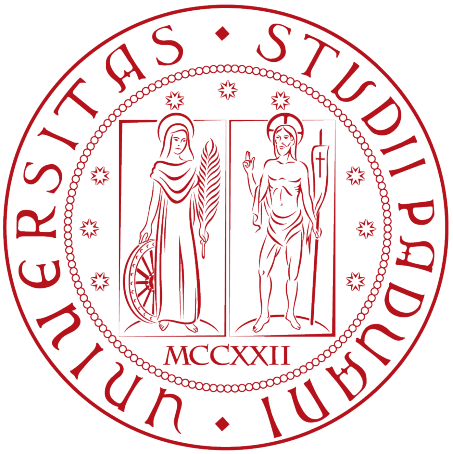
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Studio e implementazione di un front-end per Blockchain con il framework Angular

Tesi di laurea

Relatore

Prof. Luigi De Giovanni

Laureando

Filippi Gabriele

Anno Accademico 2021-2022

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecento ore, dal laureando Gabriele Filippi presso l'azienda Synclab S.p.A. In conformità con gli obiettivi e la pianificazione dello stage, riportati nella prima parte del documento, le attività svolte hanno in primo luogo richiesto uno studio approfondito delle tecnologie utilizzate, in particolare Typescript$_g$, il *framework* Angular e la BlockChain$_g$. Lo stage richiedeva lo sviluppo di un *front-end* per la gestione di transazioni tramite criptovalute$_g$. Nella tesi sono descritti i risultati delle attività di analisi dei requisiti, progettazione e codifica, con particolare riferimento alla struttura offerta dal *framework* utilizzato. Le attività dello stage hanno portato a conseguire tutti gli obiettivi richiesti.

Indice

Elenco delle figure

Elenco delle tabelle

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*$_{\text{g}}$;
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 1

Introduzione

In questo capitolo viene introdotta l'azienda in cui si è svolto il periodo di stage. Si introduce anche il progetto scelto descrivendone la tecnologia alla base.

1.1 L'azienda

Sync Lab nasce come *Software house* tramutatasi rapidamente in *System Integrator* attraverso un processo di maturazione delle competenze tecnologiche, metodologiche ed applicative nel dominio del *software* [21]. L'azienda propone sul mercato interessanti quanto innovativi prodotti *software*, nati nel proprio laboratorio di ricerca e sviluppo. Attraverso questi prodotti, Sync Lab ha gradualmente conquistato significative fette di mercato nei settori *mobile*, videosorveglianza e sicurezza delle infrastrutture informatiche aziendali.
Sono entrato in contatto con l'azienda durante lo "StageIT 2022". In questa occasione, l'ingegnere Fabio Pallaro ha segnalato la presenza di diversi progetti interessanti tra cui scegliere per il periodo di stage in azienda, in modo da permettere ai potenziali stagisti di lavorare sugli ambiti di maggiore interesse per loro. La presentazione è risultata più che convincente, attirando subito la mia attenzione. Successivamente, un colloquio individuale ha permesso di scegliere il progetto più adeguato ai miei interessi, ho così scelto il progetto "ShopChain".

1.2 Introduzione alla blockchain

Prima di parlare del progetto, è necessario introdurre la BlockChain e i concetti di transazione e di SmartContract$_g$. La BlockChain (letteralmente "catena di blocchi") sfrutta le caratteristiche di una rete informatica di nodi e consente di gestire e aggiornare, in modo univoco e sicuro, un registro contenente dati e informazioni (per esempio transazioni) in maniera aperta, condivisa e distribuita senza la necessità di un'entità centrale di controllo e verifica [5]. Le caratteristiche più importanti e interessanti per il progetto sono:

* **Decentralizzazione**: le informazioni vengono registrate distribuendole tra più nodi per garantire la resilienza dei sistemi;

* **Tracciabilità dei trasferimenti**: ciascun elemento sul registro è tracciabile in ogni sua parte, è infatti sempre possibile risalire all'esatta provenienza e destinazione;

* **Disintermediazione**: le piattaforme consentono di gestire le transazioni senza intermediari, ossia senza la presenza di enti centrali;

* **Trasparenza e Verificabilità**: il contenuto del registro risulta facilmente consultabile e visibile da tutti;

* **Immunitabilità del registro**: ogni modifica del registro richiede l'approvazione della rete, per questo non risulta possibile la sua manomissione.

Con il termine transazione [23], nella BlockChain, si intende lo scambio di criptovalute tra due *account*. Ogni transazione deve ricevere l'approvazione della rete per essere confermata. Per eseguire una transazione nella BlockChain sono necessari due account, uno con il ruolo di mittente e uno con il ruolo di ricevente. La creazione di un *account* è possibile a chiunque, in modo semplice, rapido e completamente gratuito. Oltre ai normali *account* controllati da utenti fisici, ne esistono di distribuiti nella rete che eseguono automaticamente le funzioni per cui sono stati programmati, questi sono gli SmartContract [20]. Gli SmartContract definiscono regole, come un normale contratto, e le impongono automaticamente quando un utente interagisce con essi, sfruttando il codice al loro interno. La possibilità di automatizzare le operazioni ha portato allo sviluppo delle così dette DApp_{g}, che offrono le stesse possibilità di un'app tradizionale con la differenza di non sfruttare server centralizzati ma le piattaforme BlockChain e i vari Network [8]. Le DApp risultano nettamente vincenti per quanto riguarda pagamenti e gestione dei dati. Infatti, gli applicativi tradizionali richiedono necessariamente dati come nome, cognome, indirizzo mail, numero di cellulare e dati di una carta o di un conto bancario. Nel contesto delle DApp, invece, un utente può gestire tutto tramite il suo *account* sulla BlockChain che, sfruttando le sue chiavi crittografiche uniche consente l'autenticazione senza richiedere alcun dato personale.

1.3 L'idea alla base del progetto

Proprio nel campo delle DApp nasce il progetto ShopChain. Negli ultimi anni infatti le piattaforme di $\text{e-commerce}_{\text{g}}$ sono aumentate a dismisura e con esse, purtroppo, è aumentata anche la possibiltà di cadere vittime di truffe *online*. La necessità di poter eseguire transazioni sicure è quindi più presente che mai e proprio in relazione a questo bisogno di mercato, nasce il progetto ShopChain. L'applicativo nasce per affiancarsi ad un qualsiasi e-commerce in tutti gli aspetti del pagamento, per renderlo sicuro e garantirlo tramite le classiche caratteristiche offerte da una DApp. In questo caso, la BlockChain svolge il ruolo di tramite tra compratore e venditore. L'ammontare della transazione viene infatti mantenuto come criptovaluta all'interno della BlockChain, per essere consegnato al venditore solo nel momento in cui l'acquirente verifica l'avvenuta ricezione. Nel momento della consegna del pacco l'acquirente dovrà necessariamente inquadrare il *Qr code* applicato sul pacco per certificarne l'avvenuta consegna. Quindi verrà effettuato il passaggio della criptovaluta dal Wallet_{g} della piattaforma al Wallet del venditore. La posizione di ente decentralizzato e neutrale della BlockChain si dimostra in questo caso il perfetto metodo per garantire sicurezza.

1.4 Organizzazione del testo

Nel primo capitolo si descrive brevemente la scelta del progetto e la tecnologia alla base di esso.

Il secondo capitolo entra maggiormente nei dettagli del progetto di stage descrivendo la pianificazione temporale e i gli obiettivi richiesti.

Nel terzo capitolo viene esposta l'analisi dei requisiti con i rispettivi casi d'uso analizzati.

Nel quarto capitolo vengono descritte nel dettaglio le tecnologie utilizzate per la codifica e le diverse fasi di progettazione e implementazione.

Nel settimo e ultimo capitolo viene riportato un consuntivo delle attività svolte e una valutazione dello stage con alcune riflessioni personali.

Capitolo 2

Descrizione dello stage

In questo capitolo viene presentato nel dettaglio il progetto ShopChain insieme agli obiettivi attesi e alla pianificazione delle attività da svolgere.

2.1 Contesto e oggetto del progetto

Il progetto ShopChain consiste nella creazione di una DApp sfruttabile come unico metodo di pagamento per qualsiasi sito di e-commerce. Per raggiungere questo obiettivo, risulta necessario uno studio della BlockChain e del funzionamento delle transazioni in essa. Il cuore pulsante dell'applicazione è lo SmartContract, che definisce tutte le operazioni possibili e memorizza le informazioni utili come:

* indirizzo pubblico degli *account* registrati come venditori o compratori;
* lista degli ordini con le relative informazioni;
* lista delle modifiche su ogni ordine.

La gestione dei pagamenti tramite questo contratto garantisce la massima sicurezza, la somma di denaro infatti viene bloccata in esso una volta effettuato il pagamento da parte del compratore, e viene ricevuta dal venditore solo una volta che il compratore ha confermato la ricezione del pacco. La BlockChain, come visto nella Sezione 1.2, garantisce un *log* pubblico e facilmente consultabile su tutte le transazioni che avvengono in essa. In questo modo è sempre possibile, per le parti coinvolte, verificare i movimenti effettuati dalla somma di denaro. L'oggetto dello stage è l'analisi e l'implementazione delle funzionalità che, nel progetto ShopChain, sono legate alla parte del venditore. Lo scopo dello stage è di familiarizzare e apprendere tecnologie per lo sviluppo *front-end* di applicazioni *web* con il *framework* Angular e l'interazione con la BlockChain, anche vedendo superficialmente tecnologie per lo sviluppo *back-end*.

2.2 Obiettivi

La divisione del lavoro in periodi evidenzia una prima parte dedicata allo studio delle tecnologie necessarie allo sviluppo del progetto, incluse quelle per lo sviluppo del *front-end* e anche approfondimenti sulle principali tecnologie per lo sviluppo del *back-end* negli applicativi *web*. Superata la prima fase, si passa allo studio del progetto di per sé, con conseguente studio della BlockChain, degli SmartContract e di MetaMask$_g$, un

gestore di Wallet *online* che permette all'utente di interagire con le DApp ed effettuare transazioni direttamente dal *browser*. Successivamente, si passa all'implementazione vera e propria e quindi alla creazione delle maschere richieste. Nella Tabella 2.1, vengono riportati gli obiettivi attesi, come definiti in fase di pianificazione dello stage. Il codice identificativo prevede la seguente sintassi:

O[Priorità] - [Id]

dove:

* **O**: identifica che si tratta di un obiettivo.
* **Priorità**: grado di priorità, può assumere i valori:
  - **A**: obbligatorio, necessario;
  - **B**: desiderabile, non strettamente necessario;
  - **C**: opzionale, non necessario.
* **Id**: numero progressivo.

Obiettivo	Descrizione
OA01	Acquisizione competenze sulle tecnologie e gli strumenti utilizzati
OA02	Capacità di raggiungere gli obiettivi richiesti in autonomia seguendo il cronoprogramma
OA03	Portare a termine le implementazioni previste con una percentuale di superamento pari al 80% (equivalente alla maschera di *login* e Profilo Venditore e il suo *service* con le chiamate al *backend*)
OB01	Portare a termine le implementazioni previste con una percentuale di superamento pari al 100% (equivalente alla maschera di *login*, Profilo Venditore e il suo *service* con le chiamate al *backend* e con la presenza di un'interfaccia *responsive* per l'utilizzo da *mobile*)
OC01	Apportare un valore aggiunto al gruppo di lavoro durante le fasi di progettazione delle interfacce

Tabella 2.1: Tabella degli obiettivi

Si evidenzia che l'obiettivo "OB01" prevedeva inizialmente la creazione della maschera del profilo compratore, ma, come indicato nella Sezione 2.3, ha subito una modifica per consentire a un secondo stagista di lavorare al progetto, e comprende ora la realizzazione di un'interfaccia *responsive* per l'utilizzo da *mobile*.

2.3 Pianificazione

Durante la stesura del piano di stage, è stato deciso di dividere il periodo in 8 settimane, ciascuna composta da 40 ore lavorative, per un totale di 320 ore. La pianificazione settimanale è riportata in seguito:

* **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 (40 ore)**
  - Termine integrazioni e collaudo finale.

Con l'azienda, è stato deciso di modificare la pianificazione in modo da permettere ad un secondo stagista di lavorare su questo progetto. Rispetto alla pianificazione iniziale, lo stage è stato maggiormente concentrato sull'implementazione degli aspetti di interesse per il lato della DApp accessibile dal venditore, mentre il secondo stagista ha sviluppato la parte dedicata al compratore. Questa scelta non influisce particolarmente sul lavoro da svolgere, visto che le due parti sono quasi sovrapponibili, con l'unica differenza riguardante le operazioni possibili sugli ordini, che ovviamente differiscono tra compratore e venditore. Per compensare una mole di lavoro in parte ridotta rispetto alla pianificazione iniziale, è stato deciso di sostituire le attività relative al lato compratore con l'attività di implementazione di un'interfaccia *responsive* per l'utilizzo da *mobile*.

Ripartizione delle ore

Nella Tabella 2.2 vengono indicate le ore pianificate per le diverse attività previste.

Durata in ore	**Descrizione dell'attività**
100	Formazione sulle tecnologie
30	Definizione architettura di riferimento e relativa documentazione
10	Stesura documentazione relativa ad analisi e progettazione
30	Analisi del problema e del dominio applicativo
50	Progettazione e implementazione
60	Scrittura dei *service* (su *front-end*) di chiamata al *back-end*
30	Collaudo
7	Stesura documentazione finale
1	Incontro di presentazione della piattaforma con gli *stakeholders*
2	*Live* demo di tutto il lavoro di stage
Totale ore	**320**

Tabella 2.2: Tabella della ripartizione oraria

Si fa notare che, le ore assegnate alla nuova attività di implementazione di un'interfaccia *responsive* per l'utilizzo da *mobile* sostituiscono le ore inizialmente assegnate al lato compratore incluse nella sezione "Progettazione ed implementazione" della tabella.

Capitolo 3

Analisi dei requisiti

In questo capitolo viene presentata l'analisi dei requisiti che evidenzia i diversi casi d'uso dell'applicativo da sviluppare. I diagrammi dei casi d'uso riportati nelle figure permettono una rappresentazione visiva delle possibili interazioni che l'utente può avere con l'applicativo.

3.1 Attori

La Figura 3.1 riporta il diagramma rappresentante gli attori primari. Per comprendere al meglio le loro differenze è necessario introdurre MetaMask [15]. MetaMask è un gestore di Wallet online scaricabile sia come applicazione su *mobile* che come estensione per *browser*, in caso di utilizzo su *desktop*. Permette la creazione di account senza l'utilizzo di indirizzi *mail* garantendo così il principio di anonimia della BlockChain.

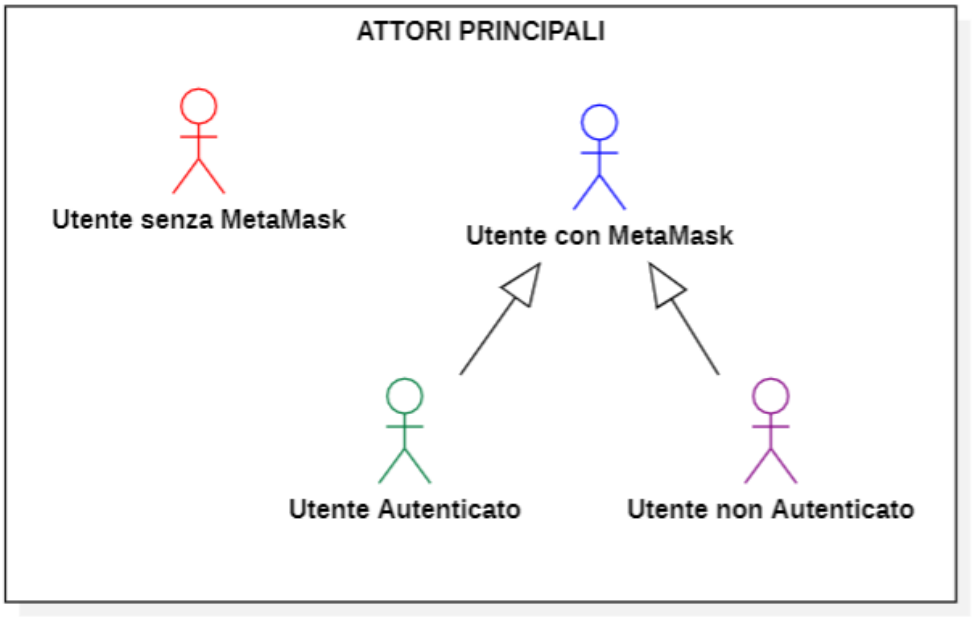


Figura 3.1: Attori Primari

La Figura 3.1 evidenzia la presenza di 4 diversi attori primari. Una prima distinzione individua:

* **Utente senza MetaMask**: non possiede l'estensione di MetaMask nel proprio *browser*;
* **Utente con MetaMask**: possiede l'estensione per il *browser*.

Nel diagramma si evidenzia una generalizzazione che permette di distinguere un utente con MetaMask che risulta registrato o meno come venditore:

* **Utente non Autenticato**: utente che non risulta registrato come venditore;
* **Utente Autenticato**: utente che risulta registrato come venditore.

Come mostrato in Figura 3.2, è inoltre presente un unico attore secondario:

* **MetaMask**: plugin esterno che permette la gestione di Wallet su diversi Network.

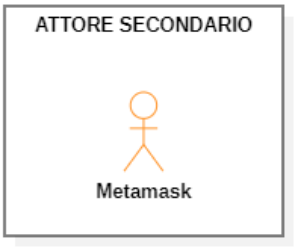


Figura 3.2: Attore Secondario

3.2 Casi d'uso

Per lo studio dei casi d'uso del prodotto, sono stati creati dei diagrammi. I diagrammi dei casi d'uso (in inglese *Use Case Diagram*) sono diagrammi di tipo Unified Modeling Language (UML) dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso [11]. Essendo il progetto finalizzato alla creazione di un applicativo che permetta di svolgere le operazioni in modo semplice, le interazioni da parte dell'utilizzatore sono state opportunamente ridotte allo stretto necessario. Per questo motivo i diagrammi d'uso risultano semplici e in numero ridotto. Nel seguito, si descriveranno i diversi casi d'uso (*Use Case* - (UC)) individuati, e, per ogni caso d'uso, si indicano un codice identificativo e un nome, il diagramma UML (quando significativo), gli attori, le pre condizioni, le post condizioni ed eventuali estensioni.

UC1 - Connessione al wallet

* **Attori primari**: utente con Metamask.
* **Attori secondari**: MetaMask.
* **Pre-condizione**: l'utente non ha ancora connesso il proprio Wallet.
* **Descrizione**: l'utente deve eseguire la connessione al proprio Wallet.

* **Post-condizione**: l'utente ha connesso il proprio Wallet.
* **Scenario principale**:
    1. l'utente accede alla pagina e effettua il collegamento con il Wallet registrato su MetaMask.

UC2 - Registrazione come venditore

* **Attori primari**: utente non Autenticato.
* **Attori secondari**: MetaMask.
* **Pre-condizione**: l'utente non è registrato come venditore.
* **Descrizione**: l'utente vuole registrarsi come venditore sulla piattaforma.
* **Post-condizione**: l'utente risulta registrato come venditore.
* **Scenario principale**:
    1. l'utente non autenticato clicca sul bottone per registrarsi come venditore, si apre quindi il *pop-up* di MetaMask e l'utente conferma la transazione che va a buon fine.
* **Estensioni**:
    - L'utente non conferma la transazione tramite il *pop-up*.
    - L'utente conferma la transazione che però non va a buon fine.

UC3 - Visualizzazione lista Ordini (Figura 3.3)

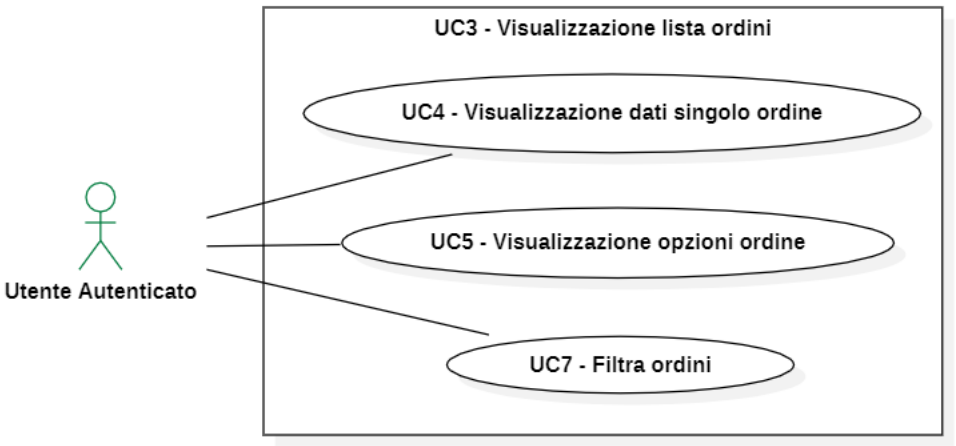


Figura 3.3: Diagramma UC3 - Visualizzazione lista Ordini

* **Attori primari**: utente autenticato.

* **Pre-condizione**: l'utente ha effettuato la connessione al Wallet e si trova nella pagina contente la lista degli ordini.
* **Descrizione**: l'utente visualizza la lista di tutti gli ordini collegati al suo Wallet.
* **Post-condizione**: l'utente ha visualizzato gli ordini a lui registrati.
* **Scenario principale**:
  1. L'utente registrato come venditore accede alla pagina contente la lista dei suoi ordini e consulta le informazioni in essa presenti.
* **Estensioni**:
  - Non sono presenti ordini con l'utente come attore. L'utente viene quindi notificato con un messaggio "*There are no orders registered on this wallet*".

UC4 - Visualizzazione dati singolo ordine (Figura 3.4)

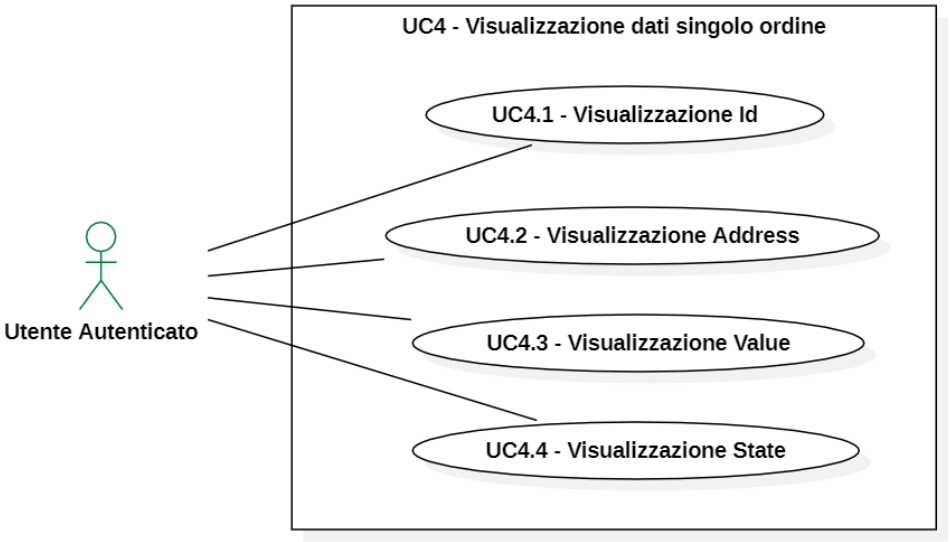


Figura 3.4: Diagramma UC4 - Visualizzazione dati singolo ordine

* **Attori primari**: utente autenticato.
* **Pre-condizione**: l'utente ha effettuato la connessione al Wallet.
* **Descrizione**: l'utente può visualizzare i dati relativi ad un ordine.
* **Post-condizione**: l'utente ha visualizzato i dati relativi ad un ordine.
* **Scenario principale**:
  1. L'utente registrato come venditore consulta la tabella contentente uno o più ordini per visualizzare i loro dati.

UC4.1 - Visualizzazione Id

* **Attori primari**: venditore registrato.
* **Pre-condizione**: il Wallet dell'utente risulta registrato come venditore.
* **Descrizione**: l'utente vuole visualizzare l'*id* di un ordine dalla lista degli ordini.
* **Post-condizione**: l'utente ha visualizzato l'*id* dell'ordine dalla lista.
* **Scenario principale**:
    1. l'utente registrato come venditore accede alla pagina contente la lista dei suoi ordini e consulta la sezione *id* della tabella.

UC4.2 - Visualizzazione Address

* **Attori primari**: venditore registrato.
* **Pre-condizione**: il Wallet dell'utente risulta registrato come venditore.
* **Descrizione**: l'utente vuole visualizzare l'*address* di un ordine dalla lista degli ordini.
* **Post-condizione**: l'utente ha visualizzato l'*address* dell'ordine dalla lista.
* **Scenario principale**:
    1. l'utente registrato come venditore accede alla pagina contente la lista dei suoi ordini e consulta la sezione *address* della tabella.

UC4.3 - Visualizzazione Value

* **Attori primari**: venditore registrato.
* **Pre-condizione**: il Wallet dell'utente risulta registrato come venditore.
* **Descrizione**: l'utente vuole visualizzare il *value* di un ordine dalla lista degli ordini.
* **Post-condizione**: l'utente ha visualizzato il *value* dell'ordine dalla lista.
* **Scenario principale**:
    1. l'utente registrato come venditore accede alla pagina contente la lista dei suoi ordini e consulta la sezione *value* della tabella.

UC4.4 - Visualizzazione State

* **Attori primari**: venditore registrato.
* **Pre-condizione**: il Wallet dell'utente risulta registrato come venditore.
* **Descrizione**: l'utente vuole visualizzare lo *state* di un ordine dalla lista degli ordini.
* **Post-condizione**: l'utente ha visualizzato lo *state* dell'ordine dalla lista.

* **Scenario principale**:

    1. l'utente registrato come venditore accede alla pagina contente la lista dei suoi ordini e consulta la sezione *state* della tabella.

UC5 - Visualizzazione opzioni ordine (Figura 3.5)

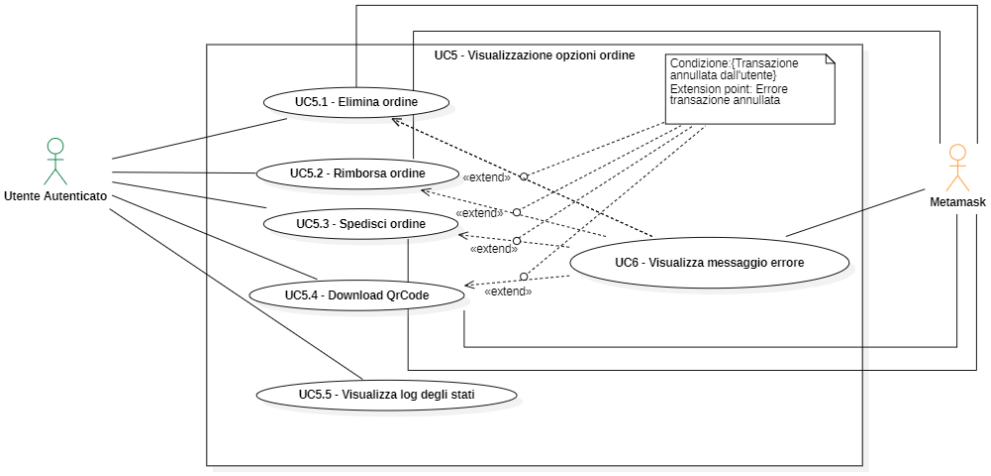


Figura 3.5: Diagramma UC5 - Visualizzazione opzioni ordine

* **Attori primari**: utente autenticato.

* **Attori secondari**: MetaMask.

* **Pre-condizione**: l'utente ha effettuato la connessione al Wallet e ha premuto sul bottone "options" per uno specifico ordine.

* **Descrizione**: l'utente può visualizzare tutte le informazioni su uno specifico ordine, le operazioni possibili e un *log* degli stati.

* **Post-condizione**: l'utente si trova nella pagina relativa ad un singolo ordine.

* **Scenario principale**:

    1. L'utente registrato come venditore si trova nella pagina con la lista degli ordini e preme sul bottone "options" di un ordine, entrando nella pagina relativa ad il singolo ordine.

* **Estensioni**:

    - L'utente non conferma la transazione su MetaMask;
    - L'utente conferma la transazione ma questa non va a buon fine.

UC5.1 - Elimina ordine

* **Attori primari**: utente autenticato.
* **Attori secondari**: MetaMask.
* **Pre-condizione**: l'utente si trova nella pagina di un ordine collecato al suo Wallet e con stato "confirmed" oppure "shipped".
* **Descrizione**: l'utente vuole eliminare uno specifico ordine.
* **Post-condizione**: l'utente ha eliminato l'ordine desiderato.
* **Scenario principale**:
  1. l'utente registrato come venditore preme sul bottone "delete order" e conferma la transazione tramite il *pop-up* di MetaMask .
* **Estensioni**:
  - L'utente non conferma la transazione su MetaMask;
  - L'utente conferma la transazione ma questa non va a buon fine.

UC5.2 - Rimborsa ordine

* **Attori primari**: utente autenticato.
* **Attori secondari**: MetaMask.
* **Pre-condizione**: l'utente registrato si trova nella pagina relativa ad un ordine con stato "refundasked".
* **Descrizione**: l'utente vuole eseguire il rimborso di un ordine.
* **Post-condizione**: l'utente ha effettuato il rimborso.
* **Scenario principale**:
  1. l'utente registrato come venditore preme sul bottone "refund order" e conferma la transazione tramite il *pop-up* di MetaMask.
* **Estensioni**:
  - L'utente non conferma la transazione su MetaMask;
  - L'utente conferma la transazione ma questa non va a buon fine.

UC5.3 - Spedisci ordine

* **Attori primari**: venditore registrato.
* **Attori secondari**: MetaMask.
* **Pre-condizione**: l'utente registrato si trova nella pagina relativa ad un ordine con stato "created".
* **Descrizione**: l'utente vuole segnalare l'avvenuta spedizione dell'ordine.
* **Post-condizione**: l'utente ha impostato correttamente il nuovo stato.

* **Scenario principale**:
    1. l'utente registrato come venditore preme sul bottone "ship order" e conferma la transazione tramite il *pop-up* di MetaMask.
* **Estensioni**:
    - L'utente non conferma la transazione su MetaMask;
    - L'utente conferma la transazione ma questa non va a buon fine per un errore nella BlockChain.

UC5.4 - Download QRCode

* **Attori primari**: venditore registrato.
* **Attori secondari**: MetaMask.
* **Pre-condizione**: l'utente registrato si trova nella pagina relativa ad un ordine con stato "Created".
* **Descrizione**: l'utente vuole scaricare il *QrCode* da attaccare al pacco da spedire.
* **Post-condizione**: l'utente ha scaricato correttamente il *QrCode* come immagine.
* **Scenario principale**:
    1. l'utente registrato come venditore preme sul bottone *download*; *QrCode* e scarica l'immagine.

UC5.5 - Visualizza log degli stati

* **Attori primari**: venditore registrato.
* **Pre-condizione**: l'utente registrato si trova nella pagina relativa ad uno specifico ordine.
* **Descrizione**: l'utente vuole visualizza una lista con riportato un *log* per ogni cambiamento di stato dell'ordine con relativa data e ora.
* **Post-condizione**: l'utente ha visualizzato correttamente le informazioni di interesse.
* **Scenario principale**:
    1. l'utente registrato come venditore visualizza la lista contente il *log* dei cambiamenti di stato dell'ordine.

UC6 - Visualizza messaggio di errore

* **Attori primari**: venditore registrato.
* **Pre-condizione**: l'utente ha avviato un'operazione e si è aperto il *pop-up* di MetaMask.
* **Descrizione**: l'utente preme sul bottone per eseguire un'operazione ma questa non va a buon fine e viene visualizzato un errore.

* **Post-condizione**: l'utente visualizza un messaggio che indica cosa ha impedito l'esecuzione dell'operazione.
* **Scenario principale**: I casi possibili sono riconducibili a due scenari principali:
  1. l'utente annulla la transazione tramite il *pop-up* di MetaMask.
     - viene visualizzato a schermo l'errore: "*Transaction ended with error: MetaMask Tx Signature: User denied transaction signature.*"
  2. l'utente conferma la transazione ma questa non va a buon fine.
     - viene visualizzato a schermo l'errore: "*Transaction ended with error: MetaMask Tx Signature: Failed Transaction.*", in questo caso l'utente può accedere alla pagina di Snowtrace del proprio account e visualizzare i dettagli della transazione.

UC7 - Filtra Ordini (Figura 3.6)

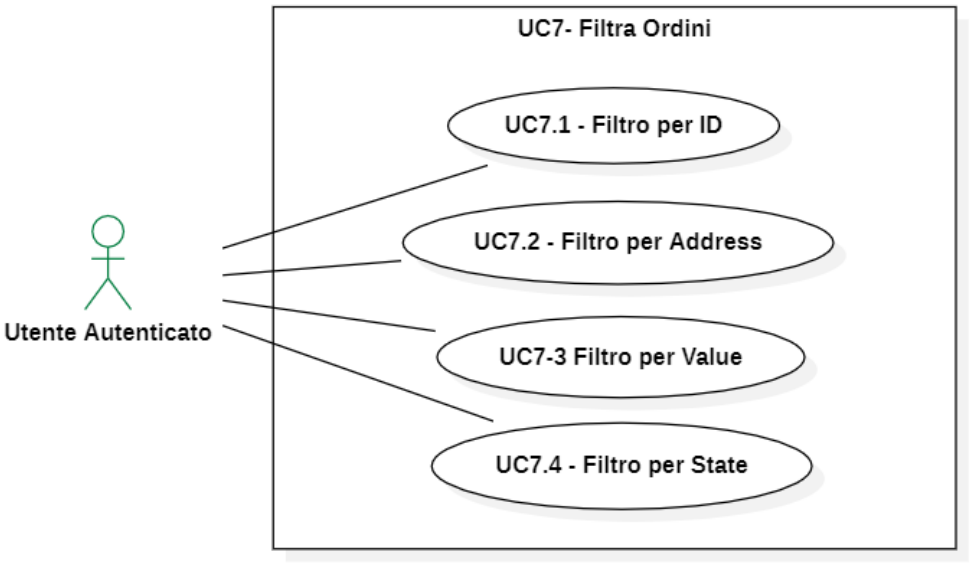


Figura 3.6: Diagramma UC7 - Filtra ordini

* **Attori primari**: venditore registrato.
* **Pre-condizione**: sono presenti ordini registrati al venditore.
* **Descrizione**: l'utente sceglie uno dei filtri disponibili per ricercare in modo più efficace un ordine nella lista.
* **Post-condizione**: l'utente ha filtrato la lista degli ordini come voleva.
* **Scenario principale**:
  1. l'utente filtra gli ordini per *id*.
  2. l'utente filtra gli ordini per *Address*.

3. l'utente filtra gli ordini per *Value*.
4. l'utente filtra gli ordini per *State*.

UC7.1 - Filtro per Id

* **Attori primari**: venditore registrato.
* **Pre-condizione**: il Wallet dell'utente risulta registrato come venditore.
* **Descrizione**: l'utente filtra la lista per visualizzare solo ordini con un certo *id*.
* **Post-condizione**: l'utente ha filtrato la lista per un determinato *id*.
* **Scenario principale**:
    1. l'utente registrato come venditore filtra la lista degli ordini per un determinato *id*.

UC7.2 - Filtro per Address

* **Attori primari**: venditore registrato.
* **Pre-condizione**: sono presenti ordini registrati all'utente.
* **Descrizione**: l'utente filtra la lista per visualizzare solo ordini con un certo *address*.
* **Post-condizione**: l'utente ha filtrato la lista per un determinato *address*.
* **Scenario principale**:
    1. l'utente registrato come venditore filtra la lista degli ordini per un determinato *address*.

UC7.3 - Filtro per Value

* **Attori primari**: venditore registrato.
* **Pre-condizione**: sono presenti ordini registrati all'utente.
* **Descrizione**: l'utente filtra la lista per visualizzare solo ordini con un certo *value*.
* **Post-condizione**: l'utente ha filtrato la lista per un determinato *id*.
* **Scenario principale**:
    1. l'utente registrato come venditore filtra la lista degli ordini per un determinato *value*.

UC7.4 - Filtro per State

- **Attori primari**: venditore registrato.
- **Pre-condizione**: sono presenti ordini registrati all'utente.
- **Descrizione**: l'utente filtra la lista per visualizzare solo ordini con un certo stato.
- **Post-condizione**: l'utente ha filtrato la lista per un determinato stato.
- **Scenario principale**:
  1. l'utente registrato come venditore filtra la lista degli ordini per un determinato stato.

3.3 Tracciamento dei requisiti

Ogni requisito viene definito nel seguente modo:

- Codice identificativo.
- Descrizione.
- Caso d'uso.

Il codice identificativo prevede la seguente sintassi:

R [Tipologia] - [Identificativo]

dove:

- **R**: identifica che si tratta di un requisito.
- **Priorità**: grado di priorità, può assumere i valori:
  - **A**: obbligatorio, necessario;
  - **B**: desiderabile, non strettamente necessario;
  - **C**: opzionale, non necessario.
- **Tipologia**: tipo di requisito, può assumere i valori:
  - **F**: funzionale;
  - **P**: prestazionale;
  - **Q**: qualitativo;
  - **V**: vincolo.
- **ID**: numero progressivo.

Nelle Tabelle 3.1 e 3.2 sono riassunti i requisiti e il loro tracciamento con gli use case delineati in fase di analisi e riportati nella Sezione 3.2. Si fa notare che tutti i vincoli individuati sono funzionali o di vincolo e di priorità "A".

Tabella 3.1: Tabella del tracciamento dei requisti funzionali

Requisito	Descrizione	Use Case
RAF-1	Il sistema deve permettere la registrazione di un nuovo utente tramite MetaMask	UC1
RAF-2	Il sistema deve permettere la registrazione come venditore	UC2
RAF-3	Il sistema deve permettere la visualizzazione della lista degli ordini	UC3
RAF-4	Il sistema deve permettere di filtrare la lista degli ordini	UC7
RAF-5	Il sistema deve permettere di filtrare la lista degli ordini	UC7
RAF-6	Il sistema deve permettere di filtrare la lista degli ordini per ID	UC7.1
RAF-7	Il sistema deve permettere di filtrare la lista degli ordini per Address	UC7.2
RAF-8	Il sistema deve permettere di filtrare la lista degli ordini per Value	UC7.3
RAF-9	Il sistema deve permettere di filtrare la lista degli ordini per State	UC7.4
RAF-10	Il sistema deve permettere di visualizzare le opzioni su un ordine	UC5
RAF-11	Il sistema deve permettere di visualizzare i dati dei singoli ordini	UC4
RAF-12	Il sistema deve permettere la visualizzazione degli ID	UC4.1
RAF-13	Il sistema deve permettere la visualizzazione degli Address	UC4.2
RAF-14	Il sistema deve permettere la visualizzazione dei Value	UC4.3
RAF-15	Il sistema deve permettere la visualizzazione degli State	UC4.4
RAF-16	Il sistema deve permettere di eliminare un ordine	UC5.1
RAF-17	Il sistema deve permettere di effettuare il rimborso verso un compratore	UC.2
RAF-18	Il sistema deve permettere di segnalare l'iniziata spedizione di un ordine	UC5.3
RAF-19	Il sistema deve permettere il "*download QrCode*" contenente *address* e id	UC5.4
RAF-20	Il sistema deve permettere la visualizzazione dei *log* degli stati per ogni ordine	UC5.5
RAF-21	Il sistema deve permettere la visualizzazione di un messaggio in caso di errore	UC5.5

Tabella 3.2: Tabella del tracciamento dei requisiti di vincolo

Requisito	Descrizione
RAV-1	Lo sviluppo del front-end deve avvenire tramite il *framework* Angular
RAV-2	Il sistema deve interagire con MetaMask
RAV-3	ShopChain dovrà utilizzare la BlockChain C-Chain$_g$ del Network Avalanche$_g$

Capitolo 4

Progettazione e codifica

In questo capitolo vengono inizialmente descritte le tecnologie e gli strumenti utilizzati, successivamente si espongono le fasi di progettazione e codifica relativi all'applicativo sviluppato.

4.1 Tecnologie e strumenti

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

JavaScript e TypeScript

JavaScript è un linguaggio di programmazione multi paradigma orientato agli eventi, comunemente utilizzato nella programmazione *Web* lato *client*, per la creazione di effetti dinamici interattivi. L'assenza della tipizzazione dei dati in questo linguaggio viene risolta da Typescript. Typescript è un linguaggio che estende la struttura esistente di *JavaScript* aggiungendo o rendendo più flessibili e potenti varie sue caratteristiche come la tipizzazione delle variabili e la creazione di interfacce e classi [26]. I file con estensione ".ts" sono scritti con il linguaggio Typescript.

Framework 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 e di avere un elevato riutilizzo del codice. Permette inoltre un'elevata manutenibilità in quanto ogni componente è adibito ad un'unica funzione [13].

Visual Studio Code

Visual Studio Code è un editor di codice sorgente, particolarmente comodo nell'utilizzo in quanto include il supporto per *debugging*, un controllo per *Git* integrato, *syntax highlighting*, *IntelliSense*, *snippet* e *refactoring* del codice [27].

Solidity

Solidity è un linguaggio di programmazione orientato agli oggetti per l'implementazione di contratti intelligenti su varie piattaforme BlockChain, in particolare Ethereum. Lo SmartContract utilizzato in questo progetto è scritto in questo linguaggio [25].

Hardhat

Hardhat è un ambiente di sviluppo per compilare, distribuire, testare ed eseguire il debug del software Ethereum. Fornisce un Network locale basato su Ethereum studiato appositamente per lo sviluppo. Le sue funzionalità sono incentrate intorno al *debugging* e *deploying* di SmartContract scritti in *Solidity*. In questo progetto è stato utilizzato appunto per *debugging* e *deploy* dello SmartContract [24].

Remix

Remix è un ambiente per *testing* e *deploy* di SmartContract simile ad *Hardhat* ma interamente *online* [18]. È stato utilizzato utilizzato nel progetto per testare le funzioni dello SmartContract.

4.2 Progettazione

Si presenta ora l'architettura del *framework* Angular, dell'applicativo sviluppato e i *design pattern* utilizzati.

4.2.1 Architettura di Angular

Come mostrato in Figura 4.1, in Angular, un modulo è una classe a cui applichiamo il particolare decoratore (vedi Sezione 4.2.3) *"@NgModule"*, che permette di organizzare la nostra applicazione strutturandola in unità che contengono entità aventi una certa relazione fra loro. In questo modo possiamo raggruppare all'interno di un singolo contenitore componenti e direttive che sono in qualche modo collegati fra loro perché contribuiscono alla realizzazione di una certa funzionalità comune. Ogni modulo possiede un *component* al quale viene associato un *template*. Un *component*, in Angular, contiene la logica di interazione dati e utente che definisce l'aspetto ed il comportamento della vista, il *template* invece rappresenta la vista stessa contenente il codice HTML. Le applicazioni Angular possono avere diversi componenti. Ogni componente viene dedicato a gestire una singola porzione dell'interfaccia utente. Questi *component* lavorano insieme per fornire la UI (User Interface) dell'applicazione. La Figura 4.1 illustra anche il concetto di *Injector* e *Service*. Un *Service* è un oggetto che viene istanziato una sola volta durante il ciclo di vita dell'applicazione e contiene metodi che mantengono informazioni e le rendono sempre disponibili. Una classe *Injector* (generalmente un servizio), contiene il decoratore *"@Injector"* che permette la sua iniezione in altre classi, come dettato dal *Dependecy Injection Pattern* (vedi Sezione 4.2.3).

4.2.2 Architettura dell'applicazione sviluppata

L'architettura dell'applicazione è stata progettata seguendo le *best practise* già consolidate nell'ambiente di Angular [1]. L'utilizzo di questo *framework* ha semplificato la

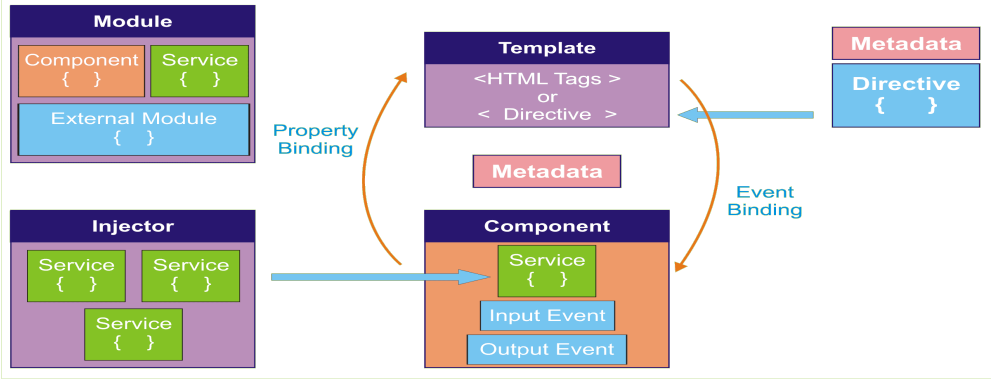


Figura 4.1: Schema dell'Architettura di Angular [3]

creazione di un codice altamente modulare e manutenibile, la sua architettura infatti guida lo sviluppatore nella creazione di un programma che rispetti tali caratteristiche. Come si può vedere nella Figura 4.2, tutti i *component* sono situati in apposite cartelle tutte contenute nella cartella *"components"*, lo stesso vale anche per il *service*. Per lo spostamento tra le pagine, è stato scelto di sfruttare il *routing*, una tecnica che permette di modificare quello che l'utente vede senza cambiare pagina, ma mostrando o nascondeno parti che corrispondono a determinati *component* per permettere lo spostamento rapido tra diverse *view*. Nel file "orders.ts" viene definita una interfaccia per gli ordini con queste caratteristiche:

* id: number;
* buyerAddress: string;
* sellerAddress: string;
* amount: string;
* state: number.

L'interfaccia viene usata nei *component*: "order-info" e "order-list". Una seconda interfaccia viene definita nel "Log.ts" per il *log* delle modifiche sull'ordine:

* state: string;
* timestamp: string.

L'interfaccia viene poi sfruttata dal *component* "order-log".

4.2.3 Design Pattern utilizzati

MVC Pattern

Il classico MVC (Model View Controller) consiste in un'architettura costituita da tre parti distinte [16]:

* Model: interfacce e classi senza logica;
* View: user interface;

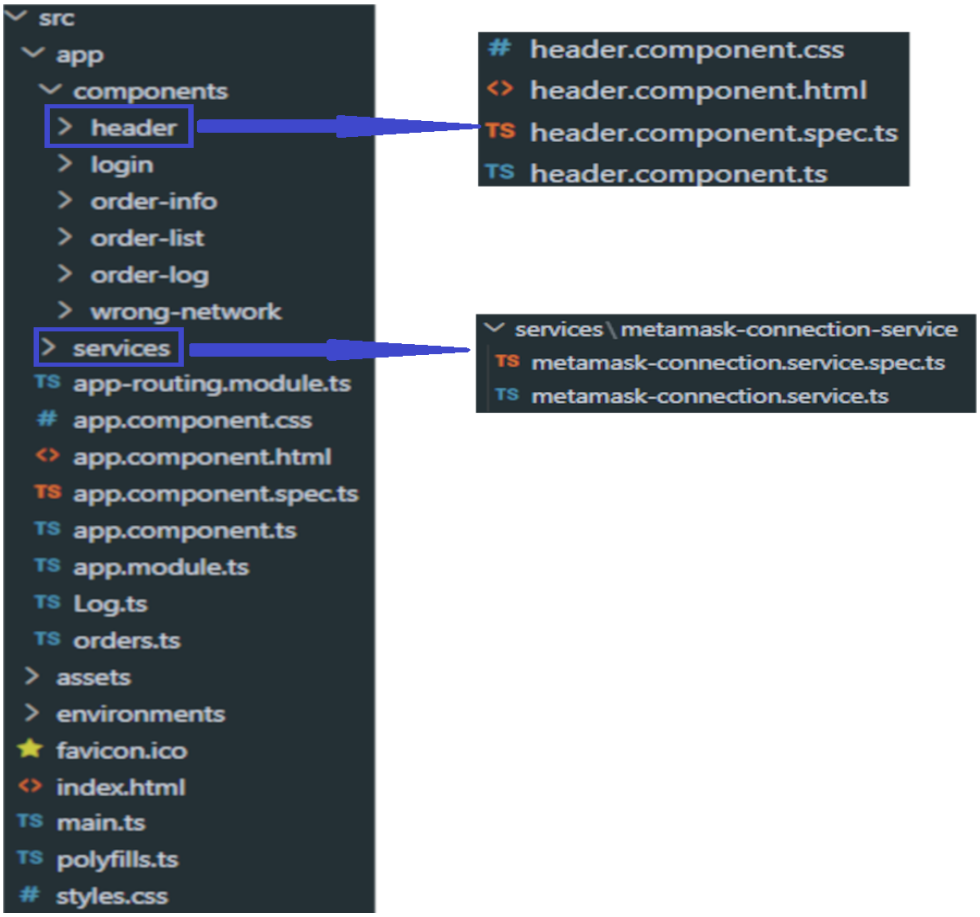


Figura 4.2: Struttura dell'applicativo sviluppato

* Controller: uniscono i due layer precedenti.

Angular intrinsecamente sfrutta un *design pattern* leggermente variato e semplificato che si accosta ad un MVVM (Model View ViewModel) [17] che, come mostrato in Figura 4.3, permette un *data binding* a due direzioni tra "View" e "ViewModel".

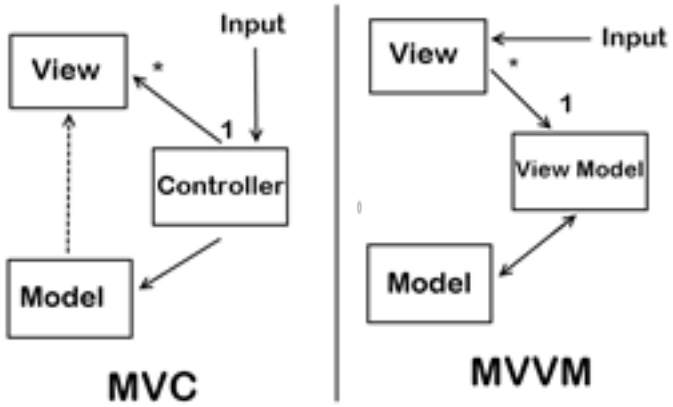


Figura 4.3: Struttura MVC e MVVM [2]

Dependency Injection

DI (Dependecy Injection) è un *design pattern* della programmazione orientata agli oggetti il cui scopo è quello di semplificare lo sviluppo e migliorare la testabilità di *software* di grandi dimensioni [10]. In particolare, la DI permette di delegare parte delle funzionalità del *software* ad un servizio ("MetamaskConnectionService"), anziché implementarle direttamente nella UI, e in particolar modo, nel caso di Angular, all'interno di componenti/direttive/*pipes*/ecc. Facendo ciò si migliora l'efficienza e la modularità. Il DI usato da Angular è di tipo *constructor*, questo prevede che in ogni componente vengano dichiarati nel costruttore i servizi dei quali ha bisogno e, in fase di inizializzazione, tali dipendenze gli vengano fornite dall'esterno.

Decorator

Il *design pattern* Decorator permette l'aggiunta di nuove funzionalità ad oggetti o classi già esistenzi, senza modificare il codice sorgente. Questo è reso possibile da una classe "decoratore" alla quale viene passato come parametro l'oggetto originale [9]. Questo *pattern* differisce dalla classica ereditarietà, che non permette la modifica, aggiunta o rimozione delle funzionalità a *run-time*.

4.3 Codifica

Si presentano ora nello specifico le diverse classi create per le maschere da sviluppare.

4.3.1 Service

Come anticipato nella Sezione 4.2.1, un *service* in Angular è semplicemente una classe Typescript con il decoratore "*@Injectible*", il quale comunica ad Angular che tale classe

può essere iniettata in altre che devono usufruire delle funzioni da essa fornite. Come mostrato nella Figura 4.4, un *service* può essere iniettato in molteplici *component*.

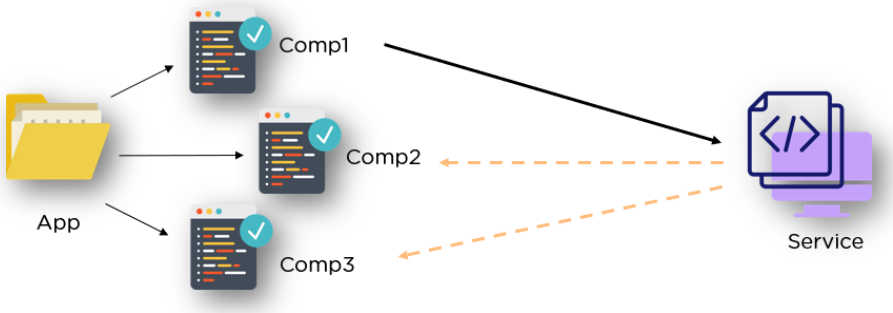


Figura 4.4: Service in Angular [14].

Nel progetto svolto è presente un unico *service*, "MetamaskConnectionService".

MetamaskConnectionService

Questo *service* contiene tutte le funzioni che permettono l'interazione con lo SmartContract e con MetaMask. Nello specifico, la classe sfrutta la libreria Ethers$_g$ [12], per connettersi alla BlockChain e riuscire a richiamare le funzioni dello SmartContract per eseguire le operazioni sugli ordini o per recuperare informazioni come la lista degli ordini e il *log* degli eventi. Per quanto riguarda MetaMask, il *service* permette lo scambio di informazioni quali:

* Presenza del *plugin* nel *browser* dell'utente;
* Connessione alla Network corretta;
* Bilancio del Wallet;
* Stato delle transazioni.

Questa classe è stata la più complessa da sviluppare proprio perché gestisce tutte le interazioni con lo SmartContract e con la BlockChain. Il problema principale è causato dalle funzioni da utilizzare per la connessione con MetaMask e con il Network, che vengono aggiornate e modificate molto frequentemente, rendendo la documentazione disponibile *online* spesso obsoleta e di scarsa utilità.

4.3.2 Components

Nella cartella "Components" sono contenuti tutti i componenti dell'applicativo, ognuno di essi svolge un particolare ruolo nella la formazione del sistema completo. Analizziamo ora nel dettaglio ogni *component* presente.

Header

Il *component* "Header" dà origine all'*header* presente in tutto l'applicativo.

Figura 4.5: Header Component

Come si vede in Figura 4.5, è composto dal logo e dal nome dell'applicazione sul lato sinistro. Sul lato destro mostra all'utente l'indirizzo del Wallet connesso e anche le informazioni riguardo al bilancio (nella criptovaluta $AVAX_g$). Per mostrare il bilancio del Wallet è richiesta l'interazione con lo SmartContract, la classe chiama quindi il service "MetamaskConnectionService" per comunicare con esso.

Login

Il *component* "Login" si occupa delle funzionalità di accesso al sistema, permette l'accesso alla lista degli ordini, l'accesso al proprio Wallet e la registrazione come venditore per gli utenti non ancora autenticati (vedi Figura 4.6).

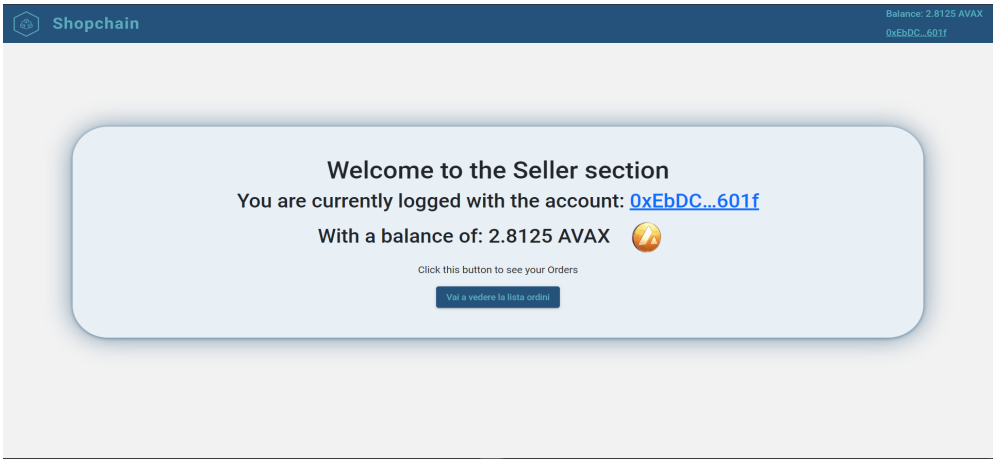


Figura 4.6: Schermata pagina di login

Contiene due funzioni principali:

* *isRegistered()*: si occupa di verificare se il Wallet dell'utente risulta nella lista dei venditori, per farlo chiama l'omonima funzione del *service*;

* *registerAsSeller()*: si occupa di registrare un Wallet nella lista dei venditori, per farlo chiama l'omonima funzione del *service*.

OrderList

Questo *component* si occupa di gestire la lista degli ordini registrati ad un determinato venditore.

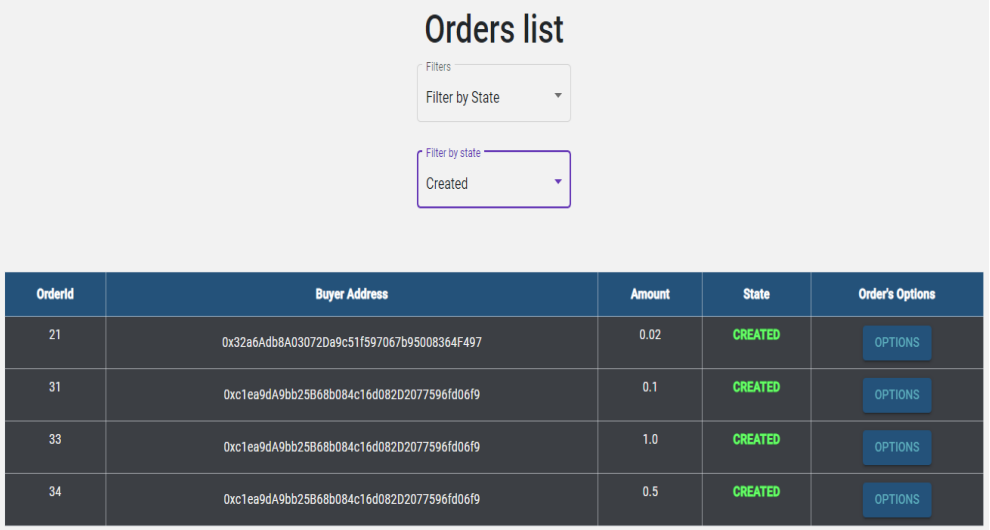


Figura 4.7: Lista degli ordini

Come mostrato in Figura 4.7, "OrderList" mette a disposizione le funzioni per filtrare gli ordini in base a:

* **Id** dell'ordine;
* **Address** del compratore;
* **Value** della transazione;
* **State** attuale dell'ordine.

Dalla lista degli ordini è anche possibile accedere alla pagina "orderinfo", relativa ad ogni singolo ordine, tramite l'apposito bottone "options".

OrderInfo

Questo è il *component* che si occupa della pagina riguardante le opzioni su un singolo ordine.

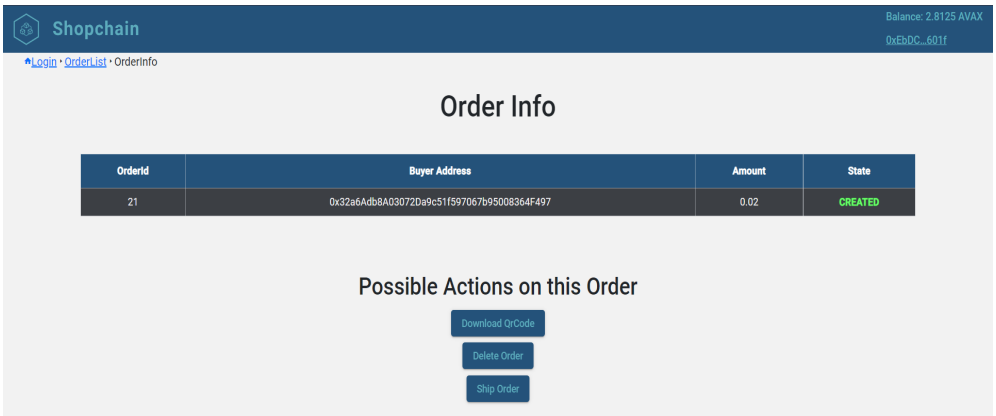


Figura 4.8: Schermata pagina OrderInfo

Come mostrato in Figura 4.8, oltre ad esporre le informazioni dell'ordine, mostra all'utente le operazioni possibili sull'ordine, che differiscono in base allo stato dello stesso. Un ordine nello stato "created" permette il *download* del *QrCode*, la segnalazione di avvenuta spedizione e la cancellazione dell'ordine stesso. Lo stato "shipped" consente al venditore la sola operazioni di cancellazione. Gli stati "confirmed","deleted" e "refunded" non consentono alcuna operazione aggiuntiva. Lo stato "refundasked" permette al venditore di risarcire il compratore. Questo *component* si occupa anche di gestire la schermata che appare durante una transazione con MetaMask richiesta per una delle operazioni sull'ordine.

All'interno di "OrderInfo", viene anche richiamato il *component* "OrderLog".

OrderLog

È il *component* che gestisce la lista delle modifiche di stato avvenute sull'ordine. Come

State	Data
created	8/6/2022, 11:05:08
refundAsked	10/6/2022, 10:29:49
refunded	10/6/2022, 10:29:49

Figura 4.9: Tabella modifiche avvenute sull'ordine

mostrato in Figura 4.9, "OrderLog" mostra all'utente una tabella che riporta lo stato e la data (giorno e orario) nella quale l'ordine è stato aggiornato a quello stato.

WrongNetwork

"WrongNetwork" si occupa di gestire il caso in cui l'utente non sia connesso al Network corretto.

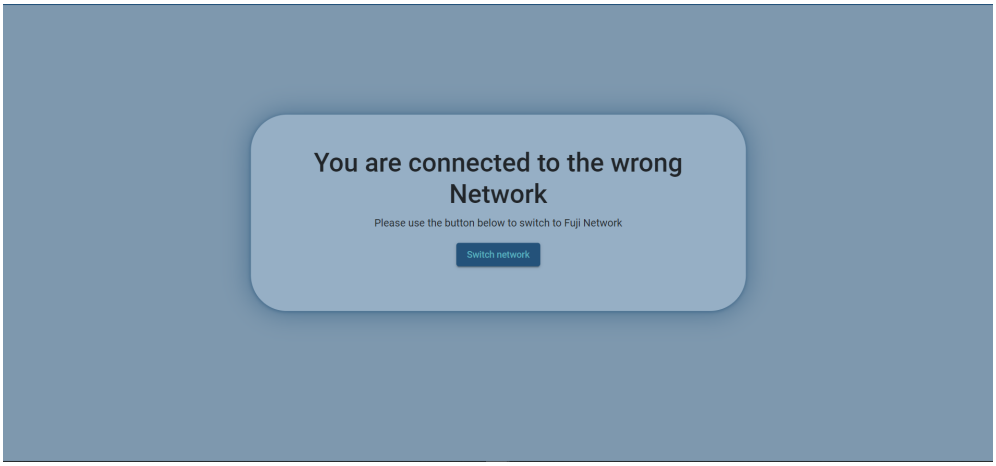


Figura 4.10: Schermata in caso di Network sbagliata

Come mostrato in Figura 4.10, è presente il bottone "switch network" che apre un *pop-up* di MetaMask che aiuta l'utente nel cambiare Network.

4.4 Accessibilità

In questa sezione vengono trattati gli aspetti di accessibilità che sono stati presi in considerazione durante lo sviluppo. Si evidenzia il fatto che l'inevitabile utilizzo di MetaMask rende molto difficile la navigazione per chi sfrutta solamente tecnologie assistive come gli screen reader$_g$. MetaMask richiede infatti l'interazione con un *pop-up* che non viene facilmente evidenziato e navigato da questi strumenti. È stato comunque ricercato un buon livello di accessibilità per le porzioni dell'applicativo nelle quali fosse possibile.

4.4.1 Colori

I colori sono stati scelti seguendo le *palette* di Angular *material*, così da garantire un buon rapporto per passare il *test* WCAG AA (Web Content Accessibility Guidelines) [22], che richiede un rapporto di contrasti di almeno 4.5:1 per il testo normale e 3:1 per il testo in grassetto.

La Tabella 4.1 mostra i colori del testo, del corrispondente sfondo e il rapporto di contrasto tra i due.

Colore Testo	Colore Sfondo	Contrasto
#FFFFFF	#3C3F44	10.57:1
#AACACA	#24527A	4.68:1
#66FF66	#3C3F44	8.08:1
#00CC00	#3C3F44	4.85:1
#F2C846	#3C3F44	6.6:1

Tabella 4.1: Tabella rapporto contrasti

4.4.2 Attributi HTML

Per rendere il sito più accessibile, sono stati utilizzati i seguenti attributi *HTML*:

* **scope**, utilizzato per facilitare la lettura delle tabelle;
* **lang**, utilizzato per indicare allo screen reader in che lingua leggere una parola o frase diversa dall'italiano;
* **role**, utilizzato per definire la funzione di alcuni *tag*;
* **aria-hidden**, utilizzato per nascondere icone inutili agli screen reader perchè già presenti descrizioni testuali;
* **aria-label**: utilizzato per definire una descrizione testuale per bottoni rappresentati solo da icone.

4.4.3 Altri elementi di accessibilità

Per aumentare ulteriormente i livelli di accessibilità, si sono evitati testi scorrevoli, lampeggianti, barrati ed in generale *font* troppo elaborati, in modo da agevolare la lettura.

Capitolo 5

Conclusioni

In questo capitolo si dà un consuntivo delle attività, viene analizzato il raggiungimento degli obiettivi prefissati ad inizio stage e vengono esposte delle riflessioni sul percorso di stage.

5.1 Consuntivo delle attività

In questa sezione si analizza, settimana per settimana, la conformità delle tempistiche preventivate (2.3) con quanto realmente accaduto nel corso del tirocinio.

* **Settimana 1**: settimana svolta rispettando a pieno la pianificazione.
* **Settimana 2**: settimana svolta rispettando la pianificazione. Si è inoltre svolto un colloquio per verificare il corretto apprendimento di quanto studiato.
* **Settimana 3**: settimana svolta rispettando completamente la pianificazione.
* **Settimana 4**: lo studio approfondito di Angular ha richiesto anche due giorni della settimana successiva, comportando un leggero ritardo rispetto alla pianificazione.
* **Settimana 5**: lo SmartContract ha richiesto delle ore di lavoro aggiuntive a causa di qualche errore al suo interno. Il ritardo accumulato è stato però recuperato nel periodo dedicato alla creazione della maschera di accesso.
* **Settimana 6**: la progettazione e l'implementazione della maschera per il venditore sono state svolte nel tempo previsto, si nota che la quasi totalità delle ore è stata dedicata alla scrittura del *service* per l'interazione con lo SmartContract.
* **Settimana 7**: durante la settima settimana è stata implementata la versione *mobile*.
* **Settimana 8**: nell'ultima settimana è stata rispettata la pianificazione e si è svolto un incontro finale con il *tutor* aziendale nel quale è stato presentato il prodotto ottenuto.

Nella Tabella 5.1 vengono esposte le ore previste per ogni attività in relazione con le ore effettivamente dedicate ad ognuna di esse.

Descrizione dell'attività	Ore previste	Ore effettive
Formazione sulle tecnologie	100	100
Definizione architettura di riferimento e relativa documentazione	30	30
Stesura documentazione relativa ad analisi e progettazione	10	10
Analisi del problema e del dominio applicativo	30	30
Progettazione e implementazione	50	40
Scrittura dei service (su front-end) di chiamata al back-end	60	70
Collaudo	30	30
Stesura documentazione finale	7	7
Incontro di presentazione della piattaforma con gli stakeholders	1	1
Live demo di tutto il lavoro di stage	2	2
Totale ore	320	320

Tabella 5.1: Tabella di confronto tra le ore previste e le ore effettive per ogni attività

Come indicato dalla tabella, le ore totali coincidono con quelle previste, ci sono però state delle differenza rispetto a quanto pianificato. L'attività di scrittura dei *service* ha richiesto più tempo del previsto a causa di alcuni errori presenti nelle funzioni dello SmartContract. Le ore di ritardo sono però state recuperate nell'attività di implementazione delle maschere.

Il periodo iniziale di studio si è rivelato importante per apprendere al meglio il *framework* Angular che, prima di questo momento, non avevo mai utilizzato. Lo studio approfondito del progetto è stato essenziale per quanto riguarda l'apprendimento dello sviluppo di applicativi che interagiscono con la BlockChain. Una conoscenza di base come quella che possedevo ad inizio percorso si è infatti rivelata quasi inutile in quanto limitata ad un punto di vista da utente finale. In generale, il periodo di stage ha progredito in modo lineare e soddisfacente, e il prodotto finito è risultato all'altezza delle aspettative dell'azienda che si è dimostrata soddisfatta del lavoro svolto.

5.2 Raggiungimento degli obiettivi

Gli obiettivi prefissati a inizio stage risultano totalmente raggiunti. Come mostrato nella Tabella 5.2, tutti gli obiettivi sono stati raggiunti.

Obiettivo	Descrizione	Soddisfatto
O01	Acquisizione competenze sulle tecnologie e gli strumenti utilizzati	Si
O02	Capacità di raggiungere gli obiettivi richiesti in autonomia seguendo il cronoprogramma	Si
O03	Portare a termine le implementazioni previste con una percentuale di superamento pari al 80% (equivalente alla maschera di login e Profilo Venditore e il suo *service* con le chiamate al *backend*)	Si
D01	Portare a termine le implementazioni previste con una percentuale di superamento pari al 100% (equivalente alla maschera di login, Profilo Venditore e il suo *service* con le chiamate al *backend* e con la presenza di un'interfaccia *responsive* per l'utilizzo da *mobile*)	Si
F01	Apportare un valore aggiunto al gruppo di lavoro durante le fasi di progettazione delle interfacce	Si

Tabella 5.2: Tabella degli obiettivi

5.3 Conoscenze acquisite

Il percorso di stage mi ha permesso di fare notevoli progressi sulle conoscenze relative alla tecnologia della BlockChain e anche sullo sviluppo *front-end* di applicativi *web*. Il periodo di studio iniziale mi ha permesso di apprendere Angular nel migliore dei modi e quindi di non riscontrare particolari problemi nella fase di sviluppo. Qualche difficoltà incontrata invece nelle prime interazioni con lo SmartContract ha inizialmente rallentato il procedere dello sviluppo, ma, il lavoro svolto per la risoluzione di questi problemi si è

rivelato molto utile per conoscere ancora meglio la tecnologia ed apprendere numerose nozioni che altrimenti non avrei potuto conoscere.

5.4 Valutazione personale

Lo stage è stata un'esperienza molto positiva, che mi ha sicuramente pemesso di comprendere meglio il mondo del lavoro nel campo informatico. Poter lavorare finalmente su un progetto scelto da me e che tratta argomenti di mio maggiore interesse, mi ha permesso di apprezzare ancora di più questa attività. Ho ritenuto particolarmente stimolante il dover approfondire e studiare argomenti solo tramite documentazione online, diversamente da come avviene tipicamente nel percorso di studi. Ho potuto finalmente capire cosa significa lavorare in una realtà aziendale nel campo dell'Informatica, grazie anche all'interazione con, non solo altri stagisti, ma anche numerosi dipendenti dell'azienda Synclab. In conclusione credo questo percorso di stage sia un fondamentale componente del mio percorso di studio.

Glossario

Avalanche "Avalanche è un Network complesso, del quale Avax è il token di riferimento, che permette di accedere a diversi tipi di funzionalità, dagli SmartContract alla *tokenizzazione* degli *asset* finanziari esterni a questa BlockChain" [4]. 20, 35

Avalanche Contract Chain La *Avalanche Contract Chain* è la BlockChain di default su Avalanche che permette la creazione di SmartContract compatibili con Ethereum. 20, 36

AVAX Moneta di riferimento nel network di Avalanche. 27

BlockChain Un insieme di tecnologie, in cui il registro è strutturato come una catena di blocchi contenenti le transazioni e il consenso è distribuito su tutti i nodi della rete. Tutti i nodi possono partecipare al processo di validazione delle transazioni da includere nel registro. iii, 1, 2, 5, 9, 16, 20, 22, 26, 32, 33, 35, 36

Criptovaluta "Una è una valuta virtuale che, secondo la definizione di Banca d'Italia, costituisce una rappresentazione digitale di valore ed è utilizzata come mezzo di scambio o detenuta a scopo di investimento. Le critpovalute possono essere trasferite, conservate o negoziate elettronicamente" [7]. iii, 2, 27, 35, 36

DApp È un'applicazione decentralizzata, cioè un'applicazione che può funzionare in modo autonomo, in genere attraverso l'uso di SmartContract, che viene eseguita su un sistema informatico decentralizzato, BlockChain o altro sistema di contabilità distribuita. 2, 5, 6, 8

E-commerce Letteralmente "*electronic commerce*", un *e-commerce* è appunto una pratica commerciale che sfrutta *internet* per l'interazione tra venditore e compratore. 2, 5

Ethereum Ethereum è la BlockChain con il più grande numero di applicazioni e utilizzi attualmente esistente, la criptovaluta a esso legata, Ether, è seconda in capitalizzazione dietro ai Bitcoin. 22, 35, 36

Ethers.js Ethers.js è una libreria che permette di interagire con la BlockChain di Ethereum e il suo ecosistema [12]. 26

MetaMask MetaMask è un gestore di Wallet online, scaricabile come *add-on* su *browser web* oppure come applicazione su *smartphone*. Nel contesto di questo documento e del progetto di stage, ci si riferisce solo al *plug-in* per *browser*. 5, 9–11, 14–17, 20, 26, 29, 30

MVC L'MVC è un *design pattern* che porta alla creazione di un'architettura divisa in Model, View e Controller. 37

MVVM L'MVVM è un design pattern che porta alla creazione di un'architettura divisa in Model, View e ViewModel. 37

Network Nel campo della BlockChain un *network* è una particolare rete sulla quale ci si può appoggiare per comunicare con gli SmartContract in essa contenuti. vii, 2, 10, 20, 22, 26, 29, 35

Screen Reader Uno screen reader è una forma di tecnologia assistiva che identifica ed interpreta il testo mostrato sullo schermo di un computer, presentandolo come *output* in sintesi vocale o tramite uno schermo *braille* [19]. 30

SmartContract Uno smartcontract o "contratto intelligente", proprio come qualsiasi altro contratto, regola i termini e le condizioni di un accordo tra le parti. I termini di uno smart contract vengono eseguiti sulla base di un codice programmato su una BlockChain come Ethereum. 1, 2, 5, 22, 26, 27, 31–33, 35, 36

Snowtrace Sito che permette di esplorare l'C-Chain per visualizzare transazioni, indirizzi e ogni altra informazione scambiata su di essa. 17

Typescript Linguaggio di programmazione che implementa "JavaScript" aggiungendo la tipizzazione. iii, 7, 21, 25

UI Letteralmente interfaccia utente, rappresenta tutto quello che è visibile all'utente finale. 37

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. 37

Wallet Letteralmente "portafoglio", contiene le criptovalute dell'utente. Possiede una chiave pubblica, visibile a tutti, che ne permette l'identificazione e una chiave privata, visibile solo al proprietario, che permette la crittazione dei dati trasferiti. 2, 6, 9–15, 18, 26, 27, 35

Acronimi

Dependecy Injection DI. 25, 37

Model View Controller MVC. 23, 36

Model View ViewModel MVVM. 25, 36

UC Use Case (caso d'uso). 10

UML Unified Modeling Language. 10, 36

User Interface UI. 22, 36

Web Content Accessibility Guidelines WCAGAA. 30, 37

Bibliografia

Siti web consultati

[1] *Angular Best Practises*. URL: https://aglowiditsolutions.com/blog/angular-best-practices/.

[2] *Angular MVC Pattern*. URL: https://medium.com/@maaouikimo/why-angular-is-your-best-choice-for-you-next-projects-9d754fb18f91.

[3] *Architettura di Angular*. URL: https://www.ngdevelop.tech/angular/architecture/.

[4] *Avalanche*. URL: https://www.criptovaluta.it/avalanche.

[5] *Blockchain*. URL: https://blog.osservatori.net/it_it/blockchain-spiegazione-significato-applicazioni.

[6] *Componenti Angular*. URL: https://dariopironi.com/it/componenti-angular-cosa-sono-e-come-crearli.

[7] *Criptovaluta*. URL: https://www.borsaitaliana.it/borsa/glossario/criptovaluta.html.

[8] *Dapp*. URL: https://blog.osservatori.net/it_it/dapp-blockchain-cosa-sono.

[9] *Decorator Design Pattern*. URL: https://www.javatpoint.com/angular-decorators.

[10] *Dependecy Injection*. URL: https://it.wikipedia.org/wiki/Dependency_injection.

[11] *Diagrammi dei casi d'uso*. URL: https://it.wikipedia.org/wiki/Use_Case_Diagram.

[12] *Ethers*. URL: https://docs.ethers.io/v5/.

[13] *Framework Angular*. URL: https://angular.io/.

[14] *Immagine Angular Service*. URL: https://www.simplilearn.com/tutorials/angular-tutorial/angular-service.

[15] *MetaMask*. URL: https://metamask.io/.

[16] *MVC Pattern*. URL: https://www.laramind.com/blog/definizione-di-mvc-cose-un-framework-mvc/.

[17] *MVVM Pattern*. URL: https://it.wikipedia.org/wiki/Model-view-viewmodel.

[18] *Remix*. URL: https://remix.ethereum.org/.

[19] *Screen Reader*. URL: https://it.wikipedia.org/wiki/Screen_reader.

[20] *Smart contract*. URL: https://ethereum.org/it/developers/docs/smart-contracts/.

[21] *Synclab*. URL: https://www.synclab.it/.

[22] *Test WCAG AA*. URL: https://www.w3.org/WAI/WCAG2AA-Conformance.

[23] *Transazioni*. URL: https://affidaty.io/blog/it/2019/07/inside-blockchain-la-transazione-come-e-da-cosa-e-composta.

[24] *Tutorial utilizzo HardHat*. URL: https://hardhat.org/tutorial.

[25] *Tutorial utilizzo Solidity*. URL: https://docs.microsoft.com/it-it/learn/modules/blockchain-solidity-ethereum-smart-contracts/.

[26] *Typescript*. URL: https://www.typescriptlang.org/.

[27] *Visual Studio Code*. URL: https://visualstudio.microsoft.com/it/.