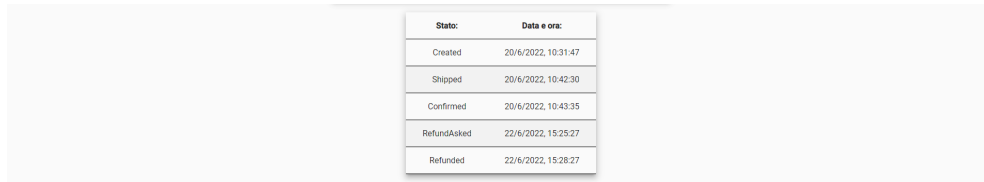
- [SmartContractService](#).

Metodi

- **ngOnInit:** all'inizializzazione del componente viene controllato, mediante il servizio [SmartContractService](#), se l'utente è già collegato tramite MetaMask e se si trova sulla blockchain giusta, inoltre recupera le informazioni dell'ordine dal server dell'e-commerce;
- **hasConnected:** event handler che risponde all'evento generato dal bottone "Connetti wallet" di [AccessComponent](#);
- **fetchOrder:** metodo che recupera le informazioni dell'ordine di cui l'utente deve effettuare il pagamento dal server dell'e-commerce;
- **createOrder:** metodo che crea la transazione da pagare tramite MetaMask al premere del bottone "Crea Ordine".

LogComponent

Questo componente permette all'utente di visualizzare la tabella di cambio stato di un ordine.



Stato:	Data e ora:
Created	20/6/2022, 10:31:47
Shipped	20/6/2022, 10:42:30
Confirmed	20/6/2022, 10:43:35
RefundAsked	22/6/2022, 15:25:27
Refunded	22/6/2022, 15:28:27

Figura 4.14: Componente LogComponent

Servizi usati

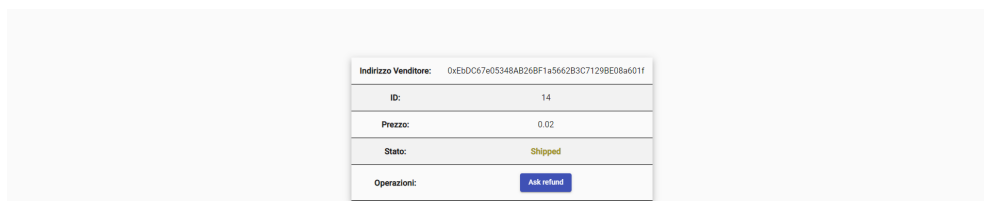
- [SmartContractService](#).

Metodi

- **ngOnInit**: all'inizializzazione del componente viene recuperato il log degli stati dell'ordine selezionato mediante il servizio SmartContractService.

OrderInfoComponent

Questo componente permette all'utente di visualizzare i dettagli di un ordine e le operazioni da eseguire disponibili su quel determinato ordine.



Indirizzo Venditore: 0xEbDC67e05348AB268F1a5662B3C71298E08a601f	
ID:	14
Prezzo:	0.02
Stato:	Shipped
Operazioni:	Ask refund

Figura 4.15: Componente OrderInfoComponent

Servizi usati

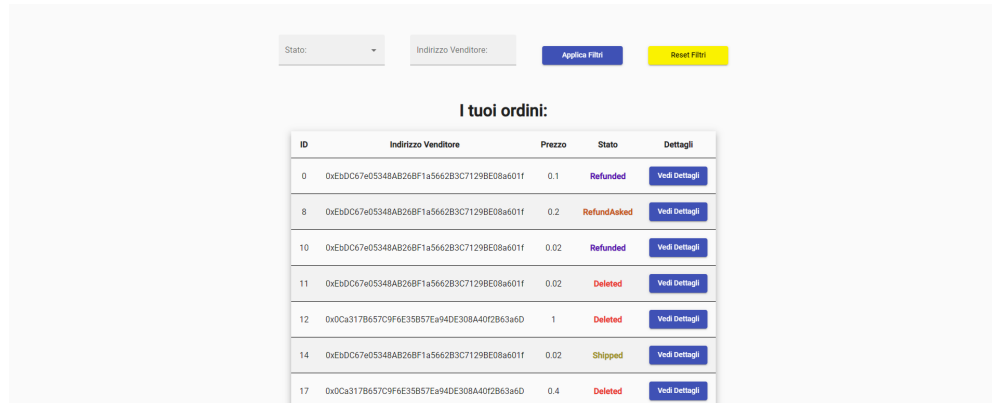
- [SmartContractService](#).

Metodi

- **ngOnInit**: all'inizializzazione del componente vengono recuperate le informazioni dell'ordine selezionato mediante il servizio SmartContractService;
- **askRefund**: metodo che permette all'utente di richiedere il reso di un ordine tramite MetaMask.

OrdersComponent

Questo componente permette all'utente di visualizzare la lista degli ordini correlati al wallet con cui è collegato a MetaMask e permette di applicare i filtri per indirizzo e per stato dell'ordine.



The screenshot shows the 'OrdersComponent' interface. At the top, there are two input fields: 'Stato:' with a dropdown arrow and 'Indirizzo Venditore:'. To the right of these fields are two buttons: 'Applica Filtri' (blue) and 'Reset Filtri' (yellow). Below the filters, the text 'I tuoi ordini:' is displayed. Underneath is a table with the following data:

ID	Indirizzo Venditore	Prezzo	Stato	Dettagli
0	0xE8DC67e05348AB26BF1a5662B3C7129BE08a601f	0.1	Refunded	Vedi Dettagli
8	0xE8DC67e05348AB26BF1a5662B3C7129BE08a601f	0.2	RefundAsked	Vedi Dettagli
10	0xE8DC67e05348AB26BF1a5662B3C7129BE08a601f	0.02	Refunded	Vedi Dettagli
11	0xE8DC67e05348AB26BF1a5662B3C7129BE08a601f	0.02	Deleted	Vedi Dettagli
12	0x0Ca317B657C9F6E35B57Ea94DE308A4072B63a6D	1	Deleted	Vedi Dettagli
14	0xE8DC67e05348AB26BF1a5662B3C7129BE08a601f	0.02	Shipped	Vedi Dettagli
17	0x0Ca317B657C9F6E35B57Ea94DE308A4072B63a6D	0.4	Deleted	Vedi Dettagli

Figura 4.16: Componente OrdersComponent

Componenti

- [HeaderComponent](#);
- [LogComponent](#).

Servizi usati

- [SmartContractService](#).

Metodi

- **ngOnInit**: all'inizializzazione del componente vengono recuperate le informazioni del wallet collegato e degli ordini correlati ad esso mediante il servizio SmartContractService;
- **getOrders**: metodo che recupera gli ordini dell'utente dallo smart contract;
- **filterOrders**: metodo che applica i filtri selezionati dall'utente sulla lista degli ordini e mostra all'utente la lista di ordini che rispettano i filtri scelti, altrimenti un messaggio di errore e la lista completa degli ordini;
- **resetFilters**: metodo che toglie i filtri selezionati dall'utente e mostra all'utente la lista completa degli ordini.

SwitchNetworkComponent

Questo componente permette all'utente di collegarsi alla blockchain corretta tramite MetaMask nel caso in cui egli sia collegato alla blockchain sbagliata.



Figura 4.17: Componente SwitchNetworkComponent

Servizi usati

- [SmartContractService](#).

Metodi

- **changeNetwork**: metodo che permette all'utente di collegarsi alla blockchain corretta tramite il servizio SmartContractService.

4.5.3 Servizi

SmartContractService

Questo servizio si occupa di gestire la comunicazione tra l'applicazione web e lo smart contract di ShopChain, presentando quindi tutti i metodi dello smart contract utili all'utente.

Metodi

- **getWebProvider**: metodo che istanzia un oggetto di tipo *Web3Provider*, utile per la comunicazione con MetaMask;
- **getContract**: metodo che istanzia lo smart contract di ShopChain per renderlo visibile all'interno dell'applicazione web;
- **connectWallet**: metodo che connette il wallet dell'utente all'applicazione web tramite MetaMask;
- **isRightChain**: metodo che controlla se l'utente è collegato alla blockchain corretta;
- **setCurrentAddress**: metodo che salva in apposite variabili l'indirizzo e il bilancio del wallet attualmente collegato a MetaMask;
- **getOrdersOfUser**: metodo che recupera dallo smart contract gli ordini correlati al wallet attualmente collegato;
- **getOrderById**: metodo che ritorna l'ordine corrispondente all'id preso in input;
- **getLog**: metodo che ritorna il log di cambio stato dell'ordine corrispondente all'id preso in input;

- **createOrder**: metodo che crea una transazione in blockchain e l'ordine ad essa corrispondente nello smart contract;
- **askRefund**: metodo che chiede il reso dell'ordine corrispondente all'id preso in input;
- **listenerAccountChange**: metodo che resta in ascolto del cambio wallet collegato a MetaMask;
- **listenerNetworkChange**: metodo che resta in ascolto del cambio della blockchain a cui è collegato l'utente;
- **changeNetwork**: metodo che permette all'utente di collegarsi alla blockchain corretta tramite MetaMask.

4.6 Problematiche riscontrate

Durante l'implementazione del servizio SmartContractService non si riusciva ad accedere all'oggetto `window.ethereum`, oggetto che viene iniettato da MetaMask ed utile a comunicare con blockchain basate su Ethereum. Dopo un'attenta ricerca e dopo aver consultato dei colleghi si è scoperto che Angular disincentiva l'uso diretto dell'oggetto `window` e, di conseguenza, tutti gli oggetti da esso derivati. A questo punto sono state trovate due soluzioni possibili al problema:

1. Creare un servizio dedicato che, tramite una funzione esterna, avvolga l'oggetto `window` e lo ritorni tramite un metodo pubblico (Figura 4.18);
2. Utilizzare il package di npm `@metamask/detect-provider` per rilevare il provider iniettato nell'oggetto `window.ethereum` se MetaMask è installato come estensione del browser utilizzato dall'utente (Figura 4.19).

Dopo essermi confrontato con dei colleghi e con il tutor aziendale si è optato per la soluzione 2 perché viene consigliata da molti sviluppatori online e nella documentazione di MetaMask stessa, inoltre la soluzione 1 non è molto elegante ed è solo uno stratagemma per aggirare il problema.[6]

```
1  import { Injectable } from '@angular/core';
2  function _window(): any {
3    return window;
4  }
5  @Injectable()
6  export class WindowRef {
7    get nativeWindow(): any {
8      return _window();
9    }
10 }
```

Figura 4.18: Esempio soluzione 1

```
1  // This function detects most providers injected at window.ethereum
2  import detectEthereumProvider from '@metamask/detect-provider';
3
4  const provider = await detectEthereumProvider();
5
6  if (provider) {
7    // From now on, this should always be true:
8    // provider === window.ethereum
9    startApp(provider); // initialize your app
10 } else {
11   console.log('Please install MetaMask!');
12 }
```

Figura 4.19: Esempio soluzione 2

Capitolo 5

Verifica e validazione

5.1 Accessibilità

5.1.1 Attributi HTML

Per rendere l'applicazione web più accessibile sono stati utilizzati diversi attributi HTML specifici per l'accessibilità:

- **scope**: utilizzato per facilitare la lettura delle tabelle agli screen reader;
- **summary**: utilizzato per riassumere il contenuto di una tabella di piccole dimensioni;
- **title**: utilizzato per descrivere gli elementi che altrimenti non sarebbero leggibili dagli screen reader, totalmente o parzialmente;
- **lang**: utilizzato per indicare agli screen reader in che lingua leggere una parola o una frase diversa dall'italiano.

5.1.2 Colori

I colori delle maschere sono stati selezionati per passare il test WCAG AA che richiede un rapporto di contrasti di almeno 4.5:1 per il testo normale e 3:1 per il testo in grassetto. Di seguito è riportata la tabella dei colori del testo e del loro sfondo e il loro rapporto di contrasto.

Tabella 5.1: Rapporto contrasto testo-sfondo

Colore testo	Colore sfondo	Rapporto
#FFFFFF	#3F51B5	6.87:1
#000000	#FFFFFF	21:1
#000000	#F2F2F2	18.75:1
#FAF200	#3F51B5	5.81:1
#000000	#FAF200	17.76:1
#CF1507	#FFFFFF	5.58:1

Altri elementi di accessibilità

- Sono stati rimossi i link circolari per evitare confusione negli utenti e permettere di identificare univocamente il percorso raggiunto;
- Si sono evitati testi scorrevoli, lampeggianti, barrati ed in generale font troppo elaborati per agevolare la lettura;
- Sono stati utilizzati gli attributi [Web Accessibility Initiative - Accessible Rich Internet Applications \(WAI-ARIA\)](#)^[8] per specificare meglio alcuni elementi:
 - **role**: usato per definire la funzione di alcuni tag;
 - **aria-hidden**: usato per nascondere elementi inutili agli screen reader perché già presenti descrizioni testuali.

5.2 Validazione e collaudo

È stata organizzata settimanalmente una riunione di allineamento con tutte le persone coinvolte nel progetto. In queste riunioni si otteneva un feedback sul lavoro svolto fino a quel momento e una lista dei possibili cambiamenti da fare per raggiungere l'obiettivo voluto.

Durante lo svolgimento del progetto sono stati definiti dei test di accettazione che l'applicazione deve passare per essere ritenuta conclusa.

Di seguito vengono elencati i test di accettazione per il front-end del progetto. Il codice identificativo dei test è così strutturato TA[numero], dove:

- **TA**: test di accettazione;
- **[numero]**: indica un numero progressivo per identificare univocamente il test.

Tabella 5.2: Test di accettazione

ID	Descrizione	Stato
TA1	La landing page deve ricevere le informazioni dell'ordine appena creato nell'e-commerce	✓
TA2	La landing page deve creare una transazione corrispondente all'ordine creato nell'e-commerce	✓
TA3	Una volta confermata la transazione, la landing page deve poter reindirizzare l'utente alla lista dei suoi ordini	✓
TA4	L'applicazione deve mostrare la lista degli ordini correlati al wallet attualmente connesso a MetaMask	✓
TA5	L'utente deve poter filtrare gli ordini per stato, per indirizzo venditore o per entrambi	✓
TA6	L'utente deve poter richiedere il reso solo degli ordini in stato <i>Created</i> , <i>Shipped</i> e <i>Confirmed</i>	✓

L'ultima settimana di stage è stata organizzata una presentazione finale dell'applicazione web per collaudare tutte le funzionalità richieste e verificare che superi i test di accettazione. Alla fine della presentazione è stata effettuata anche una revisione del codice sorgente insieme al tutor aziendale, superata con successo.

Capitolo 6

Conclusioni

6.1 Raggiungimento degli obiettivi

Per quanto riguarda gli obiettivi prefissati all'inizio dello stage (sezione 2.2) risultano tutti raggiunti. Di seguito la tabella riassuntiva.

Tabella 6.1: Riepilogo obiettivi tirocinio

Obiettivo	Stato
O01	Soddisfatto
O02	Soddisfatto
O03	Soddisfatto
D01	Soddisfatto
F01	Soddisfatto

6.2 Conoscenze acquisite

Grazie a questo percorso di tirocinio si è potuto approfondire e apprendere un gran numero di argomenti che riguardano le applicazione web decentralizzate, ovvero basate su blockchain. Oltre a ciò, l'esperienza di stage è stata utile per apprendere un metodo lavorativo reale, derivante dal modus operandi dell'azienda. Di seguito vengono riportate le principali conoscenze a livello professionale che ho acquisito con questo percorso.

Nuovi linguaggi di programmazione e framework All'inizio dello stage si è avuto modo di studiare e approfondire il framework **Angular**, nonché tutti gli strumenti ausiliari per la gestione grafica (*Angular Material* e *Bootstrap*). Con l'apprendimento del linguaggio **TypeScript** e l'approfondimento di **JavaScript**, mi sono ulteriormente avvicinato allo sviluppo di applicazioni web, scoprendo nuove funzionalità di questi linguaggi, quali gli *Observables* per la gestione degli eventi.

Nuove competenze progettuali Attraverso l'analisi e la progettazione delle macchine si sono messe in pratica le mie conoscenze per la realizzazione di componenti e servizi, facendo uso di opportuni **design patterns** supportati dal framework utilizzato per la codifica, in modo da ottimizzare al meglio l'applicazione web.

Nuovi strumenti di collaborazione Ho appreso in buona parte il funzionamento di repository *Git*, in particolare utilizzando *GitHub* per il versionamento del codice e la collaborazione con gli altri membri che lavoravano al progetto. Ho imparato, inoltre, ad utilizzare uno strumento per la gestione delle attività, ovvero *Trello*.

Apprendimento del metodo Scrum Nel corso del tirocinio ho appreso il metodo **Scrum** utilizzato dall'azienda e ho imparato principalmente a suddividere e organizzare le mie attività e i miei obiettivi affinché rientrassero nel tempo a mia disposizione. Ho fruito a pieno di tutte le risorse a mia disposizione e ho apprezzato i meeting settimanali di allineamento sullo svolgimento del progetto.

Approfondimento tecnologia blockchain Durante lo svolgimento del progetto ho avuto modo di confrontarmi con colleghi più esperti di me in ambito blockchain e, grazie al confronto con loro, ho potuto approfondire le mie conoscenze su questa tecnologia.

6.3 Valutazione personale

Nel corso del tirocinio ho maturato diverse idee riguardanti il rapporto tra il mondo universitario e il mondo del lavoro. Buona parte delle conoscenze del mondo universitario sono in gran parte essenziali, perché mi hanno aiutato ad avere un'impostazione mentale e uno spirito critico sul mio operato, ma alcuni degli argomenti trattati mi sono apparsi troppo vecchi rispetto alle necessità del mercato attuale. Questa esperienza lavorativa mi ha aiutato a capire la complementarità tra università e lavoro. Nel mondo del lavoro manca la rigidità organizzativa e metodica tipica dell'università. Tuttavia però, il mondo lavorativo è molto più dinamico e in continua evoluzione rispetto al mondo universitario, che rimane purtroppo bloccato in un pensiero più antico che può portare gli studenti a perdere interesse per certi corsi di studio che non stanno al passo con i tempi.

Questa attività di tirocinio, secondo me, è stata molto utile per avvicinarsi all'ambiente lavorativo reale e sono stato molto soddisfatto dell'azienda che mi ha ospitato e dei colleghi con cui ho avuto modo di collaborare. La loro disponibilità e quella del tutor aziendale è stata per me essenziale per avere dei punti di riferimento con cui delimitare il mio percorso di tirocinio.

Per quanto riguarda il percorso universitario, invece, ritengo che si siano affrontati argomenti fondamentali per formarsi ed inserirsi nel mondo del lavoro. La maggior parte delle lezioni che ho frequentato mi ha aiutato ad imparare come affrontare un progetto di gruppo e come risolvere le difficoltà. In aggiunta, le conoscenze teoriche e pratiche di progettazione e codifica, acquisite con i corsi universitari, sono state sufficienti nel progetto di stage per poter operare su nuovi linguaggi di programmazione e su framework sconosciuti.

La parte più carente che ho trovato nell'offerta formativa universitaria riguarda la poca formazione sull'utilizzo di strumenti di configurazione e di collaborazione. In particolare, faccio riferimento ad uno strumento molto famoso, come *Git*, che può essere adottato trasversalmente a varie discipline e che è molto utilizzato nel mondo del lavoro. Questa mancanza è purtroppo colmata solo da alcuni corsi a scelta o dallo studio autonomo dello studente, che però difficilmente riesce a conciliarlo con le ore di studio richieste dai corsi universitari. Una seconda carenza l'ho ritrovata nella poca modernità di alcuni contenuti riportati in alcuni corsi, a discapito dello studio

di tecnologie e argomenti più moderni e attuali. Questo rischia di lasciare il mondo universitario bloccato a tecnologie antiche e non più in uso, e che quindi diventa quasi inutile studiarle se non per fini storici.

Concludendo questa esperienza di tirocinio per me è stata molto formativa ed utile a livello personale non solo perché mi ha fatto capire i punti di forza e i punti di debolezza del mio carattere e come valorizzarli al meglio, ma anche perché mi ha aiutato a capire meglio quali possono essere i miei ambiti e obiettivi a livello lavorativo.

Glossario

Angular *Angular* è un framework open source per lo sviluppo di applicazioni web con licenza MIT, evoluzione di AngularJS, sviluppato principalmente da Google. [v](#)

Continuous Integration/Continuous Delivery (CI/CD) *Continuous Integration/Continuous Delivery* è un metodo per la distribuzione frequente delle applicazioni ai clienti, che prevede l'introduzione dell'automazione nelle fasi di sviluppo dell'applicazione. Principalmente, si basa sui concetti di integrazione, distribuzione e deployment continui.. [24](#), [51](#)

ERC20 *ERC20* è lo standard tecnico dietro gli smart contract che servono per l'implementazione dei token sulla blockchain di Ethereum. Definisce una struttura dati comune a tutti i token.. [1](#)

MetaMask *MetaMask* è un'estensione browser scaricabile e che permette la connessione del proprio browser alla piattaforma Ethereum e alle applicazioni decentralizzate basate su di essa; consente inoltre di accedere a wallet Ethereum per lo scambio di token ERC20.. [3](#), [11](#)

UML in ingegneria del software *UML, Unified Modeling Language* (ing. linguaggio di modellazione unificato) è un linguaggio di modellazione e specifica basato sul paradigma object-oriented. L'*UML* svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa tale linguaggio per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. [51](#)

Web Accessibility Initiative - Accessible Rich Internet Applications (WAI-ARIA)

WAI-ARIA descrive come aggiungere una semantica o altri metadati al contenuto HTML allo scopo di rendere i controlli lato utente e i contenuti dinamici più accessibili.. [51](#)

Acronimi

CI/CD [Continuous Integration/Continuous Delivery](#). 49

UML [Unified Modeling Language](#). 12, 49

WAI-ARIA [Web Accessibility Initiative - Accessible Rich Internet Applications](#). 44,
49

Bibliografia

Siti web consultati

- [1] *Angular*. URL: <https://angular.io/> (cit. a p. 23).
- [2] *Angular Architecture*. URL: <https://www.c-sharpcorner.com/article/basic-architecture-of-angular-2-applications/> (cit. a p. 24).
- [3] *Angular Material*. URL: <https://material.angular.io/> (cit. a p. 23).
- [4] *Architettura Decentralized Application*. URL: <https://medium.com/hexamount/architecting-modern-decentralized-applications-52b3ac3baa5a> (cit. a p. 3).
- [5] *Engineering Software Architectures of Blockchain-Oriented Applications*. URL: <https://ieeexplore.ieee.org/abstract/document/8432174> (cit. a p. 3).
- [6] *Ethereum Provider API*. URL: <https://docs.metamask.io/guide/ethereum-provider.html#table-of-contents> (cit. a p. 41).
- [7] *IPFS*. URL: <https://ipfs.io/> (cit. a p. 3).
- [8] *Node.js*. URL: <https://nodejs.org/it/> (cit. a p. 24).
- [9] *TypeScript*. URL: <https://www.typescriptlang.org/> (cit. a p. 23).