

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA "

CORSO DI LAUREA IN INFORMATICA



**Analisi e sviluppo front-end di una web
application in ambito blockchain/NFT**

Tesi di laurea

Relatore

Prof. Gilberto Filè

Laureanda

Margherita Mitillo

1098971

ANNO ACCADEMICO 2020-2021

"Tre Anelli ai Re degli Elfi sotto il cielo che risplende,
Sette ai Principi dei Nani nelle lor rocche di pietra,
Nove agli Uomini Mortali che la triste morte attende,
Uno per l'Oscuro Sire chiuso nella reggia tetra
Nella Terra di Mordor, dove l'Ombra nera scende.
Un Anello per domarli, Un Anello per trovarli,
Un Anello per ghermirli e nel buio incatenarli,
Nella Terra di Mordor, dove l'Ombra cupa scende."

— John Ronald Reuel Tolkien

Dedicato a Daniele, Alessandra, Gianni e Luciano

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage dalla laureanda Margherita Mitillo, della durata di circa trecento ore, presso l'azienda Sync Lab S.r.l. supervisionato e coordinato dal tutor aziendale Fabio Scettro. Lo scopo principale era lo sviluppo di maschere per il `front end`_G (ovvero interfacce utente), dell'applicazione NFTLab tramite il `framework`_G Vue.js.

Oltre a questo era previsto lo studio delle tecnologie coinvolte per la progettazione e la codifica del prodotto. Infine era richiesta l'implementazione di tali interfacce e la stesura di un documento tecnico che raccolga la descrizione di ciò che è stato implementato.

“Certe strade, è meglio intraprenderle che rifiutarle, anche se il loro esito è oscuro”

— John Ronald Reuel Tolkien

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Gilberto Filè, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero inoltre ringraziare il mio tutor aziendale Fabio Scettro per avermi seguito durante il periodo di stage.

Ringrazio con tanto affetto il mio ragazzo Daniele per avermi supportato e aver sempre creduto in me durante tutto il periodo di studi. Senza di lui probabilmente non sarei arrivata fino a qui.

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

Ringrazio mia sorella Camilla e mio fratello Marco per i momenti di svago e le risate che mi hanno fatto vivere questo periodo di studio più spensieratamente.

Desidero inoltre di ringraziare poi i miei amici, in particolare Carlotta e Marco G, per i bellissimi anni passati assieme.

Padova, Luglio 2021

Margherita Mitillo

Indice

1	Introduzione	1
1.1	Organizzazione del testo	1
1.2	Regole tipografiche	1
2	Descrizione dello stage	3
2.1	L'azienda	3
2.1.1	La storia	3
2.1.2	I prodotti	4
2.2	L'idea dello stage	5
2.2.1	Il gruppo di lavoro	5
2.3	Organizzazione del lavoro	6
2.3.1	Modello di sviluppo	6
2.3.2	Strumenti per l'organizzazione e la comunicazione	7
2.3.2.1	Trello	7
2.3.2.2	Report giornaliero	8
2.3.2.3	Discord	8
2.3.2.4	Notion	9
2.4	Pianificazione del lavoro	9
2.5	Requisiti ed obiettivi	9
2.5.1	Notazione	9
2.5.2	Obiettivi fissati	10
3	Analisi dei requisiti	11
3.1	Casi d'uso	11
3.1.1	Attori dei casi d'uso	11
3.2	Tracciamento dei requisiti	30
4	Progettazione e codifica	33
4.1	Tecnologie e strumenti	33
4.1.1	Tecnologie	33
4.1.1.1	Vue.js	33
4.1.1.2	Vuetify	34
4.1.1.3	Vuex	35
4.1.2	Strumenti	36
4.1.2.1	Github	36
4.1.2.2	Figma	37
4.1.2.3	Stoplight	37
4.1.2.4	Visual Studio Code	37

4.2	Organizzazione dei file	38
4.3	Chiamate al back end	39
4.3.1	CurrentUser.js	40
4.3.2	NftService.js	42
4.4	Maschere implementate	44
4.4.1	Barra di navigazione	44
4.4.2	Home	45
4.4.3	Visualizza dettagli	45
4.4.4	Login	46
4.4.5	Registrazione	46
4.4.6	Pagina dell'utente	47
4.4.7	Upload opera	48
4.4.8	Gestione opere	48
4.4.9	Modifica dati personali	49
4.4.10	Modifica password	49
4.5	Generazione della preview	50
5	Verifica e validazione	51
5.1	Strumenti per la verifica	51
5.2	Verifica	51
5.3	Validazione e collaudo	52
6	Nozioni apprese	53
6.1	Organizzazione dello studio	53
6.2	Nozioni principali	53
6.2.1	Javascript	53
6.2.2	Vue.js	54
6.3	Nozioni secondarie	54
6.3.1	Java	54
6.3.2	Concetti web	55
6.3.2.1	Servlet	55
6.3.2.2	REST	55
6.3.3	Spring	56
6.3.3.1	Spring Boot	56
6.3.3.2	Spring Data	57
6.3.3.3	Spring Data JPA	57
6.3.3.4	Spring Data REST	57
6.3.4	Blockchain	57
6.3.5	Ethereum	58
6.3.6	NFT	58
7	Conclusioni	59
7.1	Raggiungimento degli obiettivi	59
7.2	Conoscenze acquisite	60
7.3	Valutazione personale	60
A	Appendice A	61
A.1	Confronto tra framework di sviluppo	61
	Glossario	63

<i>INDICE</i>	xi
Acronimi	65
Bibliografia	67

Elenco delle figure

2.1	Logo di Sync Lab	3
2.2	Motto e punti di forza di Sync Lab	4
2.3	Logo di NFTLab	5
2.4	Metodo Scrum	7
2.5	Spezzone del report giornaliero	8
3.1	U.C0 Scenario principale	12
3.2	U.C1 Registrazione	13
3.3	U.C1.1 Inserimento dei dati per la registrazione	14
3.4	U.C2 Login	17
3.5	U.C2.1 Inserimento dati per il login	18
3.6	U.C4 Visualizzazione lista opere	20
3.7	U.C5 Visualizzazione pagina personale	21
3.8	U.C5.1 Modifica dati	22
3.9	U.C5.2 Visualizzazione lista opere personali	24
3.10	U.C5.2.2 Modifica opera	25
3.11	U.C5.3 Upload di una nuova opera	27
4.1	Logo di Vue.js	33
4.2	Funzionalità di Vuetify	35
4.3	Pattern di Vuex	36
4.4	Home Page	45
4.5	Finestra di dialogo delle informazioni dell'opera	45
4.6	Pagina di login	46
4.7	Pagina di registrazione	47
4.8	Pagina personale dell'utente	47
4.9	Pagina di upload dell'opera	48
4.10	Finestra di dialogo per la modifica dell'opera	49
4.11	Finestra di dialogo per la modifica dei dati	49
4.12	Finestra di dialogo per la modifica della password	50

Elenco delle tabelle

2.1	Tabella della pianificazione del lavoro	9
3.1	Tabella del tracciamento dei requisiti funzionali	31
3.2	Continuazione della tabella del tracciamento dei requisiti funzionali . .	32
3.3	Tabella del tracciamento dei requisiti qualitativi	32
3.4	Tabella del tracciamento dei requisiti di vincolo	32
7.1	Tabella dello stato di soddisfacimento degli obiettivi	59
A.1	Tabella riassuntiva delle differenze tra Vue.js ed Angular.js	61

Listings

4.1	Organizzazione dei file.	38
4.2	Autenticazione.	40
4.3	Registrazione.	40
4.4	Logout.	41
4.5	Modifica dei dati personali.	41
4.6	Modifica della password.	42
4.7	Opere dell'utente.	42
4.8	Categorie disponibili.	43
4.9	Caricamento di una nuova opera.	43
4.10	Modifica di un'opera esistente.	43
4.11	Visualizzazione delle opere nella home.	44

5.1	Esempio di test-Pagina di login.	51
-----	--	----

Capitolo 1

Introduzione

Lo scopo di questo progetto di stage è studiare il [framework](#) Vue.js per implementare parte delle componenti di [front end](#) di un'applicazione web in ambito [blockchain_G](#) e [NFT_G](#) chiamata NFTLab.

Tramite questa applicazione sarà possibile per l'utente creare un proprio profilo per poter compiere diverse azioni, come vendere le proprie opere multimediali, modificare alcuni dati di tali opere, visualizzare opere di altri utenti e comprare opere multimediali di altri utenti.

Per realizzare questi aspetti sono state implementate diverse maschere, ovvero interfacce utente, in collaborazione con gli altri colleghi stagisti.

Infatti un aspetto importante di questo progetto è la collaborazione con gli altri componenti del gruppo per integrare il proprio lavoro con quello degli altri, in particolare per il corretto funzionamento della [web application_G](#) è necessaria l'integrazione tra [front end](#) e [back end_G](#).

1.1 Organizzazione del testo

[Il secondo capitolo](#) approfondisce la descrizione dello stage, l'organizzazione del lavoro e il progetto da sviluppare;

[Il terzo capitolo](#) approfondisce il processo di analisi dei requisiti e il tracciamento di essi;

[Il quarto capitolo](#) approfondisce i processi di progettazione e codifica descrivendo anche le tecnologie e gli strumenti utilizzati;

[Il quinto capitolo](#) approfondisce i processi di validazione e verifica del codice prodotto;

[Il sesto capitolo](#) approfondisce le nozioni apprese nel periodo di stage;

[Il settimo capitolo](#) descrive le conclusioni tratte alla fine del periodo di stage.

1.2 Regole 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*;
- * per le successive occorrenze dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*;
- * i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.
- * i nomi dei file o delle cartelle sono evidenziati con il carattere corsivo grassetto ***nome***.

Capitolo 2

Descrizione dello stage

In questo capitolo verranno descritti in dettaglio l'idea dello stage, l'azienda dove è stato svolto e come è stato organizzato il lavoro.

2.1 L'azienda

2.1.1 La storia



Figura 2.1: Logo di Sync Lab

Sync Lab S.r.l. è un'azienda di consulenza informatica nata nel 2002 a Napoli e trasformatasi molto velocemente nel corso degli anni in *System Integrator* grazie ad un processo di maturazione delle competenze tecnologiche.

L'azienda supporta le esigenze di innovazione dei clienti offrendo soluzioni IT in ambito *Business Consultancy*, *Project Financing* e *IT Consultancy*.

Ad oggi l'azienda conta un organico di circa 200 dipendenti ed è riuscita a coprire tutto il territorio nazionale fino ad ottenere un totale di cinque sedi tra Roma, Napoli, Verona, Padova e Milano. L'azienda propone sul mercato prodotti software ed attraverso essi ha gradualmente conquistato significativamente fette di mercato nei seguenti settori: mobile, videosorveglianza e sicurezza delle infrastrutture informatiche aziendali.

L'obiettivo è quello di supportare il cliente nella Realizzazione, Messa in Opera e *Governance* di soluzioni IT, sia dal punto di vista Tecnologico, sia nel Governo del Cambiamento Organizzativo.



Figura 2.2: Motto e punti di forza di Sync Lab

2.1.2 I prodotti

I prodotti offerti più noti dell'azienda sono i seguenti:

- * **SynClinic:** sistema informativo sanitario per la gestione dei processi clinici ed amministrativi di ospedali, cliniche e case di cura. Lo scopo è diventare uno strumento indispensabile di supporto per il personale nella gestione del rischio clinico di ogni paziente;
- * **DPS4:** [web application](#) per gestire gli adempimenti della privacy GDPR, acronimo per *General Data Protection Regulation*, con lo scopo di rafforzare la privacy dei dati dei cittadini;
- * **StreamLog:** sistema in grado di effettuare in modo semplice ed efficace il controllo degli accessi degli utenti ai sistemi;
- * **StreamCrusher:** software in grado di analizzare i dati che aiuta ad essere informati adeguatamente quando bisogna prendere decisioni di business, ad identificare criticità e a riorganizzare il sistema;
- * **Wave:** acronimo per *Wide Area Videosurveillance Environment*, è un prodotto software nato per ottenere un'integrazione sinergica tra i mondi della sorveglianza e dei Sistemi Informativi Territoriali (GIS) in modo da ottenere una copertura totale del territorio;
- * **Seastream:** piattaforma in grado di migliorare la sicurezza, l'efficienza e il processo di innovazione in campo marittimo.

2.2 L'idea dello stage

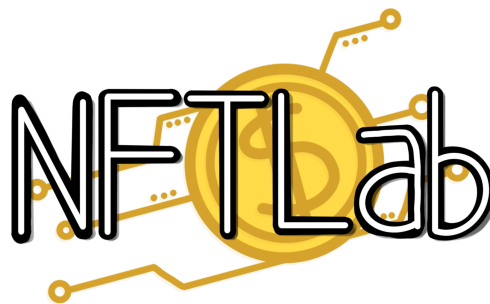


Figura 2.3: Logo di NFTLab

Il progetto da svolgere proposto dall'azienda per lo stage è una [web application](#) chiamata NFTLab legata all'ambito [blockchain](#) e [NFT](#). Nonostante siano due tecnologie relativamente recenti, in questo ultimo anno la loro popolarità è cresciuta in maniera esponenziale e questo le porta ad essere un argomento stimolante ed interessante per il progetto di stage.

All'interno della [web application](#) l'utente può navigare nella pagina principale e visualizzare le opere multimediali poste in vendita, eseguire l'accesso se possiede un account nel sito oppure registrarsi come nuovo utente. Le opere multimediali possono essere delle immagini, dei video, delle *gif*, degli audio o dei documenti.

Dopo aver effettuato l'accesso, l'utente può modificare i propri dati, inserire una nuova opera multimediale, modificare i dati di quelle precedentemente inserite e comprare un'opera multimediale caricata nel sito da un altro utente.

Essendo un progetto legato al concetto di [blockchain](#) e [Non-Fungible Token \(NFT\)](#) le azioni di compravendita verranno effettuate tramite lo scambio di una moneta virtuale detta [Ethereum](#). Un utente viene identificato tramite l'indirizzo del *wallet* con cui acquisterà o caricherà una nuova opera multimediale.

Quando un utente carica un'opera multimediale, il file viene salvato tramite codice *hash* nella [blockchain](#) ed in questo modo le opere caricate saranno univoche. Infatti salvando queste informazioni nella [blockchain](#) viene creato un *timestamp* che funge da certificato di attribuzione dell'opera all'utente che carica o compra l'opera multimediale. In questo modo nessun altro utente può registrare la stessa opera a suo nome poichè una funzione hash produce sempre e solo un risultato rendendo uniche le opere multimediali.

Per realizzare questi aspetti sono state implementate diverse maschere, ovvero interfacce utente, tramite il [framework](#) [Vue.js](#), uno strumento utile allo sviluppo di [web application](#) reattive e a pagina singola.

Parte del percorso di stage è stata riservata ad un periodo di studio delle tecnologie necessarie allo sviluppo delle maschere e ad un periodo di studio delle tecnologie legate al [back end](#) per poter collaborare al meglio con i colleghi stagisti che lavorano su questo aspetto.

2.2.1 Il gruppo di lavoro

Il gruppo di stagisti incaricati allo sviluppo di questo progetto è composto dai seguenti membri:

- * io, Margherita, addetta allo sviluppo di maschere per il [front end](#) tramite il [framework](#) Vue.js;
- * Filippo Fantinato, addetto allo studio e all'integrazione della [blockchain](#) con il [back end](#);
- * Andrea Cecchin, addetto allo sviluppo del [back end](#);
- * Michele Baldisseri, addetto allo sviluppo di maschere per il [front end](#) tramite il [framework](#) Angular.js_G.

2.3 Organizzazione del lavoro

2.3.1 Modello di sviluppo

L'azienda adotta il modello di sviluppo *Agile* che si basa sull'interazione continua con gli [stakeholder](#)_G e predilige:

- * gli individui e le loro interazioni piuttosto che i processi e gli strumenti;
- * il software funzionante piuttosto che una documentazione esaustiva;
- * la collaborazione con il cliente piuttosto che la negoziazione dei contratti;
- * essere propositivi verso il cambiamento piuttosto che rifiutarlo.

L'idea di questo modello non si basa quindi su uno sviluppo sequenziale ma sul concetto di rivedere di continuo le specifiche adeguandole durante l'avanzamento dello sviluppo software. In questo modo si possono apportare agilmente modifiche mediante metodologie iterative ed incrementali con un continuo scambio di pareri ed informazioni tra gli sviluppatori ed il committente. Il progetto di stage è stato portato avanti utilizzando il metodo [Scrum](#)_G.

[Scrum](#) è il metodo più diffuso tra i modelli di sviluppo e prevede di dividere il progetto in blocchi di lavoro detti *sprint*, alla fine dei quali si crea un incremento del prodotto software. Esso prevede vari *meeting* per controllare il lavoro svolto ed organizzare il lavoro da svolgere nel prossimo periodo. Inoltre nel corso dello sviluppo del progetto l'azienda organizza diversi *meeting* con gli [stakeholder](#) e i membri del progetto per valutare l'andamento dello sprint.

Gli incontri legati a questo modello di sviluppo che vengono svolti di solito sono:

- * *sprint planning meeting*: tenuto ad inizio *sprint* per poter organizzare il lavoro;
- * *daily scrum*: riunione giornaliera del team di sviluppo per monitorare l'andamento dello *sprint*;
- * *sprint review*: tenuto a fine *sprint* per valutare i risultati ottenuti e quali cambiamenti apportare per lo sprint successivo.

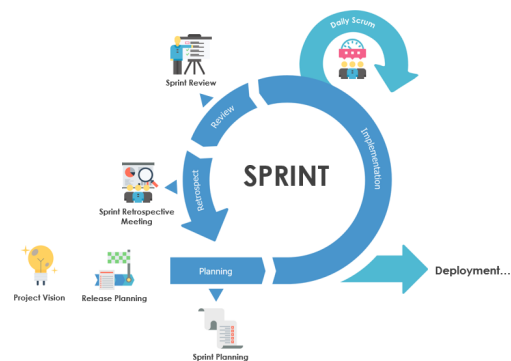


Figura 2.4: Metodo Scrum

La situazione di emergenza sanitaria ancora non conclusa mi ha portato a vivere un'esperienza lavorativa diversa dai colleghi degli anni precedenti, infatti ha fatto sì che lo stage fosse organizzato in modalità mista: in parte da remoto ed in parte in azienda (solitamente uno o due giorni alla settimana). A causa di ciò non ho potuto vivere appieno questo modello di sviluppo vista l'impossibilità di essere sempre presente in azienda. Per questo motivo il giorno concordato con i colleghi stagisti ed il tutor aziendale per andare in azienda è stato sfruttato per fare un *daily scrum* alternativo data la sua cadenza settimanale.

Nonostante l'emergenza sanitaria ancora in corso ho potuto comunque constatare l'efficacia e la funzionalità di questo modello di sviluppo, soprattutto per un progetto software in evoluzione quale è stato quello a cui ho lavorato.

2.3.2 Strumenti per l'organizzazione e la comunicazione

2.3.2.1 Trello

Trello è una piattaforma online tramite la quale si possono gestire i progetti in modo semplice, gratuito e flessibile. Questa piattaforma risulta essere molto utile per l'organizzazione e la gestione del *workflow* online.

Tramite questa piattaforma è possibile rappresentare le attività da svolgere in schede inserite sotto apposite colonne di avanzamento. In base a quanto si è riusciti a realizzare, è possibile spostare manualmente o automaticamente lo stato di avanzamento di ogni scheda. Le diciture più comuni delle colonne utilizzate nella piattaforma sono le seguenti:

- * **attività:** elenco di schede della attività da svolgere nel corso del progetto;
- * **in corso:** elenco di schede delle attività in corso di sviluppo;
- * **in verifica:** elenco di schede delle attività in corso di verifica;
- * **conclude:** elenco di schede delle attività concluse.

Per ogni scheda inoltre è possibile indicare chi svolgerà quell'attività, commenti relativi ad essa, link ad informazioni utili per il suo svolgimento ed un elenco puntato rappresentante delle sotto-attività da compiere per il corretto svolgimento dell'attività. Proprio per i numerosi vantaggi offerti dalla piattaforma l'azienda ha deciso di utilizzarla creando due bacheche:

- * **bacheca personale:** disponibile solo allo stagista ed al tutor aziendale, dove sono stati inseriti i compiti decisi nel piano di lavoro suddivisi per settimana;
- * **bacheca collettiva:** disponibile a tutti gli stagisti che lavorano allo stesso progetto e ai loro rispettivi tutor aziendali, dove sono state inserite le prime idee e funzionalità legate al progetto.

Nella bacheca personale sono presenti diverse schede di numero pari alle settimane di stage da svolgere mentre nella bacheca collettiva il numero delle schede dipende dal numero di funzionalità e servizi da sviluppare.

2.3.2.2 Report giornaliero

L'azienda ha deciso inoltre di far scrivere ad ogni stagista un file Excel su Drive come report giornaliero delle attività svolte. Grazie ad esso il tutor poteva monitorare lo stagista anche da remoto nei giorni in cui non era richiesta la presenza in azienda. Il report è suddiviso in quattro colonne:

- * **giorno:** indica la data in cui è stata fatta un'attività in formato AAAA-MM-GG, in ordine cronologico dal giorno di inizio al giorno di fine stage;
- * **descrizione:** breve descrizione dell'attività svolta dal tirocinante compilata solitamente alla fine di ogni giornata lavorativa;
- * **nota:** campo dove il tutor aziendale può aggiungere note di consiglio riguardo all'attività svolta come suggerimenti sugli argomenti da leggere o link a siti utili, inoltre in questo campo viene specificato quando cadono i giorni festivi;
- * **spunta:** casella di spunta dove il tutor indica di aver visionato l'attività svolta.

Giorno	Attività	Nota	Conferma del tutor
2021-05-03	Organizzazione lavoro, ripasso Java e Servlet		ok
2021-05-04	Ripasso del concetto di JSON, inizio dello studio del pattern REST	per approfondire Rest vs Restful: https://martinfowler.com/articles/richardsonMaturityModel.html	ok
2021-05-05	Continuazione studio del pattern REST		ok
2021-05-06	Studiato Rest vs Restful, principi del REST e best practice delle REST API	Consiglio: se non l'hai mai visto dai un'occhiata a postman	ok
2021-05-07	Concluso lo studio di REST con esercizi pratici e ripasso generale delle conoscenze apprese nel corso della settimana		ok

Figura 2.5: Spezzione del report giornaliero

2.3.2.3 Discord

Discord è una piattaforma gratuita per la comunicazione dove è possibile utilizzare canali sia testuali, per lo scambio di messaggi, che vocali, per effettuare delle chiamate. Inoltre è possibile creare dei server per poter comunicare con più persone contemporaneamente, ed è possibile suddividere in canali diversi in base all'argomento in modo da rendere più chiara ed efficiente la conversazione degli utenti partecipanti al server.

Per comunicare con l'intero gruppo di stagisti l'azienda ha proposto di utilizzare questa piattaforma dove noi stagisti potevamo comunicare sia con l'intero personale aziendale che singolarmente con il proprio tutor per consigli, delucidazioni o motivi organizzativi.

2.3.2.4 Notion

Notion è un'applicazione che fornisce elementi come note, database, bacheche *kanban*, *wiki*, calendari e promemoria. L'utente può collegare questi elementi tra di loro per creare il proprio sistema per la gestione della presa di appunti, la conoscenza, la gestione dei dati e la gestione dei progetti. Questo può essere un sistema da utilizzare da soli oppure in collaborazione.

L'azienda ha deciso di adottare questa applicazione per segnare le presenze in azienda in modo da potersi organizzare tra i dipendenti e noi stagisti e poter rispettare le norme vigenti sul Covid.

2.4 Pianificazione del lavoro

Come definito dal piano di lavoro, lo stage è composto da un primo periodo di studio ed approfondimento teorico dei concetti e delle tecnologie ed un secondo momento di implementazione e sviluppo. La ripartizione oraria è così definita:

Tabella 2.1: Tabella della pianificazione del lavoro

Durata in ore	Descrizione dell'attività
40	Presentazione dell'ambiente di sviluppo, analisi del progetto da svolgere, verifica delle credenziali assegnate e ripasso del linguaggio Java e dei concetti Web
40	Studio dei principi generali di Spring Core (IOC e Dependency Injection), Spring Boot e Spring Data REST
40	Studio di Spring Data JPA e studio dei concetti generali di Blockchain, Ethereum, NFT
40	Ripasso del linguaggio di Javascript e studio del framework Vue.js
40	Analisi e studio del progetto NFTLab, progettazione ed implementazione della nuova maschera di login
40	Progettazione ed implementazione della nuova maschera "Upload Opera", scrittura dei service su front end di chiamata al back end ed implementazione della maschera "Gestione Opere dell'utente"
40	Termine delle implementazioni ed integrazione con l'applicativo
40	Termine delle integrazioni e collaudo finale

2.5 Requisiti ed obiettivi

Durante la compilazione del Piano di Lavoro, documento necessario per l'inizio dello stage, sono stati definiti degli obiettivi di raggiungimento per la conclusione dello stage.

2.5.1 Notazione

La notazione tramite la quale si farà riferimento agli obiettivi è così strutturata:

[classificazione][numero]

La descrizione del codice è la seguente:

* **classificazione:** individua la classificazione del requisito e può essere:

O = obbligatorio

D = desiderabile

F = facoltativo

* **numero:** coppia sequenziale di numeri identificativa del requisito.

2.5.2 Obiettivi fissati

Gli obiettivi fissati sono i seguenti:

* **obbligatori:**

- O01: Acquisizione delle 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: Riuscire a studiare e prevedere nell'implementazione la gestione del JWT Token durante la navigazione all'interno della web app.

Capitolo 3

Analisi dei requisiti

In questo capitolo verranno elencati i casi d'uso delle funzionalità implementate con i relativi requisiti ed i loro tracciamento

3.1 Casi d'uso

Per lo studio dei casi d'uso del prodotto creato sono stati creati dei diagrammi. Questi diagrammi, detti appunto dei casi d'uso (in inglese *Use Case Diagram*), sono diagrammi di tipo [Unified Modeling Language \(UML\)](#) dedicati alla descrizione del sistema e delle funzioni o servizi offerti da un esso, e come gli utilizzatori interagiscono con esso.

Lo strumento utilizzato per la realizzazione di tali diagrammi è draw.io, piattaforma online che permette di creare diagrammi direttamente nel browser e salvarli nel proprio *cloud* di *Google*.

3.1.1 Attori dei casi d'uso

Dopo un'attenta analisi ho concluso che per le funzionalità offerte sono presenti unicamente attori primari, in quanto non ci sono collegamenti con alcun [framework](#) o libreria esterna. Di conseguenza i possibili attori dei casi d'uso analizzati sono i seguenti attori primari:

- * **utente non autenticato:** indica che l'utente non ha ancora effettuato l'autenticazione o la registrazione all'interno della [web application](#);
- * **utente autenticato:** indica che l'utente ha effettuato l'autenticazione all'interno della [web application](#) e comporta che può accedere a diverse funzionalità che sarebbero altrimenti inaccessibili.

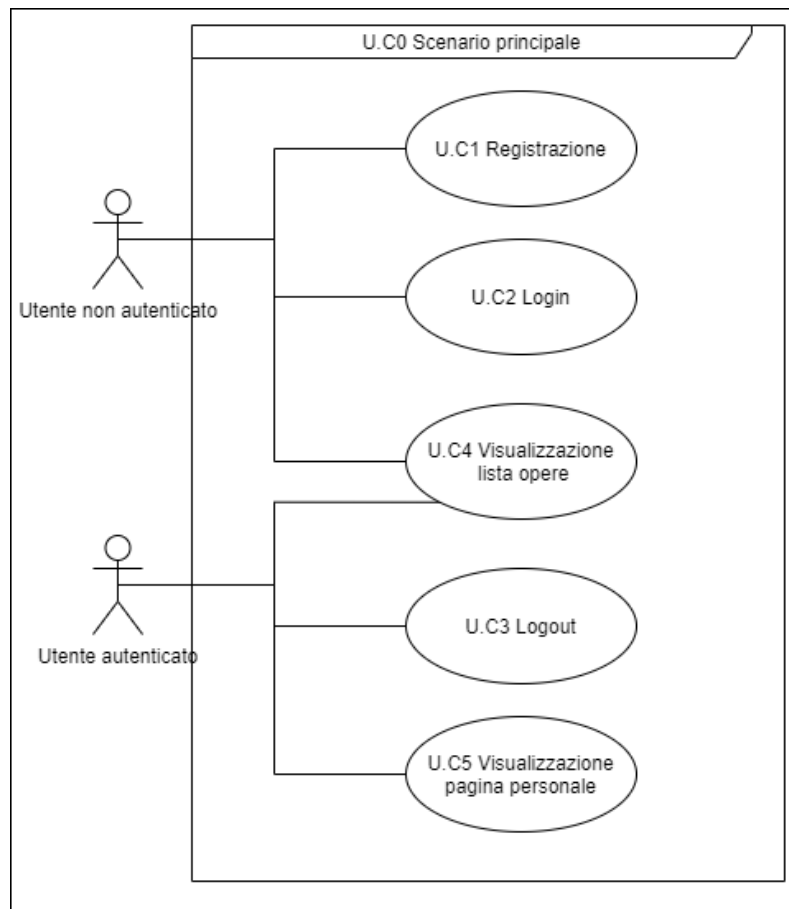
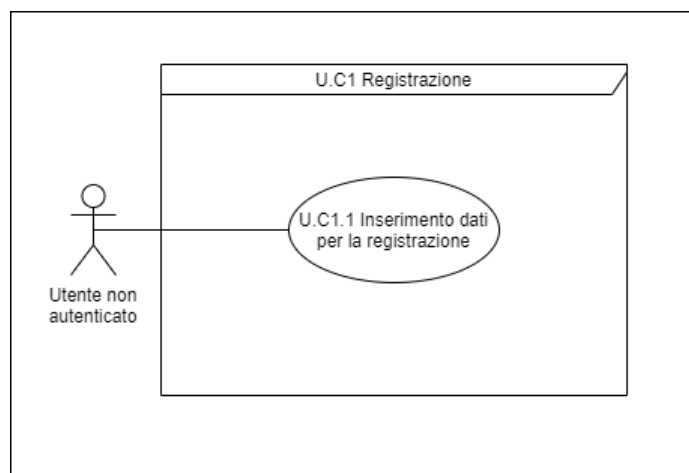


Figura 3.1: U.C0 Scenario principale

UC1: Registrazione**Figura 3.2:** U.C1 Registrazione

Attori Principali: Utente non autenticato.

Precondizioni: L'utente non è ancora presente nei registri del sistema.

Descrizione: L'utente accede all'applicazione e naviga fino alla pagina di registrazione. L'utente inserisce i dati necessari e li conferma, questo porta l'utente a possedere un account nel sistema.

Postcondizioni: L'utente risulta presente nei registri del sistema ed è autenticato nella piattaforma.

UC1.1: Inserimento dati per la registrazione

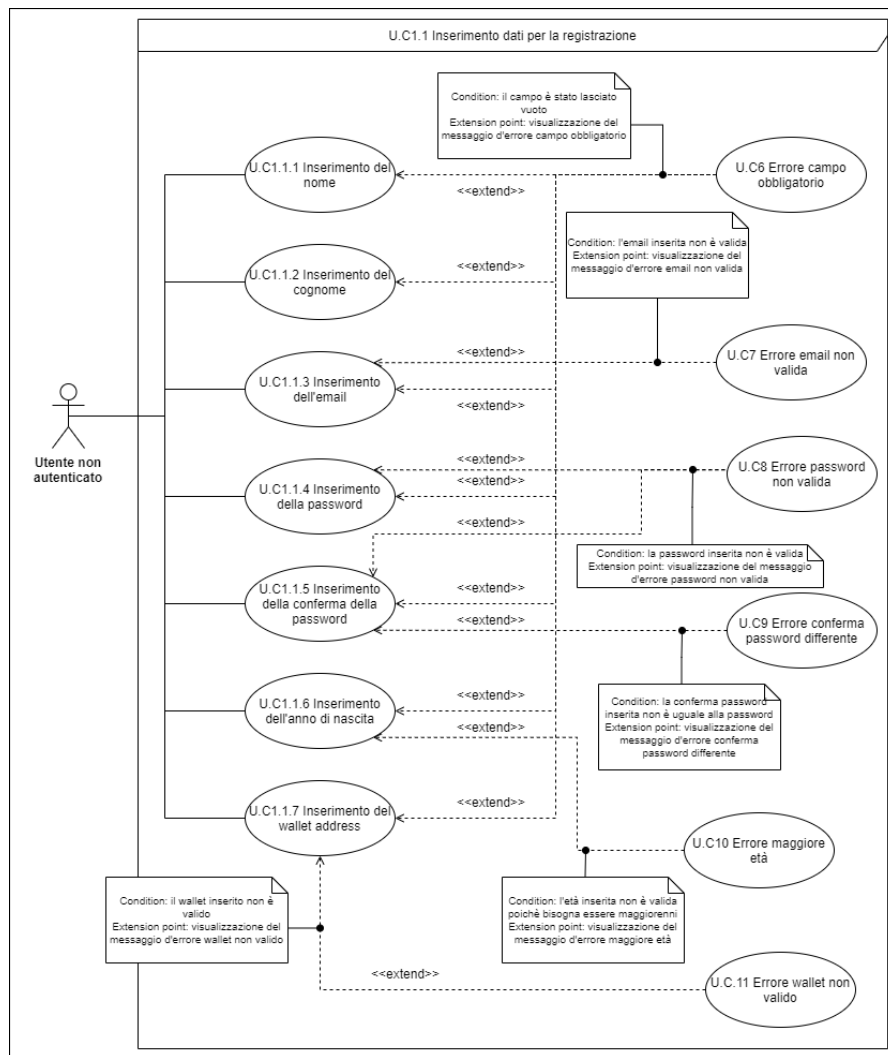


Figura 3.3: U.C1.1 Inserimento dei dati per la registrazione

Attori Principali: Utente non autenticato.

Precondizioni: L'utente non autenticato si trova nella pagina di registrazione.

Descrizione: L'utente non autenticato compila i campi nel seguente modo:

- * Inserimento del nome ([UC1.1.1](#));
- * Inserimento del cognome ([UC1.1.2](#));
- * Inserimento dell'email ([UC1.1.3](#));
- * Inserimento della password ([UC1.1.4](#));
- * Inserimento della conferma della password ([UC1.1.5](#));
- * Inserimento dell'anno di nascita ([UC1.1.6](#));

- * Inserimento del Wallet Address ([UC1.1.7](#)).

Postcondizioni: L'utente ha completato la compilazione dei campi e può procedere con la registrazione.

UC1.1.1: Inserimento del nome

Attori Principali: Utente non autenticato.

Precondizioni: L'utente non è autenticato e non ha compilato questo campo.

Descrizione: L'utente non autenticato inserisce il proprio nome.

Estensioni: Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato ([UC6](#)).

Postcondizioni: L'utente ha inserito il nome.

Estensioni: L'utente non inserisce nulla e appare il messaggio di errore.

UC1.1.2: Inserimento del cognome

Attori Principali: Utente non autenticato.

Precondizioni: L'utente non è autenticato e non ha compilato questo campo.

Descrizione: L'utente non autenticato inserisce il proprio cognome.

Estensioni: Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato ([UC6](#)).

Postcondizioni: L'utente ha inserito il cognome.

UC1.1.3: Inserimento dell'email

Attori Principali: Utente non autenticato.

Precondizioni: L'utente non è autenticato e non ha compilato questo campo.

Descrizione: L'utente non autenticato inserisce la sua email.

Estensioni:

- * Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato ([UC6](#));
- * Viene effettuato un controllo sul campo inserito e risulta non essere valido quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato ([UC7](#)).

Postcondizioni: L'utente ha inserito l'email.

UC1.1.4: Inserimento della password

Attori Principali: Utente non autenticato.

Precondizioni: L'utente non è autenticato e non ha compilato questo campo.

Descrizione: L'utente non autenticato inserisce la sua password.

Estensioni:

- * Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC6);
- * Viene effettuato un controllo sul campo inserito e risulta non essere valido quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC7).

Postcondizioni: L'utente ha inserito la password.

UC1.1.5: Inserimento della conferma della password

Attori Principali: Utente non autenticato.

Precondizioni: L'utente non è autenticato e non ha compilato questo campo.

Descrizione: L'utente non autenticato inserisce la conferma della password.

Estensioni:

- * Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC6);
- * Viene effettuato un controllo sul campo inserito e risulta non essere valido quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC8);
- * Viene effettuato un controllo di uguaglianza con la password precedentemente inserita e i due campi non risultano uguali quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC9).

Postcondizioni: L'utente ha inserito la conferma della password.

UC1.1.6: Inserimento dell'anno di nascita

Attori Principali: Utente non autenticato.

Precondizioni: L'utente non è autenticato e non ha compilato questo campo.

Descrizione: L'utente non autenticato inserisce il proprio anno di nascita.

Estensioni:

- * Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC6);

- * Viene effettuato un controllo sul campo inserito e risulta non essere valido in quanto l'utente risulta essere non maggiorenne quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (??).

Postcondizioni: L'utente ha inserito l'anno di nascita.

UC1.1.7: Inserimento del Wallet Address

Attori Principali: Utente non autenticato.

Precondizioni: L'utente non è autenticato e non ha compilato questo campo.

Descrizione: L'utente non autenticato inserisce il Wallet Address.

Estensioni:

- * Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC6);
- * Viene effettuato un controllo sul campo inserito e risulta non essere valido quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (??).

Postcondizioni: L'utente ha inserito il Wallet Address.

UC2: Login

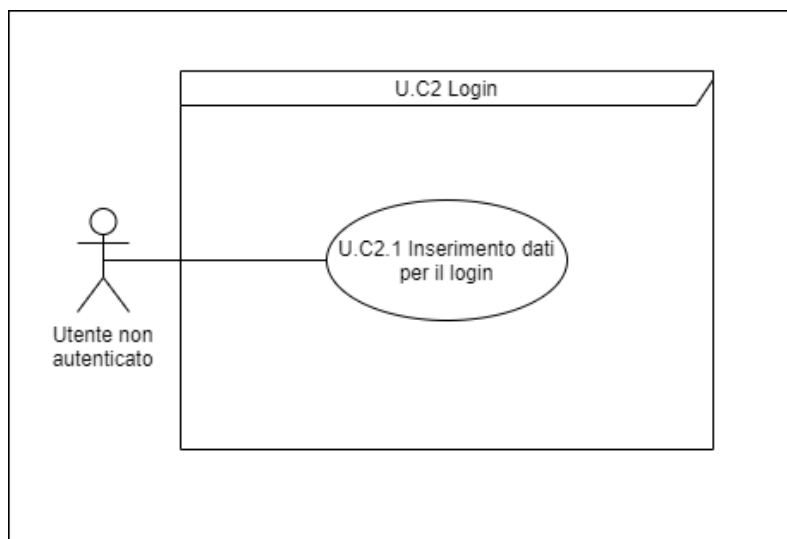


Figura 3.4: U.C2 Login

Attori Principali: Utente non autenticato.

Precondizioni: L'utente non è autenticato ma è presente nei registri di sistema.

Descrizione: L'utente accede all'applicazione e naviga fino alla pagina di login. L'utente inserisce i dati necessari (UC2.1) e li conferma, questo porta l'utente ad autenticarsi nel sistema.

Estensioni: L'utente inserisce i dati ma non è presente nel database quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente i dati (UC12).

Postcondizioni: L'utente risulta autenticato nella piattaforma.

UC2.1: Inserimento dati per il login

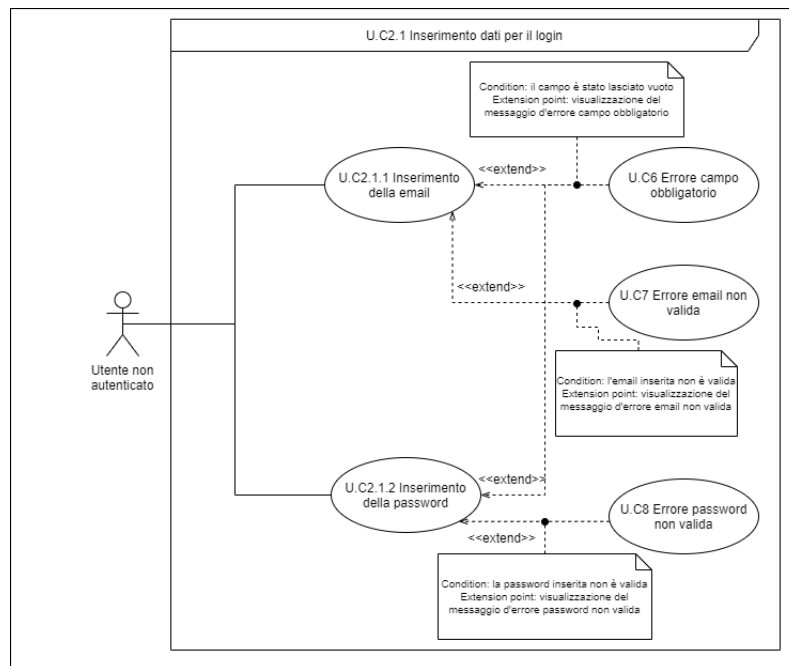


Figura 3.5: U.C2.1 Inserimento dati per il login

Attori Principali: Utente non autenticato.

Precondizioni: L'utente non autenticato si trova nella pagina di login.

Descrizione: L'utente non autenticato compila i campi nel seguente modo:

- * Inserimento della email (UC2.1.1);
- * Inserimento della password (UC2.1.2).

Postcondizioni: L'utente è autenticato come utente autenticato all'interno della [web application](#).

UC2.1.1: Inserimento della email

Attori Principali: Utente non autenticato.

Precondizioni: L'utente non autenticato si trova nella pagina di login e possiede le credenziali di accesso.

Descrizione: L'utente, per procedere con l'autenticazione inserisce l'email.

Estensioni:

- * Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC6);
- * Viene effettuato un controllo sul campo inserito e risulta non essere valido quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC7).

Postcondizioni: L'utente ha inserito l'email.

UC2.1.2: Inserimento della password

Attori Principali: Utente non autenticato.

Precondizioni: L'utente non autenticato si trova nella pagina di login e possiede le credenziali di accesso.

Descrizione: L'utente, per procedere con l'autenticazione inserisce la password.

Estensioni:

- * Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC6);
- * Viene effettuato un controllo sul campo inserito e risulta non essere valido quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC8).

Postcondizioni: L'utente ha inserito la password.

UC3: Logout

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e vuole uscire dal proprio account.

Descrizione: L'utente vuole uscire dal proprio account

Postcondizioni: L'utente non è più autenticato

UC4: Visualizzazione lista opere

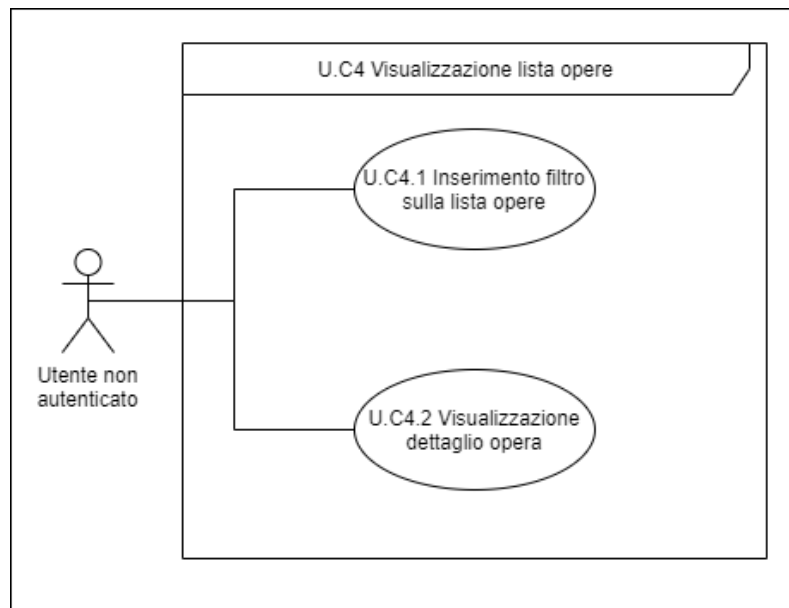


Figura 3.6: U.C4 Visualizzazione lista opere

Attori Principali: Utente non autenticato.

Precondizioni: L'utente ha aperto il sito e si trova nella pagina iniziale.

Descrizione: L'utente può visualizzare la lista delle opere in vendita. In particolare l'utente può filtrare le opere ([UC4.1](#)) oppure visualizzare nel dettaglio un'opera selezionata ([UC4.2](#))

Postcondizioni: L'utente ha visualizzato la lista.

UC4.1: Inserimento filtro sulla lista opere

Attori Principali: Utente non autenticato.

Precondizioni: L'utente si trova nella pagina principale e sta visualizzando le opere.

Descrizione: L'utente può applicare dei filtri alla lista di opere presenti nella homepage.

Postcondizioni: L'utente ha visualizzato la lista di opere filtrate.

UC4.1.1: Inserimento filtro per categorie sulla lista opere

Attori Principali: Utente non autenticato.

Precondizioni: L'utente si trova nella pagina principale e sta visualizzando le opere.

Descrizione: L'utente può applicare dei filtri per categoria alla lista di opere presenti nella homepage.

Postcondizioni: L'utente ha visualizzato la lista di opere filtrate per categoria.

UC4.2: Visualizzazione dettaglio opera

Attori Principali: Utente non autenticato.

Precondizioni: L'utente si trova nella pagina principale.

Descrizione: L'utente può selezionare un'opera per poter visualizzare le sue informazioni nel dettaglio.

Postcondizioni: L'utente ha visualizzato i dettagli dell'opera selezionata.

UC5: Visualizzazione pagina personale

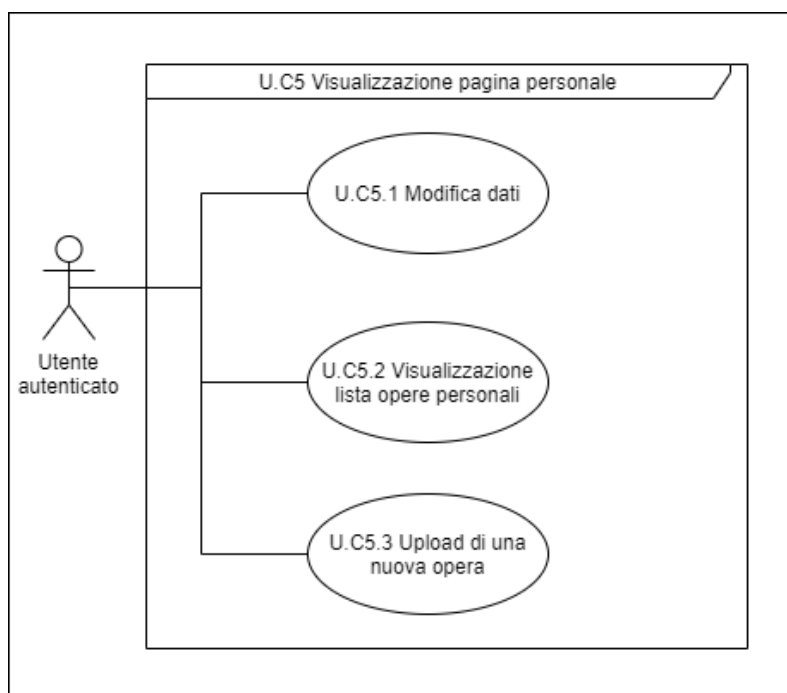


Figura 3.7: U.C5 Visualizzazione pagina personale

Attori Principali: Utente autenticato.

Precondizioni: L'utente si trova nella home page.

Descrizione: L'utente può navigare nel sito per raggiungere la sua pagina personale.

Postcondizioni: L'utente ha visualizzato la sua pagina.

UC5.1: Modifica dati

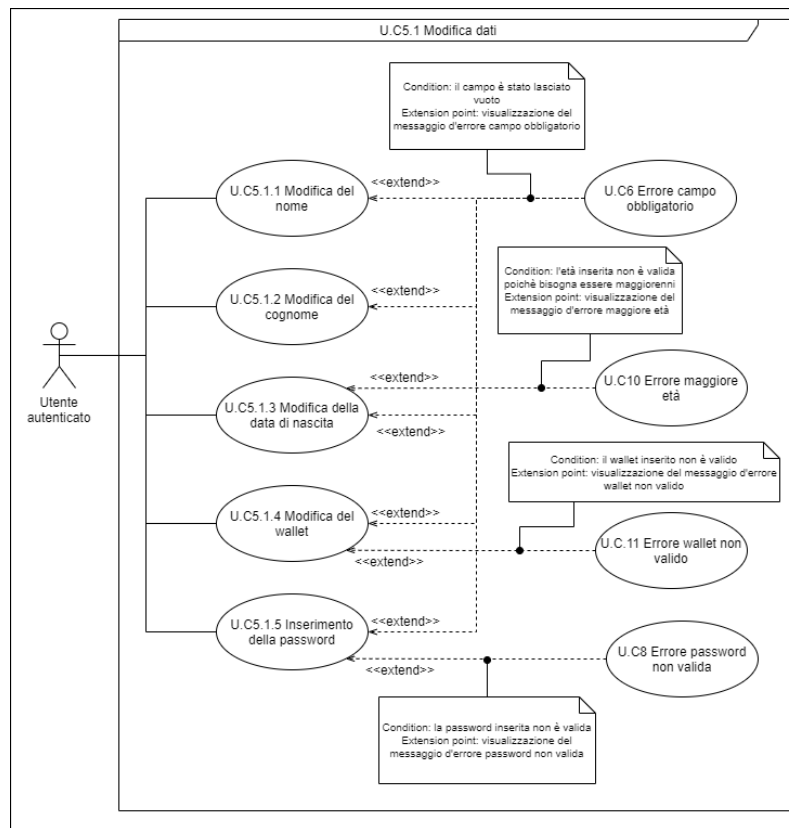


Figura 3.8: U.C5.1 Modifica dati

Attori Principali: Utente autenticato.

Precondizioni: L'utente si trova nella sua pagina personale.

Descrizione: L'utente può modificare i dati personali. In dettaglio l'utente può modificare:

- * il nome ([UC5.1.1](#));
- * il cognome ([UC5.1.2](#));
- * la data di nascita ([UC5.1.3](#));
- * l'address wallet ([UC5.1.4](#)).

Per poter confermare le modifiche l'utente deve inserire la propria password ([UC5.1.5](#))

Postcondizioni: L'utente ha modificato i suoi dati.

UC5.1.1: Modifica del nome

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e non ha modificato questo campo.

Descrizione: L'utente può modificare il proprio nome.

Estensioni: Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC6).

Postcondizioni: L'utente ha modificato il nome.

UC5.1.2: Modifica del cognome

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e non ha modificato questo campo.

Descrizione: L'utente può modificare il proprio cognome.

Estensioni: Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC6).

Postcondizioni: L'utente ha modificato il cognome.

UC5.1.3: Modifica della data di nascita

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e non ha modificato questo campo.

Descrizione: L'utente può modificare la propria data di nascita.

Estensioni:

- * Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC6);
- * Viene effettuato un controllo sul campo inserito e risulta non essere valido in quanto l'utente risulta essere non maggiorenne quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC10).

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

UC5.1.4: Modifica del wallet

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e non ha modificato questo campo.

Descrizione: L'utente può modificare il proprio address wallett.

Estensioni:

- * Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC6);
- * Viene effettuato un controllo sul campo inserito e risulta non essere valido quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC11).

Postcondizioni: L'utente ha modificato il proprio address wallett.

UC5.1.5: Inserimento della password

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e non ha compilato questo campo.

Descrizione: L'utente può inserire la propria password per procedere alla modifica dei dati.

Estensioni:

- * Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato ([UC6](#));
- * Viene effettuato un controllo sul campo inserito e risulta non essere valido quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato ([UC8](#)).

Postcondizioni: L'utente ha inserito la password.

UC5.2: Visualizzazione lista opere personali

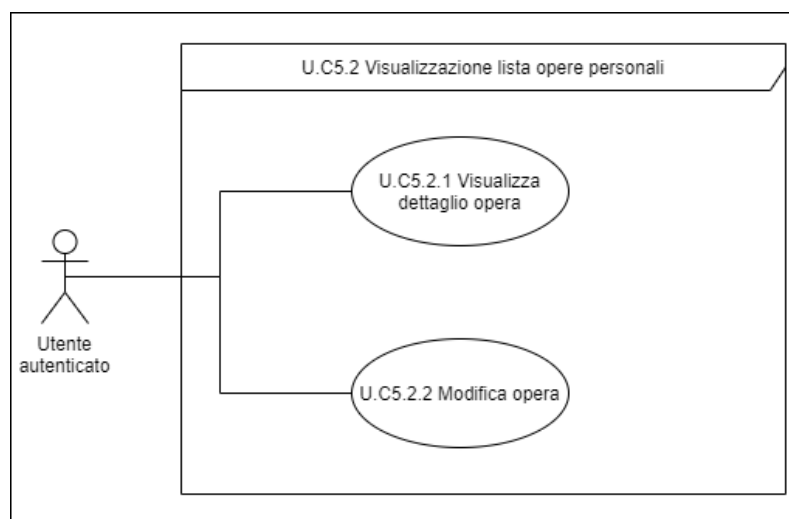


Figura 3.9: U.C5.2 Visualizzazione lista opere personali

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e si trova nella sua pagina personale.

Descrizione: L'utente può visualizzare la lista delle sue opere. Inoltre ha la possibilità di visualizzare in dettaglio i dati ([UC5.2.1](#)) oppure modificare i dati ([UC5.2.2](#)) un'opera

selezionata

Postcondizioni: L'utente visualizza le sue opere.

UC5.2.1: Visualizza dettaglio opera

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e si trova nella pagina personale e sta visualizzando la lista delle sue opere.

Descrizione: L'utente può selezionare un'opera per poter visualizzare le sue informazioni nel dettaglio.

Postcondizioni: L'utente ha visualizzato i dettagli dell'opera selezionata.

UC5.2.2: Modifica opera

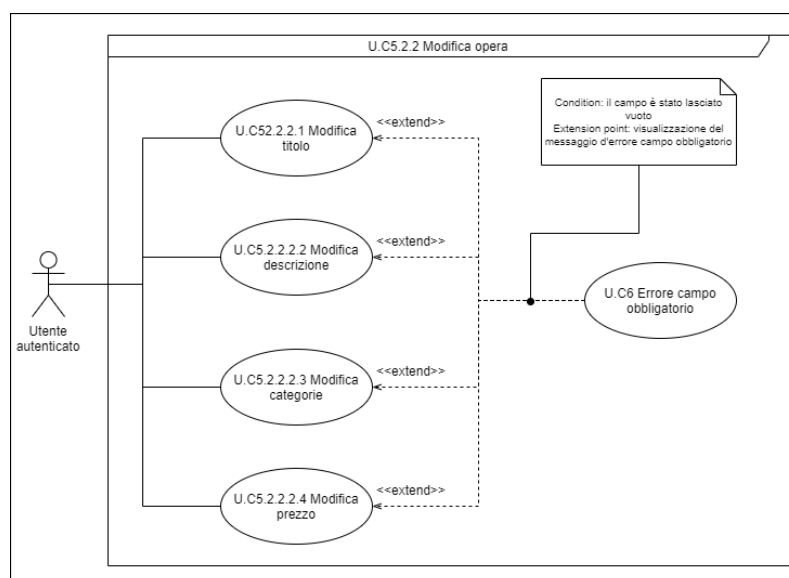


Figura 3.10: U.C5.2.2 Modifica opera

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e si trova nella pagina personale e sta visualizzando la lista delle sue opere.

Descrizione: L'utente può selezionare un'opera per poter modificare i suoi dati.

Postcondizioni: L'utente ha modificato i dati dell'opera.

UC5.2.2.1: Modifica titolo

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e non ha modificato questo campo.

Descrizione: L'utente può modificare il titolo dell'opera.

Estensioni: Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC6).

Postcondizioni: L'utente ha modificato il titolo dell'opera.

UC5.2.2.2: Modifica descrizione

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e non ha modificato questo campo.

Descrizione: L'utente può modificare la descrizione.

Estensioni: Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC6).

Postcondizioni: L'utente ha modificato la descrizione.

UC5.2.2.3: Modifica categorie

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e non ha modificato questo campo.

Descrizione: L'utente può modificare le categorie.

Estensioni: Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC6).

Postcondizioni: L'utente ha modificato le categorie.

UC5.2.2.4: Modifica prezzo

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e non ha modificato questo campo.

Descrizione: L'utente può modificare il prezzo.

Estensioni: Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC6).

Postcondizioni: L'utente ha modificato il prezzo.

UC5.3: Upload di una nuova opera

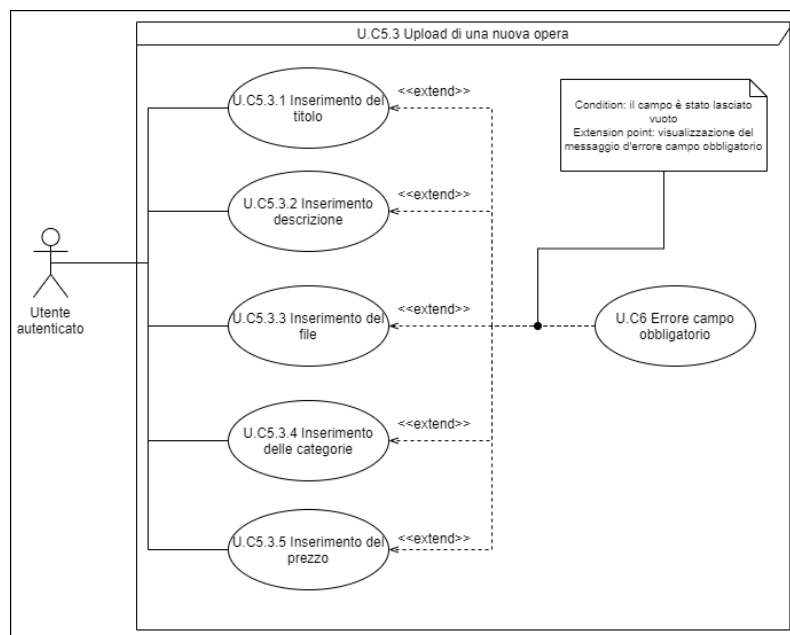


Figura 3.11: U.C5.3 Upload di una nuova opera

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e si trova nella schermata di caricamento dell'opera.

Descrizione: L'utente ha la possibilità di caricare una nuova opera.

Postcondizioni: L'utente ha caricato una nuova opera.

UC5.3.1: Inserimento del titolo

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e non ha compilato questo campo.

Descrizione: L'utente ha la possibilità di inserire il titolo dell'opera.

Estensioni: Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC6).

Postcondizioni: L'utente ha inserito il titolo dell'opera.

UC5.3.2: Inserimento della descrizione

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e non ha compilato questo campo.

Descrizione: L'utente ha la possibilità di inserire la descrizione dell'opera.

Estensioni: Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di

inserire nuovamente il dato (UC6).

Postcondizioni: L'utente ha inserito la descrizione.

UC5.3.3: Inserimento del file

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e non ha compilato questo campo.

Descrizione: L'utente ha la possibilità di inserire il file.

Estensioni: Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC6).

Postcondizioni: L'utente ha inserito il file.

UC5.3.4: Inserimento delle categorie

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e non ha compilato questo campo.

Descrizione: L'utente ha la possibilità di inserire le categorie.

Estensioni: Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC6).

Postcondizioni: L'utente ha inserito le categorie.

UC5.3.5: Inserimento del prezzo

Attori Principali: Utente autenticato.

Precondizioni: L'utente è autenticato e non ha compilato questo campo.

Descrizione: L'utente ha la possibilità di inserire il prezzo.

Estensioni: Viene effettuato un controllo e risulta che questo campo non è stato compilato quindi si presenta un messaggio di errore e viene fornita la possibilità di inserire nuovamente il dato (UC6).

Postcondizioni: L'utente ha inserito il prezzo.

UC6: Errore campo obbligatorio

Attori Principali: Utente autenticato o utente non autenticato.

Precondizioni: L'utente non ha inserito alcun carattere in un campo dati obbligatorio.

Descrizione: L'utente non ha inserito alcun carattere in un campo dati obbligatorio e viene informato del mancato riempimento del campo dati.

Postcondizioni: All'utente viene mostrato un messaggio che lo avvisa del mancato riempimento del campo dati obbligatorio.

UC7: Errore email non valida

Attori Principali: Utente autenticato o utente non autenticato.

Precondizioni: L'utente ha inserito l'email in un formato non valido.

Descrizione: L'utente ha inserito l'email in un formato non valido e viene informato del scorretto inserimento del campo dati.

Postcondizioni: All'utente viene mostrato un messaggio che lo avvisa dello scorretto formato del campo appena inserito.

UC8: Errore password non valida

Attori Principali: Utente autenticato o utente non autenticato.

Precondizioni: L'utente ha inserito una password in un formato non valido.

Descrizione: L'utente ha inserito la password in un formato non valido e viene informato del scorretto inserimento del campo dati.

Postcondizioni: All'utente viene mostrato un messaggio che lo avvisa dello scorretto formato del campo appena inserito.

UC9: Errore conferma password differente

Attori Principali: Utente autenticato o utente non autenticato.

Precondizioni: L'utente ha compilato

Descrizione: L'utente ha inserito la conferma della password differente dalla password precedentemente inserita e viene informato del scorretto inserimento del campo dati.

Postcondizioni: All'utente viene mostrato un messaggio che lo avvisa della differenza tra le due password inserite.

UC10: Errore maggiore età

Attori Principali: Utente autenticato o utente non autenticato.

Precondizioni: L'utente ha inserito una data di nascita non valida in quanto non risulta essere maggiorenne.

Descrizione: L'utente ha inserito una data di nascita non valida in quanto non risulta essere maggiorenne e viene informato del scorretto inserimento del campo dati.

Postcondizioni: All'utente viene mostrato un messaggio che lo avvisa dello scorretto formato del campo appena inserito.

UC11: Errore wallet non valido

Attori Principali: Utente autenticato o utente non autenticato.

Precondizioni: L'utente ha inserito un address wallet in un formato non valido.

Descrizione: L'utente ha inserito un address wallet in un formato non valido e viene informato del scorretto inserimento del campo dati.

Postcondizioni: All'utente viene mostrato un messaggio che lo avvisa dello scorretto formato del campo appena inserito.

UC12: Errore utente non presente nel database

Attori Principali: Utente autenticato o utente non autenticato.

Precondizioni: L'utente ha compilato correttamente i campi ed ha inviato la richiesta al database.

Descrizione: L'utente ha compilato correttamente i campi ed ha inviato la richiesta al database ma viene effettuato un controllo e risulta non essere presente nel database, quindi viene informato l'utente dell'errore.

Postcondizioni: All'utente viene mostrato un messaggio che lo avvisa della non corrispondenza nel database.

3.2 Tracciamento dei requisiti

Come risultato di un'attenta analisi dei requisiti e i relativi casi d'uso effettuata sul progetto sono stati individuati diversi requisiti. Questi sono stati suddivisi per classificazione e tipologia, per questo motivo si utilizza un codice identificativo per distinguerli che è così strutturato:

R[classificazione][tipologia][codice]

La descrizione del codice è la seguente:

* **R:** acronimo per Requisito;

* **classificazione:** individua la classificazione del requisito e può essere:

F = funzionale

Q = qualitativo

V = di vincolo

* **tipologia:** individua la tipologia del requisito e può essere:

O = obbligatorio

D = desiderabile

F = facoltativo

Nelle sezioni seguenti sono riassunti i requisiti ed il loro tracciamento con i casi d'uso delineati in fase di analisi.

Tabella 3.1: Tabella del tracciamento dei requisiti funzionali

Requisito	Descrizione	Fonte
RFO1	L'utente non autenticato può effettuare la registrazione al sito	UC1
RFO1.1	L'utente non autenticato inserisce i dati per la registrazione	UC1.1
RFO1.1.1	L'utente non autenticato inserisce il nome	UC1.1.1
RFO1.1.2	L'utente non autenticato inserisce il cognome	UC1.1.2
RFO1.1.3	L'utente non autenticato inserisce l'email	UC1.1.3
RFO1.1.4	L'utente non autenticato inserisce la password	UC1.1.4
RFO1.1.5	L'utente non autenticato inserisce la conferma della password	UC1.1.5
RFO1.1.6	L'utente non autenticato inserisce l'anno di nascita	UC1.1.6
RFO1.1.7	L'utente non autenticato inserisce il Wallett Address	UC1.1.7
RFO2	L'utente non autenticato può effettuare il login al sito	UC2
RFO2.1	L'utente non autenticato inserisce i dati per il login	UC2.1
RFO2.1.1	L'utente non autenticato inserisce l'email	UC2.1.1
RFO2.1.2	L'utente non autenticato inserisce la password	UC2.1.2
RFO3	L'utente autenticato può effettuare il logout	UC3
RFO4	L'utente autenticato o non autenticato può visualizzare la lista delle opere nella homepage	UC4
RFO4.1	L'utente autenticato o non autenticato può applicare dei filtri sulla lista delle opere	UC4.1
RFO4.1.1	L'utente autenticato o non autenticato può applicare dei filtri per categorie sulla lista delle opere	UC4.1.1
RFO4.2	L'utente autenticato o non autenticato può visualizzare in dettaglio l'opera selezionata	UC4.2
RFO5	L'utente autenticato può visualizzare la sua pagina personale	UC5
RFO5.1	L'utente autenticato può modificare i suoi dati	UC5.1
RFO5.1.1	L'utente autenticato può modificare il suo nome	UC5.1.1
RFO5.1.2	L'utente autenticato può modificare il suo cognome	UC5.1.2
RFO5.1.3	L'utente autenticato può modificare la sua data di nascita	UC5.1.3
RFO5.1.4	L'utente autenticato può modificare il suo address wallet	UC5.1.4
RFO5.1.5	L'utente autenticato può inserire la password per confermare le modifiche effettuate	UC5.1.5
RFO5.2	L'utente autenticato può visualizzare le sue opere personali	UC5.2
RFO5.2.1	L'utente autenticato può visualizzare in dettaglio l'opera selezionata	UC5.2.1

Tabella 3.2: Continuazione della tabella del tracciamento dei requisiti funzionali

RFO5.2.2	L'utente autenticato può modificare l'opera selezionata	UC5.2.2
RFO5.2.2.1	L'utente autenticato può modificare il titolo dell'opera	UC5.2.2.1
RFO5.2.2.2	L'utente autenticato può modificare la descrizione dell'opera	UC5.2.2.2
RFO5.2.2.3	L'utente autenticato può modificare le categorie dell'opera	UC5.2.2.3
RFO5.2.2.4	L'utente autenticato può modificare il prezzo dell'opera	UC5.2.2.4
RFO5.3	L'utente autenticato può caricare una nuova opera	UC5.3
RFO5.3.1	L'utente autenticato può inserire il titolo	UC5.3.1
RFO5.3.2	L'utente autenticato può inserire la descrizione	UC5.3.2
RFO5.3.3	L'utente autenticato può inserire il file	UC5.3.3
RFO5.3.4	L'utente autenticato può inserire le categorie	UC5.3.4
RFO5.3.5	L'utente autenticato può inserire il prezzo	UC5.3.5

Tabella 3.3: Tabella del tracciamento dei requisiti qualitativi

Requisito	Descrizione	Fonte
RQ01	Il codice sorgente prodotto deve essere disponibile in una repository pubblica su Github	Interna
RQ02	Deve essere prodotto un documento tecnico che spieghi il funzionamento del codice prodotto	Interna

Tabella 3.4: Tabella del tracciamento dei requisiti di vincolo

Requisito	Descrizione	Fonte
RVO1	Le maschere devono essere sviluppate tramite il framework Vue.js	Interna
RVO2	Le maschere devono essere sviluppate tramite il linguaggio javascript	Interna

Capitolo 4

Progettazione e codifica

In questo capitolo verranno spiegate le tecnologie utilizzate per lo sviluppo e l'architettura del prodotto software

4.1 Tecnologie e strumenti

Di seguito viene data una panoramica delle tecnologie e strumenti utilizzati per lo sviluppo delle maschere e per la collaborazione con i colleghi stagisti ed il tutor aziendale.

4.1.1 Tecnologie

4.1.1.1 Vue.js



Figura 4.1: Logo di Vue.js

Vue.js è un [framework](#) Javascript *open source* nato nel 2013 che presenta un'architettura adottabile in modo incrementale che si concentra sulla composizione dei componenti, inoltre sono presenti funzionalità avanzate offerte tramite librerie e pacchetti di supporto. I componenti Vue estendono gli elementi HTML di base per incapsulare del codice riutilizzabile quindi a livello generale i componenti sono elementi personalizzati a cui il compilatore Vue associa una particolare funzionalità.

Vue utilizza quindi una sintassi basata su HTML e consente di associare il *DOM* renderizzato ai dati dell'istanza di Vue sottostante. In questo modo i modelli Vue possono

essere analizzati da browser e *parser* HTML conformi alle modifiche ed inoltre con il sistema di reattività Vue è in grado di calcolare il numero minimo di componenti per eseguire nuovamente il *rendering* applicando la quantità minima di manipolazioni DOM quando cambia lo stato dell'app. Vue presenta un sistema reattivo grazie all'utilizzo di oggetti semplici Javascript ed ad un *re-rendering* ottimizzato: ogni componente durante il render tiene traccia delle sue dipendenze in modo tale che il sistema sappia quando e di quali componenti deve effettuare nuovamente il render.

Un problema che affligge le [web application](#) a pagina singola è che forniscono agli utenti la risposta basata solamente sull'URL dal server, di conseguenza l'utilizzo dei segnalibri a determinate schermate e la condivisione dei collegamenti a sezioni specifiche risulta molto difficile se non impossibile. Vue.js per riuscire a risolvere questo problema fornisce un'interfaccia, detta *router*, che modifica ciò che viene visualizzato sulla pagina in base all'URL corrente, indipendentemente da come è stato modificato. Infatti Vue.js viene fornito con il pacchetto *open source* "*vue-router*" che fornisce un'API per aggiornare l'URL dell'applicazione, supportare la cronologia di navigazione e le reimpostazioni di email e password. Tramite questa tipologia di *router* i componenti devono essere mappati alla *route* a cui appartengono per indicare dove deve essere eseguito il loro render.

Questo [framework](#) implementa il pattern MVVM, acronimo per *Model-View-View-Model*, una declinazione del più famoso MVC, ovvero *Model-View-Controller*. I componenti del MVVM sono:

- * *Model* (o Modello): l'implementazione del dominio dati come per il classico Modello del pattern MVC;
- * *View* (o Vista): il componente grafico renderizzato dall'utente formato da HTML e CSS;
- * *ViewModel* (o Vista per il Modello): il collante tra gli altri due componenti, esso fornisce alla *View* i dati in formato consono alla rappresentazione ed il comportamento di alcuni elementi dinamici.

La grossa differenza tra il pattern implementato da Vue.js e il *Model-View-Controller* sta nella differenza tra Controller e ViewModel. Il primo, infatti, è una porzione di codice che gestisce la logica di business grazie al Model e ritorna una *View* da mostrare all'utente; il secondo, invece, rappresenta una versione parallela al Model che risulta essere legato alla *View* e descrive il comportamento di quest'ultima con funzioni associate. Quindi mentre il *Controller* esegue logiche di business prima del *rendering* della *View*, il *ViewModel* definisce il comportamento dell'applicazione a *runtime*.

4.1.1.2 Vuetify

Vuetify è un [framework](#) UI completo costruito su Vue.js nel 2014 ed il suo obiettivo è fornire agli sviluppatori gli strumenti per poter creare esperienze utente ricche e coinvolgenti. A differenza di altri [framework](#) Vuetify è progettato da zero in modo tale da renderlo facile da imparare ed essere gratificante da padroneggiare con centinaia di componenti realizzate dalle specifiche di *Material Design*.

Un pregio di questo [framework](#) è che adotta un approccio al mobile, questo significa che la [web application](#) sviluppata tramite esso sarà pienamente utilizzabile immediatamente su un tablet, un telefono ed un computer. Inoltre è un [framework](#) in sviluppo attivo che viene aggiornato settimanalmente rispondendo ai problemi e relativi *report* della

community. Un altro pregio è, come si può vedere nell'immagine sottostante, il gran numero di funzionalità che possiede Vuetify in confronto agli altri [framework](#) di Vue.






Vue Framework Comparison 2021					
Features	 Vuetify	 BootstrapVue	 Buefy	 Element UI	 Quasar
Accessibility and section 508 support	●	●	●		
Business and enterprise support	●				
Long-term Support	●				
Release cadence**	Weekly	Bi-Weekly	Bi-Monthly	Bi-Weekly	Bi-Weekly
RTL support	●	●		●	●
Premium themes	●	●			
Treeshaking	Automatic	Manual	Manual	Manual	Automatic
**Based on average of all Major/Minor/Patch releases over the last 12 months.					

Figura 4.2: Funzionalità di Vuetify

4.1.1.3 Vuex

Vuex è una libreria per applicazioni sviluppate in Vue.js e serve come *store* centralizzato per tutti i componenti dell'applicazione. Esso contiene stati e regole che assicurano che lo stato può essere modificato solo in modo prevedibile.

Inoltre Vuex è uno *state management pattern* che aiuta a salvare i dati in maniera persistente per poterli utilizzare all'interno dell'applicazione da diversi componenti in maniera efficiente.

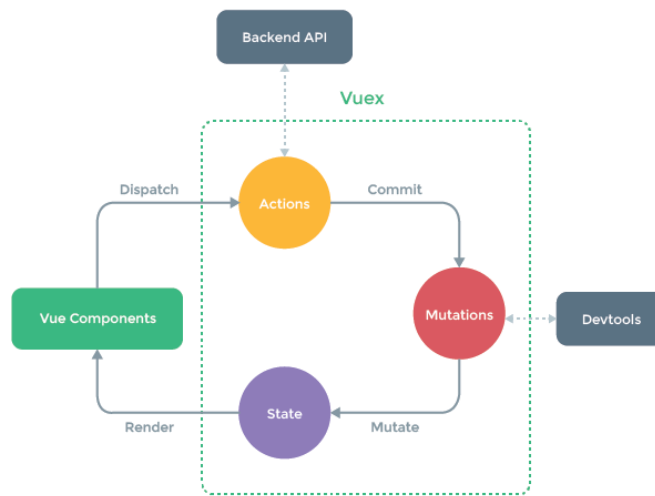


Figura 4.3: Pattern di Vuex

4.1.2 Strumenti

4.1.2.1 Github

Github è un servizio web e *cloud-based* fondato nel 2008 che aiuta gli sviluppatori ad archiviare, gestire il codice, tracciare e controllare le modifiche. I due argomenti principali legati a Github sono: controllo versioni e Git.

Il controllo versioni aiuta a tracciare e gestire le modifiche del codice di un progetto software: più un progetto risulta essere di grandi dimensioni più il controllo delle versioni diventa fondamentale. Infatti questo aiuta gli sviluppatori a lavorare con sicurezza attraverso due azioni:

- * *branching*: uno sviluppatore duplica parte del codice sorgente, detto *repository*, in modo da apportare modifiche in modo sicuro senza influenzare l'intero progetto;
- * *merging*: una volta che lo sviluppatore è certo di aver prodotto del codice funzionante può fondere quel codice nel quello sorgente e renderlo ufficiale.

In questo modo tutte le modifiche possono essere monitorate e, se necessario, ripristinate. Inoltre grazie a questi concetti è possibile lavorare sullo stesso progetto in diversi sviluppatori senza problemi di ripetizioni di codice o conflitti durante lo sviluppo.

Git è un sistema di controllo versioni distribuito realizzato nel 2005, ovvero l'intero codice base e la cronologia sono disponibili sul computer di ogni sviluppatore. In questo modo è possibile creare facilmente ramificazioni e fusioni.

Per organizzare il lavoro abbiamo creato un'*organization*, prodotto offerto da Github che rende più semplice la collaborazione su più progetti contemporaneamente. Infatti ogni stagista ha creato la propria *repository* all'interno di questo spazio comune in modo tale da lavorare con i colleghi contemporaneamente e dare la possibilità ai tutor aziendali di visionare il lavoro svolto.

4.1.2.2 Figma

Figma è uno strumento nato nel 2016 rivolto ai web designer che hanno bisogno di un tool per la progettazione di interfacce. I vantaggi offerti da questo strumento sono i seguenti:

- * accessibilità multiplatforma;
- * sistema di collaborazione in *real-time*;
- * utilizzo degli strumenti *responsive oriented* per una progettazione ottimale;
- * lavora in vettoriale.

Grazie a questo strumento io e gli altri stagisti legati all'ambito [front end](#) abbiamo potuto creare dei prototipi delle maschere che avremmo dovuto sviluppare.

4.1.2.3 Stoplight

Stoplight è una piattaforma di progettazione [Application Program Interface \(API\)](#) collaborativa che si integra perfettamente nei flussi di lavoro per consentire a chiunque lavori con le [API](#) di essere più produttive. Questa piattaforma si basa su tre principi guida fondamentali, in modo da essere responsabili, collaborativi e con intenti positivi. I principi sono i seguenti:

- * quando si vede un'opportunità per avere un impatto, bisogna coglierla;
- * bisogna confidare l'uno nell'altro per aggiungere valore e risolvere problemi attraverso il lavoro di squadra;
- * bisogna cercare di capire gli altri ascoltando, indagando e rispondendo con attenzione.

Grazie a questa piattaforma si può aiutare gli utenti interni ed esterni ad integrarsi rapidamente all'[API](#) di un altro utente pubblicando documentazione interattiva, tutorial ed esempi di codice sempre aggiornati.

Noi stagisti abbiamo utilizzato questo strumento per non far dipendere lo sviluppo del [front end](#) con quello del [back end](#) e quindi riuscire a sviluppare in parallelo le due parti ed effettuare l'integrazione tra le due solo dopo esserci assicurati di avere il lavoro quasi concluso e poterci concentrare sulla risoluzione di possibili problemi in integrazione.

4.1.2.4 Visual Studio Code

Visual Studio Code è un editor di codice sorgente sviluppato da *Microsoft* nel 2015. Esso può essere utilizzato con diversi linguaggi di programmazione, infatti incorpora in esso diverse funzioni che variano dal linguaggio con cui si sta programmando. Un punto di forza di questo editor è la sua grande malleabilità, infatti l'utente può installare o disinstallare diversi *plugin* in funzione del linguaggio che sta utilizzando.

Un altro punto di forza è la sua integrazione con Git facilitando il controllo di versione del lavoro svolto dal programmatore che ha la possibilità di controllare le modifiche fatte, risolvere eventuali conflitti e salvare il lavoro nella *repository*. Altri punti di forza di questo editor sono i seguenti:

- * *IntelliSense*, estensione che fornisce suggerimenti di completamento per variabili, metodi e moduli. In particolare per i metodi fornisce una breve spiegazione dei parametri che accettano e del loro funzionamento;

- * *Debug* semplificato grazie ai *plugin* messi a disposizione e all'interfaccia grafica intuitiva;
- * Modifica veloce delle righe di codice grazie ad avvisi su codice sospetto, possibilità di selezionare più righe contemporaneamente e suggerimenti sui parametri;
- * Navigazione e *refactoring* del codice semplice.

4.2 Organizzazione dei file

Quando si crea un progetto per la prima volta tramite il [framework](#) Vue.js vengono create di default delle cartelle e dei file per facilitare la progettazione del software. La suddivisione delle cartelle è rappresentata qui di seguito.

```

1  |-- front-end-vue
   |-- codice
   |   |-- .editorconfig
   |   |-- .eslintignore
   |   |-- .eslintrc.js
6  |   |-- .gitignore
   |   |-- .prettierrc.json
   |   |-- .prettierrc.json
   |   |-- babel.config.js
   |   |-- jest.config.js
11 |   |-- jsconfig.json
   |   |-- package-lock.json
   |   |-- package.json
   |   |-- README.md
   |   |-- vue.config.js
16 |   |-- .vscode
   |   |-- coverage
   |   |-- dist
   |   |-- public
   |   |-- src
21 |   |   |-- App.vue
   |   |   |-- main.js
   |   |   |-- style.css
   |   |   |-- assets
   |   |   |-- components
26 |   |   |-- plugins
   |   |   |-- router
   |   |   |-- store
   |   |   |-- index.js
   |   |   |-- modules
31 |   |   |-- view
   |   |-- tests
   |   |-- unit
   |-- docs
   |-- documento tecnico
36 |-- directoryList.md

```

Listing 4.1: Organizzazione dei file.

La cartella principale si chiama *front-end-vue* ed essa contiene i file di configurazione del progetto, il file *readme*, il file di *gitignore* ed il file *directoryList.md*, creato automaticamente attraverso il comando *mddir* dove viene scritta la struttura delle cartelle del progetto. Inoltre sono presenti tre cartelle:

- * *codice*, contenente il codice del progetto;

- * **docs**, contenente file YAML utilizzato per la configurazione di Stoplight;
- * **documento tecnico**, contenente i file latex utilizzati per la stesura di questo documento.

Per quanto riguarda la cartella legata al codice, essendo la più importante, necessita di una spiegazione degli elementi che la contengono. I file e le sottocartelle create automaticamente alla creazione del progetto però non verranno spiegate, mentre i restanti elementi sono i seguenti:

- * **coverage**: cartella che contiene i file e le cartelle create automaticamente quando vengono fatti partire i test di unità;
- * **src**: cartella in cui sono presenti file e cartelle necessari per il funzionamento del progetto e sono:
 - il file **App.vue** è la radice dell'applicazione definita nel formato Vue Component e di solito serve a definire il modello della pagina;
 - il file **Main.js** di tipo javascript che inizializza il file **App.vue** ed è responsabile dell'impostazione dei plugin e dei componenti di terze parti da utilizzare per lo sviluppo dell'applicazione;
 - la cartella **assets** contenente le immagini utilizzate all'interno del sito, come quella del logo;
 - la cartella **components** contenente la barra di navigazione (navbar) e le varie finestre d'avviso;
 - la cartella **plugins** contenente i file Javascript dei *plugin* utilizzati per lo sviluppo, in questo caso di Vuetify;
 - la cartella **router** contenente un omonimo file Javascript dove vengono definite le *route* ovvero le varie pagine in cui può navigare l'utente;
 - la cartella **store** contenente i file per la configurazione delle chiamate al database e lo state management;
 - la cartella **view** contenente le possibili schermate che può visualizzare l'utente;
 - la cartella **tests** che contiene la sottocartella **unit** dove sono presenti i test di unità effettuati.

Nelle sezioni di seguito verranno spiegate in dettaglio il funzionamento delle chiamate al [back end](#) e delle varie maschere implementate con le loro criticità

4.3 Chiamate al back end

Il file principale, **index.js**, contiene la configurazione di Vuex e del *persisted state* con l'inclusione dei moduli che sono di numero tanti quanti sono i servizi del [back end](#) che vanno ad invocare. Per rendere più chiaro ed efficiente il lavoro ho creato quindi il file **CurrentUser.js**, legato al servizio del [back end](#) dell'utente, e il file **NftService.js**, legato al servizio del [back end](#) degli NFT.

Entrambi i file hanno la struttura uguale e quello che cambia è il contenuto, infatti sono caratterizzati da quattro elementi costanti:

- * **state**, contenente le risorse che verranno utilizzate nei metodi;

- * *actions*, contenente i metodi dove vengono fatte le invocazioni ai servizi del [back end](#) tramite axios;
- * *mutations*, contenente i metodi che modificano le risorse, sono come un equivalente dei metodi set;
- * *getters*, contenente i metodi che ritornano i valori presenti nelle risorse.

Gli state del file *CurrentUser.js* li ho resi persistenti, in questo modo potranno essere utilizzati all'interno della [web application](#), per esempio dopo che l'utente ha eseguito l'autenticazione utilizzo i suoi dati per visualizzare il suo nome nel bottone che lo porta alla pagina personale. Queste informazioni possono essere utilizzate per fare dei controlli che decidono se reindirizzare o meno certe parti della [web application](#). Essendo legati a servizi diversi i metodi e le chiamate al [back end](#) dei due file sono molto diversi e nelle sezioni successive verranno spiegate nel dettaglio.

4.3.1 CurrentUser.js

In questo file ho implementato diversi servizi legati all'utente. I servizi in questione sono:

* Autenticazione:

```

loginUser({ commit }, user) {
  axios
    .post(urlBackend + 'login', {
4      email: user.email,
      password: user.password
    })
    .then(response => {
      commit('setUser', response.data);
      commit('setLoggedIn');
9      localStorage.setItem('user', JSON.stringify(response.data));
      router.push('/');
    })
    .catch(error => {
14      commit('setErrorMessageLog', error.response.status);
    });
}

```

Listing 4.2: Autenticazione.

Come si può vedere dallo *snippet* di codice soprastante viene eseguita una chiamata POST al database specificando i dati dell'utente che verranno inviati. Se la chiamata va a buon fine i dati dell'utente autenticato verranno salvati sia nello stato persistente che nel *localStorage* del web e si verrà reindirizzati alla homepage. Se invece la chiamata non va a buon fine viene effettuato il catch dell'errore e viene settato un messaggio di errore che può variare in base all'errore che ritorna il [back end](#).

* Registrazione:

```

signUp({ commit }, user) {
  axios
    .post(urlBackend + 'signup', {
4      email: user.email,
      password: user.password,
      name: user.name,

```

```

    surname: user.surname,
    dob: user.dob,
    wallet: user.wallet
9  })
    .then(response => {
        commit('setUser', response.data);
        commit('setLoggedIn');
14     localStorage.setItem('user', JSON.stringify(response.data));
        router.push('/');
    })
    .catch(error => {
19     commit('setMessageErrorSig', error.response.status);
    });
}

```

Listing 4.3: Registrazione.

Come si può vedere dallo *snippet* di codice soprastante viene eseguita anche per la registrazione una chiamata POST al database specificando i dati dell'utente che verranno inviati, di numero maggiore rispetto ai dati per l'autenticazione. Se la chiamata va a buon fine i dati dell'utente autenticato verranno salvati sia nello stato persistente che nel *localStorage* del web e si verrà reindirizzati alla homepage. Se invece la chiamata non va a buon fine viene effettuato il catch dell'errore e viene settato un messaggio di errore che può variare in base all'errore che ritorna il [back end](#).

* **Logout:**

```

logout({ commit }) {
    commit('setLoggedOut');
    localStorage.clear('user');
}

```

Listing 4.4: Logout.

Come si può vedere dallo *snippet* di codice soprastante questo è l'unico metodo che non possiede chiamate effettive al [back end](#), infatti quando l'utente vuole effettuare il log out verrà semplicemente chiamato un metodo che fa ritornare l'utente alla homepage e cancella i dati dell'utente nello stato persistente. Infine viene svuotato il *localStorage* dai dati dell'utente.

* **modifica dei dati personali**

```

1  updateUser({ commit }, user) {
    var url = urlStop + 'user/${JSON.parse(localStorage.getItem('
        user')).id}';
    console.log(user);
    axios
    .put(url, {
6      password: user.password,
        name: user.name,
        surname: user.surname,
        email: user.email,
        dob: user.dob,
11     wallet: user.wallet
    })
    .then(response => {
        commit('setUser', response.data);
        localStorage.setItem('user', JSON.stringify(response.data));
    });
}

```

```

16      }).catch(error =>{
          commit('setErrorMessageMod', error.response.status);
        })
    }

```

Listing 4.5: Modifica dei dati personali.

Come si può vedere dallo *snippet* di codice soprastante viene eseguita la modifica dei dati personali tramite una chiamata PUT al database specificando i dati dell'utente che verranno inviati e inviando come parametro l'id identificativo dell'utente. Se la chiamata va a buon fine i dati aggiornati verranno salvati sia nello stato persistente che nel *localStorage* del web. Se invece la chiamata non va a buon fine viene effettuato il catch dell'errore e viene settato un messaggio di errore che può variare in base all'errore che ritorna dal [back end](#).

* **Modifica della password:**

```

1      updatePassword({}, user) {
        axios.put(urlStop + 'user/password', {
          email: user.email,
          oldPassword: user.oldPassword,
          newPassword: user.newPassword
6      }).catch(error =>{
          commit('setErrorMessageMod', error.response.status);
        })
    }

```

Listing 4.6: Modifica della password.

Come si può vedere dall'immagine soprastante viene eseguita la modifica della password tramite una chiamata PUT al database. Essendo un dato che non deve essere utilizzato all'interno del sito per questioni di sicurezza, il database non ritorna alcuna risposta. Se invece la chiamata non va a buon fine viene effettuato il catch dell'errore e viene settato un messaggio di errore che può variare in base all'errore che ritorna dal [back end](#).

4.3.2 NftService.js

In questo file ho implementato diversi servizi legati agli NFT. I servizi in questione sono:

* **Opere dell'utente:**

```

1      userOperas({ commit }) {
        var url =
          urlBackEnd + 'nft/user/${JSON.parse(localStorage.getItem('user
          ')).id}';
        axios.get(url).then(response => {
          commit('setOperas', response.data);
6      });
    }

```

Listing 4.7: Opere dell'utente.

Come si può vedere dallo *snippet* di codice soprastante questo metodo serve per ritornare le opere dell'utente. Questo avviene tramite una chiamata GET al database fornendo come parametro l'id identificativo dell'utente.

* **Categorie disponibili:**

```

    getCategories({ commit }) {
      axios.get(urlBackend + 'categories'+examples).then(response =>
3      {
        commit('setCategories', response.data);
      });
    }

```

Listing 4.8: Categorie disponibili.

Come si può vedere dallo *snippet* di codice soprastante questo metodo serve per ritornare le categorie disponibili nel database tramite una chiamata GET al database. Questo metodo viene invocato ogni qual volta serve sapere le categorie presenti, quindi ad esempio per caricare una nuova opera, per modificarne una o per la creazione del filtro presente nella homepage.

* **Caricamento di una nuova opera:**

```

    uploadOpera({ commit }, opera) {
      var url =
      urlBackend + 'nft/user/${JSON.parse(localStorage.getItem('user
      ')).id}';
      axios
5      .post(url, opera)
      .then(response => {
        commit('setOpera', response.data);
        router.push('/');
      })
10      .catch(error => {
        commit('setErrorOpera', error.response.data);
      });
    }

```

Listing 4.9: Caricamento di una nuova opera.

Come si può vedere dallo *snippet* di codice soprastante questo metodo serve per caricare una nuova opera nel database. Questo avviene tramite una chiamata POST al database fornendo come dati le informazioni della nuova opera e come parametro l'id identificativo dell'utente. Se invece la chiamata non va a buon fine viene effettuato il catch dell'errore e viene settato un messaggio di errore che può variare in base all'errore che ritorna dal [back end](#).

* **Modifica di un'opera esistente:**

```

    updateOpera({ commit }, opera) {
2      var url =
      urlStop + 'nft/user/${JSON.parse(localStorage.getItem('user'))
      .id}';
      axios
      .put(url, {
        id: opera.id,
7        description: opera.description,
        title: opera.title,
        price: Number(opera.price),
        currency: 'ETH',
        type: opera.type,
12        author: opera.author,
        owner: opera.owner,
        categories: opera.categories,

```

```

    path: opera.path
  })
17  .then(response => {
    commit('setOpera', response.data);
  })
  .catch(error => {
22    commit('setErrorOpera', error.response.data);
  });
}

```

Listing 4.10: Modifica di un'opera esistente.

Come si può vedere dallo *snippet* di codice soprastante questo metodo serve per modificare un'opera già presente nel database. Questo avviene tramite una chiamata PUT al database fornendo come dati le nuove informazioni dell'opera e come parametro l'id identificativo dell'utente. Se invece la chiamata non va a buon fine viene effettuato il catch dell'errore e viene settato un messaggio di errore che può variare in base all'errore che ritorna dal [back end](#).

* **Visualizzazione delle opere nella home:**

```

2  getHomeOperas({ commit }) {
    axios.get(urlBackEnd + 'nft'+examples).then(response => {
      commit('setHomeOperas', response.data);
    });
  }

```

Listing 4.11: Visualizzazione delle opere nella home.

Come si può vedere dallo *snippet* di codice soprastante questo metodo serve per ottenere tutte le opere presenti nel database per mostrarle nella homepage. Questo avviene tramite una chiamata GET al database.

4.4 Maschere implementate

Per comprendere meglio il funzionamento delle maschere queste verranno spiegate nelle sezioni sottostanti nel dettaglio.

4.4.1 Barra di navigazione

Questa componente è presente in quasi tutto il sito ad eccezione delle pagine di Login, di Registrazione e di Upload dell'opera. Per poter implementare questo dettaglio il componente che ha il compito di reindirizzarla, ovvero il file **App.vue**, possiede una condizione dove viene fatto il controllo se la pagina in cui si sta navigando sia una di quelle dove deve essere presente o meno e di conseguenza verrà reindirizzata o meno. Se il controllo va a buon fine la Navbar verrà reindirizzata, altrimenti non verrà mostrata all'utente.

Un'altra particolarità di questo componente è la presenza di due bottoni che variano in funzione se l'utente è autenticato o meno nel sito. Infatti se l'utente non si autentica si vedranno due bottoni per effettuare il login o la registrazione, mentre se esso è autenticato i bottoni serviranno per visualizzare la pagina personale oppure effettuare il logout. Per gestire questa particolarità è stato implementato un controllo condizionale che controlla appunto lo stato dell'utente tramite una variabile chiamata *isLogged*, che assume il valore a vero se autenticato.

4.4.2 Home

Questa è la pagina iniziale tramite la quale l'utente può visualizzare tutte le opere presenti nel database con integrata la paginazione, inoltre è presente una combobox che dà la possibilità di filtrare le opere per categoria. Infine è presente un bottone "Visualizza dettagli" che ha una funzionalità che verrà spiegata in seguito.

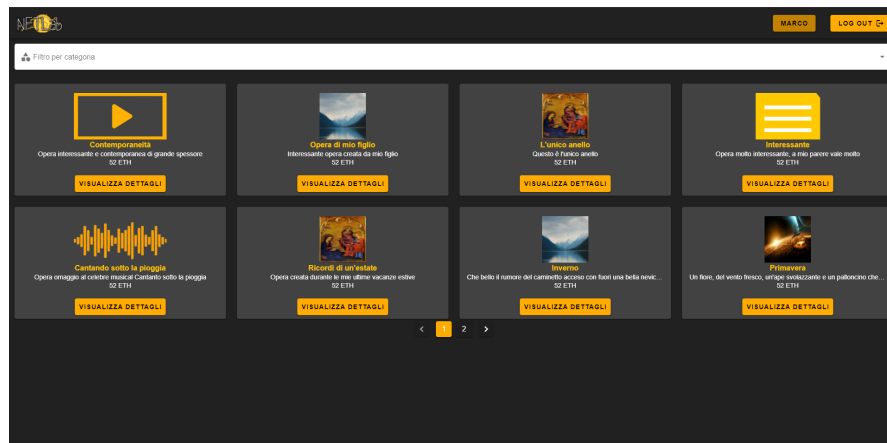


Figura 4.4: Home Page

4.4.3 Visualizza dettagli

Questa funzionalità fa apparire una finestra di dialogo a schermo intero che mostra tutte le informazioni dettagliate dell'opera selezionata. Nella parte sinistra l'utente potrà vedere una preview dell'opera, mentre nella parte destra l'utente potrà vedere: titolo, prezzo con valuta, autore, proprietario e categorie di appartenenza.

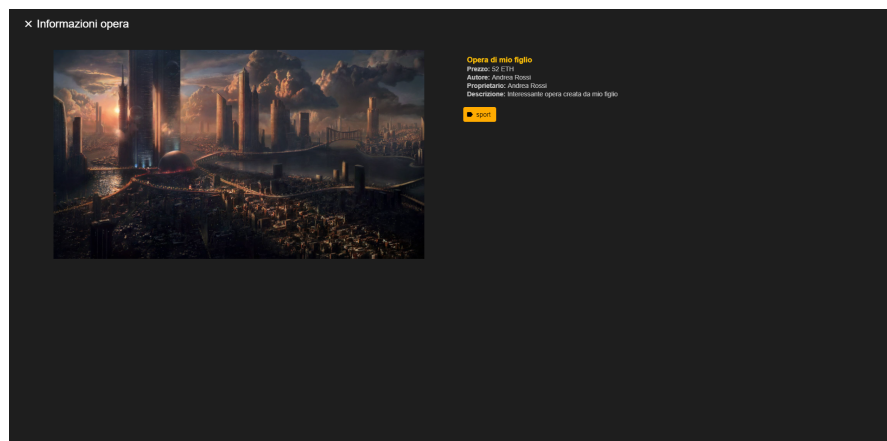


Figura 4.5: Finestra di dialogo delle informazioni dell'opera

Questa finestra di dialogo è stata progettata a schermo intero in modo tale da poter aggiungere ulteriori informazioni o funzionalità senza dover modificare la struttura della finestra di dialogo.

4.4.4 Login

In questa pagina l'utente può effettuare l'autenticazione nel sito tramite email e password. In questi due campi sono state inserite delle regole che, se non rispettate, generano un errore visualizzabile tramite una scritta sotto il campo che si sta modificando. Le regole in questione sono:

- * entrambi i campi sono obbligatori, quindi devono essere tutti compilati;
- * l'email inserita deve essere valida
esempio di email valida: nome@email.it;
- * la password deve avere una lunghezza di minimo otto caratteri di cui almeno una lettera maiuscola, una minuscola, un numero e una carattere speciale.

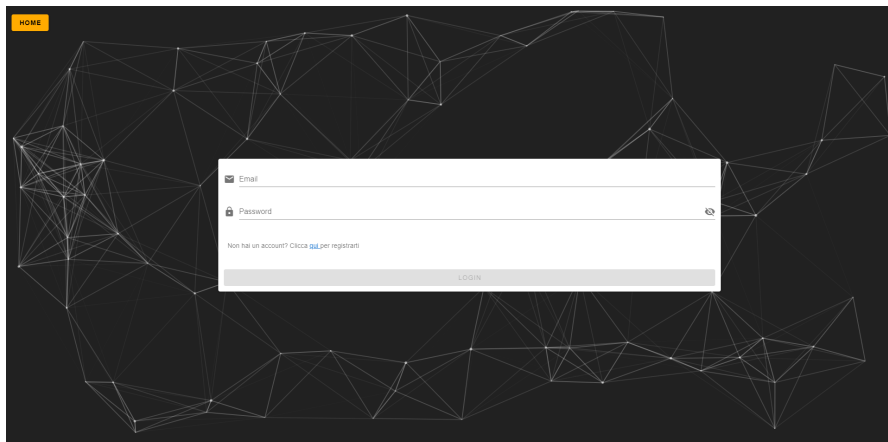


Figura 4.6: Pagina di login

In alto a sinistra è presente un bottone per tornare alla home. Inoltre il bottone per validare l'autenticazione è disabilitato fino a quando l'utente non compila entrambi i campi per evitare chiamate al [back end](#) con dati mancanti.

4.4.5 Registrazione

In questa pagine l'utente può effettuare la registrazione nel sito inserendo: nome, cognome, email, data di nascita, wallet address, password e conferma password. Similmente alla pagina di login sono state inserite delle regole che, se non rispettate, generano un errore visualizzabile tramite una scritta sotto il campo che si sta modificando. Le regole in questione sono:

- * tutti i campi sono obbligatori, quindi devono essere tutti compilati;
- * l'email inserita deve essere valida
esempio di email valida: nome@email.it;
- * la data di nascita deve corrispondere ad una persona maggiorenne;
- * il *wallett address* inserito deve essere valido
esempio di wallett address valido: 0x390c7a1EA64e521B725b1869Ea054DDBF05F9abe;

- * la password deve avere una lunghezza di minimo otto caratteri di cui almeno una lettera maiuscola, una minuscola, un numero e una carattere speciale;
- * la conferma della password deve essere identica alla password appena inserita.

Figura 4.7: Pagina di registrazione

Come per la pagina di autenticazione, in alto a sinistra è presente un bottone per tornare alla home. Inoltre il bottone per validare l'autenticazione è disabilitato fino a quando l'utente non compila entrambi i campi per evitare chiamate al [back end](#) con dati mancanti.

4.4.6 Pagina dell'utente

In questa pagina l'utente può visualizzare le sue informazioni personali e le opere che possiede. Nel momento che verranno modificate le informazioni da parte dell'utente, queste verranno modificate a schermo automaticamente.

Figura 4.8: Pagina personale dell'utente

4.4.7 Upload opera

In questa pagina è possibile per l'utente aggiungere una nuova opera. I campi da compilare sono: titolo, descrizione, file, prezzo. Quando l'utente carica un file verranno generati automaticamente una *preview* del file specifica in base al tipo del file inserito e un campo non modificabile che indica il tipo del file inserito. Similmente alla pagina di registrazione sono state inserite delle regole che, se non rispettate, generano un errore visualizzabile tramite una scritta sotto il campo che si sta modificando. Le regole in questione sono:

- * tutti i campi sono obbligatori, quindi devono essere tutti compilati.

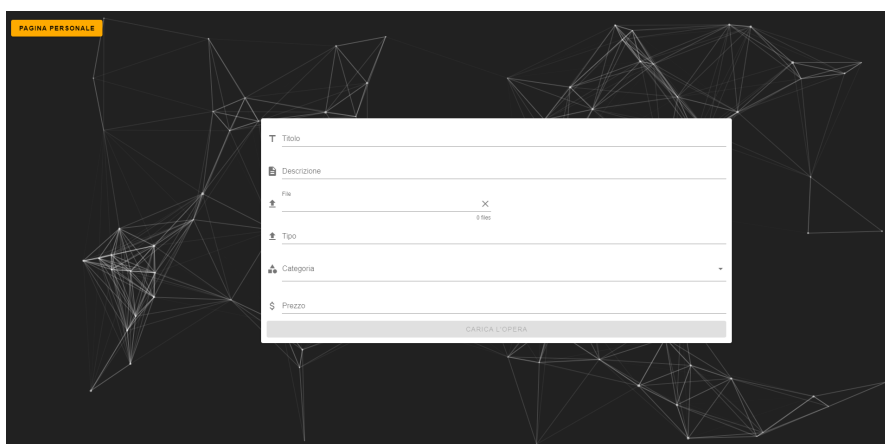
The image shows a web interface for uploading a work. At the top left, there is a yellow tab labeled 'PAGINA PERSONALE'. The background is dark with a white geometric wireframe pattern. A white form is centered on the screen. The form has the following fields: 'Titolo' (Title) with a 'T' icon, 'Descrizione' (Description) with a 'D' icon, 'File' with a file icon, a preview of the uploaded file, and an 'X' icon to remove it; 'Tipo' (Type) with a 'T' icon; 'Categoria' (Category) with a category icon and a dropdown arrow; and 'Prezzo' (Price) with a '\$' icon. At the bottom of the form is a button labeled 'CARICA OPERA'.

Figura 4.9: Pagina di upload dell'opera

Nel momento in cui l'utente carica il file verrà aggiornato automaticamente il campo legato alla tipologia di file. Questo campo non 'e modificabile da parte dell'utente.

4.4.8 Gestione opere

L'utente può unicamente modificare una sua opera ma non può cancellarla. Questo è dovuto al fatto che al momento dell'upload l'opera verrà caricata sulla [blockchain](#) rendendola un elemento unico ed irripetibile quindi non può cancellarla. Per motivi analoghi l'utente ha delle restrizioni anche nella modifica dell'opera; infatti di essa può modificare solo: titolo, descrizione, prezzo e categorie di appartenenza. Un altro campo non modificabile è il tipo del file caricato.

× Modifica opera

Titolo
Contemporaneità

Descrizione
Opera interessante e contemporanea di grande spessore

Tipo
video

Categoria
food sport

Prezzo
52

CONFERMA MODIFICHE

Figura 4.10: Finestra di dialogo per la modifica dell'opera

4.4.9 Modifica dati personali

L'utente, dopo aver premuto un bottone con la dicitura "Modifica dati personali", vedrà apparire una finestra di avviso dove può modificare i propri dati. Per motivi di univocità l'utente non potrà modificare l'email con cui ha fatto l'accesso, infatti potrà modificare solamente il nome ed il cognome. Inoltre per motivi di sicurezza il bottone per confermare la modifica è disabilitato di default e viene data la possibilità di confermare la modifica solo dopo aver inserito la password corrente.

× Modifica dati

Nome
Marco

Cognome
Rossi

Anno di nascita
1983-06-22

Wallet Address
0x3d62f66Cb1eA47fB7BC6E2f12860f9A9Cbc54622

Inserisci la password per confermare la modifica

CONFERMA MODIFICHE

Figura 4.11: Finestra di dialogo per la modifica dei dati

4.4.10 Modifica password

L'utente dopo aver premuto un bottone con la dicitura "Modifica password", vedrà apparire una finestra di avviso dove può modificare la password. Per poterlo fare, per motivi di sicurezza, l'utente dovrà inserire la sua email, la password corrente e la nuova password. Similmente alla modifica dei dati personali, il bottone per confermare la modifica rimarrà disattivato fino a quando l'utente non avrà completato tutti i campi.

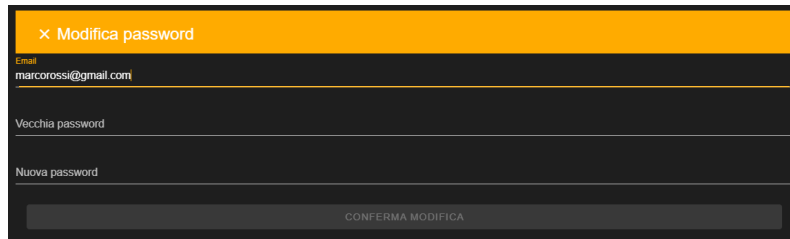
A screenshot of a web application dialog box titled "Modifica password" with a close button (X) in the top left corner. The dialog has a dark background with light-colored text. It contains three input fields: "Email" with the value "marcorossi@gmail.com", "Vecchia password", and "Nuova password". At the bottom, there is a button labeled "CONFERMA MODIFICA".

Figura 4.12: Finestra di dialogo per la modifica della password

4.5 Generazione della preview

Per quanto riguarda le immagini presenti nella homepage, nella pagina dell'utente e nelle finestre di dialogo che mostrano i dettagli dell'opera, viene generata una *preview* in base alla tipologia del file. Infatti se il tipo del file è un'immagine si potrà visualizzare una *preview* effettiva di essa, altrimenti se il tipo del file è audio, video o documento verranno visualizzate delle immagini preimpostate che fanno intuire la tipologia del file.

Questo ragionamento viene applicato anche quando l'utente carica una nuova opera. Infatti nel momento in cui l'utente seleziona un file verrà mostrata una *preview* generata utilizzando il ragionamento appena descritto.

Capitolo 5

Verifica e validazione

In questo capitolo verranno descritti gli strumenti e i processi utilizzati per la verifica del codice.

5.1 Strumenti per la verifica

Per verificare il codice prodotto ho utilizzato la libreria ufficiale di test *Vue Test Utils*. Grazie a questa libreria si possono testare le interazioni dell'utente, come un click di un bottone, le chiamate al [back end](#), le *routes* e l'utilizzo di *Vuex*.

5.2 Verifica

Per ogni componente ho creato un file che, nella fase di *run*, viene chiamato *test suit*. Ogni *text suit* può contenere al suo interno diversi test.

```
describe('login.vue', () => {  
  let vuetify;  
  beforeEach(() => {  
    vuetify = new Vuetify();  
  });  
  5 const wrapper = shallowMount(login, {  
    localVue,  
    store,  
    vuetify  
  10  });  
  it('Check if content render', () => {  
    expect(wrapper.contains('v-container')).toBe(true);  
  });  
  it('Check if button is disabled with empty fields', () => {  
    15 const email="";  
    const password="";  
  
    var emailInput = wrapper.find('#emailInput');  
    emailInput.element.value = email;  
    20 var passwordInput = wrapper.find('#passwordInput');  
    passwordInput.element.value = password;  
  
    expect(wrapper.vm.isFormValid).toBeFalsy();  
    expect(wrapper.find('v-btn').element.hasAttribute('disabled')).not  
      .toBe(true);  
    25  });  
});
```

```
30  it('Check if email is changed after user interaction', () =>{
    var emailInput = wrapper.find('#emailInput');
    wrapper.find('#emailInput').trigger('click');
    wrapper.find('#emailInput').trigger('input');
    expect(emailInput.value).not.toBe('');
  });
35  it('Check if old password is changed after user interaction', () =>{
    var passwordInput = wrapper.find('#passwordInput');
    wrapper.find('#passwordInput').trigger('click');
    wrapper.find('#passwordInput').trigger('input');
    expect(passwordInput.value).not.toBe('');
  });
});
```

Listing 5.1: Esempio di test-Pagina di login.

Prendiamo come esempio lo *snippet* di codice soprastante che è il *test suit* per la pagina di login. Per prima cosa viene effettuato il *mock* del componente tramite il comando *shallowMount* dove vengono anche specificati i vari *plugin* e lo *store* utilizzati. Ogni *it* presente nel codice rappresenta un singolo test.

Il primo test che ho effettuato è il controllo che la pagina renderizzi correttamente i suoi contenuti. I controlli successivi che ho effettuato sono invece legati all'interazione dell'utente con gli elementi presenti nella pagina. I controlli che eseguo sono ad esempio che il bottone sia disabilitato se l'utente non compila correttamente tutti i campi della *form*, oppure che i campi della *form* vengano compilati dopo l'interazione dell'utente. Per ogni componente ho effettuato test analoghi a quello appena descritto.

5.3 Validazione e collaudo

Infine durante le ultime settimane ho effettuato in collaborazione con un collega stagista l'integrazione al *back end* risolvendo possibili bug dovuti alle incongruenze tra il *back end* effettivo e quello simulato tramite *stoplight*.

Inoltre ho prodotto il documento tecnico richiesto dall'azienda riguardante il mio lavoro in modo che chiunque arrivi a prendere in mano il mio codice possa capire la logica che regge il mio lavoro.

Infine io e gli altri stagisti che hanno svolto lo stage nello stesso periodo abbiamo preparato una breve presentazione con demo effettiva del lavoro svolto da mostrare l'ultimo giorno a tutti i dipendenti disponibili dell'azienda.

Capitolo 6

Nozioni apprese

In questo capitolo verranno elencate e spiegate brevemente le nozioni apprese durante questo percorso di stage.

6.1 Organizzazione dello studio

Come precedentemente spiegato le ore complessive dello stage possono essere considerate suddivise in 3 periodi:

- * periodo di studio della durata di circa 160 ore;
- * periodo di sviluppo della durata di circa 120 ore;
- * periodo di convalidazione finale della durata di circa 40 ore.

Le prime nozioni mi sono maggiormente servite per poter sviluppare le maschere, mentre le seconde mi sono servite per conoscenza generale e migliore collaborazione con i colleghi addetti al reparto [back end](#).

6.2 Nozioni principali

6.2.1 Javascript

Javascript è un linguaggio di programmazione orientato agli oggetti e agli eventi comunemente utilizzato per la programmazione Web lato client. La sua enorme diffusione è dovuta al gran numero di librerie nate per semplificare la programmazione sul browser, dalla nascita di [framework](#) lato server e il mondo mobile che lo supporta come linguaggio principale.

La caratteristica principale di Javascript è quella di essere il linguaggio di *scripting* per eccellenza e questo comporta una serie di vantaggi e svantaggi secondo l'uso che se ne deve fare e considerando il rapporto che si instaura nel meccanismo client-server. Il server invia al *client* i dati che possono arrivare in due diversi formati (testo e binario), ma il *client* sa comprendere solo il formato binario, per cui se arrivano dati in quel formato diventeranno immediatamente eseguibili altrimenti il *client* dovrà interpretare i dati. Quindi il *client* ha bisogno di un interprete che converte i dati in formato binario.

Sono presenti numerosi vantaggi e svantaggi di questo linguaggio, più importanti da sapere sono i seguenti:

* **Vantaggi:**

- Velocità: è un linguaggio veloce, siccome ogni funzione può essere avviata immediatamente invece di dover contattare il server ed aspettare una sua risposta;
- Semplicità: è un linguaggio relativamente semplice da imparare ed implementare;
- Versatilità: è un linguaggio che lavora bene con altri linguaggi e può essere usato in una grande varietà di applicazioni, infatti può essere inserito in una pagina web indipendentemente dall'estensione del file e può essere utilizzato all'interno di script scritti in altri linguaggi;
- Carico del server: essendo Javascript un linguaggio lato *client* questo fa ridurre le domande al server del sito web.

* **Svantaggi:**

- Sicurezza: visto che il codice si esegue nel computer dell'utente può essere sfruttato per scopi dannosi;
- Affidamento all'utente finale: gli script lato server producono sempre lo stesso risultato ma gli *script* lato *client* possono risultare imprevedibili perché possono essere interpretati in maniera differente in base al browser utilizzato. Questo difetto però può essere mitigato provando il proprio codice sui browser più utilizzati dagli utenti.

6.2.2 Vue.js

Vue.js è un [framework](#) Javascript *open-source* per la configurazione di interfacce utente e *single-page application*.

Questo framework è stato spiegato nel dettaglio nel capitolo [4](#).

6.3 Nozioni secondarie

6.3.1 Java

Java è un linguaggio di programmazione ad alto livello, orientato agli oggetti e a tipizzazione statica che si appoggia sull'omonima piattaforma software di esecuzione. Java nasce dall'esigenza di risolvere due problemi concreti comuni tra i linguaggi orientati ad oggetti: garantire maggiore semplicità nella scrittura e gestione del codice, e permettere la realizzazione di programmi slegati da un'architettura precisa.

Il primo problema fu risolto utilizzando un sistema basato sulla gestione della memoria detto *garbage collector*, liberando così il programmatore dall'onere della gestione della memoria. Il secondo problema, invece, fu risolto facendo in modo che i programmi non fossero compilati in codice macchina ma in una sorta di codice intermedio che dovrà poi essere eseguito non dall'hardware direttamente ma dalla macchina virtuale detta JVM, ovvero *Java Virtual Machine*.

6.3.2 Concetti web

Riguardo a questo argomento sono stati ripassati in dettaglio due argomenti: il concetto di Servlet e il concetto di REST.

6.3.2.1 Servlet

Una Servlet è un oggetto scritto in linguaggio Java in grado di gestire le richieste generate da uno o più client attraverso scambi di messaggi tra il server ed i client stessi. La Servlet può utilizzare le Java API per implementare le diverse funzionalità e le specifiche Servlet API per mettere a disposizione un'interfaccia standard per gestire le comunicazioni tra il *web client* e la Servlet. E' importante sapere che le Servlet non hanno delle GUI.

I vantaggi della Servlet sono:

- * *efficienza*: la Servlet viene istanziata e caricata una sola volta, alla prima invocazione, mentre le chiamate successive sono gestite chiamando nuovi *thread*;
- * *portabilità*: le Servlet possono essere facilmente programmate e "portate" in diverse piattaforme;
- * *persistenza*: dopo essere stata caricata in memoria la Servlet rimane anche nelle successive richieste;
- * *gestione delle sessioni*: grazie alle Servlet si riesce a superare la limitazione dei protocolli HTTP senza stati.

6.3.2.2 REST

REST è un insieme di principi architetturali per la progettazione che rendono il Web adatto a realizzare *Web Service*. I principi sono:

- * *identificazione delle risorse*: per risorsa si intende qualsiasi elemento in oggetto di elaborazione, ovvero qualsiasi oggetto su cui è possibile effettuare operazioni. Ciascuna risorsa deve essere identificata univocamente, il meccanismo più naturale è il concetto di URI;
- * *uso esplicito dei metodi HTTP*: serve un meccanismo per indicare le operazioni che si possono fare sulle risorse, per questo motivo si sfruttano i metodi predefiniti del protocollo HTTP;
- * *risorse autodescrittive*: è opportuno usare i formati più standard possibili per semplificare l'interazione con il *client*. Il tipo di rappresentazione è indicato nella risposta HTTP;
- * *collegamenti tra risorse*: le risorse devono essere messe tra di loro in relazione tramite link ipertestuali, è un principio detto *HATEOAS*;
- * *comunicazione senza stato*: è un principio secondo il quale una richiesta non ha alcuna relazione con le richieste precedenti e successive.

6.3.3 Spring

Il [framework](#) Spring è una piattaforma Java che fornisce un'infrastruttura di supporto per sviluppatori in Java in modo tale che loro si occupino dalla parte dell'applicazione. I benefici di Spring sono:

- * dipendenze esplicite ed evidenti grazie alla *Dependency Injection*;
- * i contenitori legati all'*Inversion of Control* tendono ad essere leggeri;
- * non reinventa nulla, prende quello che è già esistente e lo rende disponibile;
- * è organizzato in modo modulare quindi il programmatore si preoccupa del pacchetto che gli interessa;
- * è un *model view controller framework* ben formato;
- * fornisce un'interfaccia di gestione delle transizioni.

Di questo [framework](#) in particolare ho studiato: Spring Boot, Spring Data, Spring Data JPA e Spring Data REST. Nelle sezioni successive verranno spiegati questi concetti.

6.3.3.1 Spring Boot

Spring Boot è un progetto Spring che ha lo scopo di rendere più semplice lo sviluppo e l'esecuzione delle applicazioni Spring. Un'applicazione richiede molti *metadati* per la configurazione e può risultare pesante, Spring Boot invece alleggerisce questo meccanismo fornendo una configurazione automatica. Infatti esso è basato su opinioni e convenzioni proprie per avere una configurazione minima con la possibilità di riscriverle se necessario. Alcune caratteristiche principali sono legate ai seguenti aspetti:

- * *starter dependencies*, ovvero configurazione automatica delle librerie e delle dipendenze;
- * configurazione automatica di *bean* e componenti e delle loro relazioni;
- * *actuator*, per ispezionare un'applicazione in esecuzione.

Spring Boot quindi semplifica la gestione delle dipendenze. La classe detta *Spring Boot Application* esegue l'avvio dell'applicazione etichettando la classe di configurazione, abilitando la scansione e l'identificazione automatica dei componenti, ed infine creando i componenti mancanti necessari alla configurazione dell'applicazione.

Per riassumere i vantaggi offerti da Spring Boot sono:

- * possibilità di incorporare applicazioni web server per cui non è necessario l'uso di file WAR, acronimo di *Web Application Archive*;
- * configurazione Maven semplificata;
- * configurazione automatica se possibile;
- * fornitura di caratteristiche non funzionali come metriche o configurazioni esternalizzate.

6.3.3.2 Spring Data

Spring Data è un progetto ad alto livello di Spring con lo scopo di unificare e facilitare l'accesso a diversi livelli di archiviazione, ovvero database sia relazionali che non relazionali. Spring Data fornisce interfacce generiche per gli aspetti legati a *Crud Repository* e *Paging And Sorting Repository* per ottenere implementazioni specifiche dall'archivio di persistenza. Grazie alle repository basterà scrivere l'interfaccia con i metodi di ricerca definiti in base ad un determinato insieme di convenzioni che possono variare in base al tipo di archiviazione in uso. Sarà Spring a fornire un'implementazione appropriata di tale interfaccia in fase di esecuzione.

6.3.3.3 Spring Data JPA

Spring Data JPA aiuta ad implementare facilmente sistemi basati su archiviazione di tipo JPA, [framework](#) per il linguaggio di programmazione Java che gestisce la persistenza dei dati in un database relazionale che usano le *Java Platform, Standard Edition* e *Java Enterprise Edition*. Questo modulo infatti si occupa del supporto per i livelli di accesso ai dati basati su JPA e rende più semplice la creazione di applicazioni basate su Spring che utilizzano tecnologie di accesso ai dati.

Per molto tempo implementare un livello di accesso ai dati è risultato complicato: era necessario scrivere molto codice per eseguire *query* semplici, impaginazione e controllo dei dati. Spring Data JPA mira a ridurre lo sforzo all'importo effettivamente necessario. Lo sviluppatore scriverà le interfacce del *repository*, inclusi i metodi personalizzati, e sarà Spring a fornire l'implementazione automaticamente.

6.3.3.4 Spring Data REST

Spring Data REST semplifica la creazione di servizi Web REST basati su *repository* di Spring Data analizzando il modello di dominio dell'applicazione ed espone le risorse HTTP basate su *hypermedia* per gli aggregati contenuti nel modello.

Spring Data REST definisce automaticamente gli *endpoint* di aggiunta, visualizzazione, rimozione e modifica. Così facendo viene eliminato molto del lavoro manuale solitamente associato a queste attività e rende semplice l'implementazione delle funzionalità [CRUD_G](#).

6.3.4 Blockchain

La [blockchain](#) è una tecnologia che sfrutta le caratteristiche di una rete informatica di nodi e consente di gestire ed aggiornare, in modo univoco e sicuro, un registro contenente dati ed informazioni in maniera aperta, condivisa e distribuita senza la necessità di un'entità centrale di controllo e verifica. Le applicazioni della [blockchain](#) sono contraddistinte dalla necessità di decentralizzazione e disintermediazione in modo da evitare banche, istituzioni finanziarie e così via.

Queste tecnologia abilita l'*Internet of Value* e si basa su un registro distribuito, sistema in cui i nodi di una rete possiedono la medesima copia di un database che modificano e leggono i dati indipendentemente dagli altri nodi. Il registro di queste tecnologie è strutturato come una catena di blocchi contenenti le transazioni ed il consenso è distribuito su tutti i nodi della rete, in questo modo i nodi possono partecipare al processo di validazione delle transazioni da includere nel registro.

6.3.5 Ethereum

Ethereum è una piattaforma digitale che permette di costruire una gamma di applicazioni decentralizzate. Queste applicazioni possono includere programmi di sicurezza, sistemi elettorali e metodi di pagamento. Questa piattaforma opera fuori dal mandato delle autorità centrali ed è per questo motivo che opera nel mercato delle criptovalute. **Ethereum** funziona come piattaforma software basata sulla tecnologia **blockchain**, simile a quella dei *bitcoin* perché conserva le transazioni, ed inoltre permette ai programmatori di costruire applicazioni decentralizzate. Queste applicazioni sono *software open source* che utilizzano la **blockchain** e gli *smart contract* e non hanno bisogno di mediatori.

6.3.6 NFT

NFT, acronimo per Non Fungible Token, si tratta di *token* insostituibili che rappresentano la proprietà di un bene, come un'opera d'arte digitale o reale. Un *token* è un oggetto con un valore particolare e simbolico, in particolare un *token* sulla **blockchain** è un'informazione digitale registrata su un registro distribuito. Questa informazione è associata ad un utente specifico e rappresenta un certo tipo di diritto, come proprietà di un oggetto o la ricezione di un pagamento. Gli **NFT** sono diversi dagli altri *token* perché sono insostituibili, unici, indivisibili ed è questo il motivo per cui si prestano alla cessione dei diritti di proprietà di opere d'arte.

Quando una persona compra un **NFT** si deve capire che:

- * non ha comprato l'opera: fisicamente essa rimane sempre in possesso dell'autore;
- * non ha comprato i diritti d'autore: non ha diritto a riprodurla, basare altre opere su di essa oppure utilizzarla come se l'avesse creata;
- * non ha comprato l'esclusività della riproduzione o dell'uso.

Quindi quando una persona decide di comprare un **NFT**, egli compra il diritto a dire che un certo oggetto è di sua proprietà.

Capitolo 7

Conclusioni

In questo capitolo verranno descritti un resoconto conclusivo del progetto di stage con una valutazione degli obiettivi raggiunti e del percorso di stage effettuato.

7.1 Raggiungimento degli obiettivi

Per quanto riguarda gli obiettivi prefissati ad inizio stage si può evincere dalla tabella sottostante che sono stati soddisfatti tutti gli obiettivi ad esclusione dell'obiettivo facoltativo legato al *JWT token*. Questa mancanza è dovuta al fatto che in fase di progettazione è stato deciso dall'azienda di gestire unicamente tramite il [back end](#) le transazioni e i *wallet address* e questo mi ha impossibilitato ad implementare la gestione del *JWT Token*.

Tabella 7.1: Tabella dello stato di soddisfacimento degli obiettivi

Obiettivo	Descrizione
O01	Soddisfatto
O02	Soddisfatto
O02	Soddisfatto
D01	Soddisfatto
F01	Non soddisfatto

Data l'impossibilità di soddisfare il requisito facoltativo a causa di una scelta progettuale, in accordo con l'azienda, ho prodotto le seguenti maschere non previste dal piano di lavoro:

- * homepage;
- * registrazione;
- * modifica dei dati personali;

- * modifica della password;
- * pagina dell'utente.

Inoltre ho iniziato la fase di *testing* anch'essa non compresa nel piano di lavoro.

7.2 Conoscenze acquisite

Durante questo periodo di stage ritengo di aver imparato molto. Sono infatti riuscita a consolidare e migliorare le mie conoscenze riguardanti il linguaggio Javascript, inoltre ho imparato ad utilizzare il [framework](#) Vue.js in maniera soddisfacente entrando nel dettaglio imparando ad utilizzare costrutti complessi rispetto alle conoscenze basilari. Inoltre ho imparato ad utilizzare le librerie Vuetify e Vuex, la prima molto utile per creare [web application](#) con *material design* e *responsive* mentre la seconda è utile per effettuare le chiamate al [back end](#) ed utilizzare lo state management.

Ho potuto inoltre migliorare la mia capacità di affrontare e risolvere un problema. Se prima dello stage ero più propensa a fare affidamento totale a chi è più esperto di me, ho imparato a non fermarmi alla prima difficoltà ma a cercare per prima cosa la soluzione al problema in autonomia e, solo dopo diversi tentativi, provare a chiedere ad una persona più esperta di me in materia.

Infine, avendo dovuto lavorare con altri colleghi stagisti, ho migliorato ulteriormente la mia capacità di lavorare in gruppo e di comunicazione. Infatti ho capito che queste due caratteristiche sono elementi molto importanti per poter riuscire a sviluppare al meglio un prodotto software.

7.3 Valutazione personale

Ritengo che questa esperienza di stage sia stata stimolante e costruttiva che mi ha permesso di crescere sia a livello personale che a livello intellettuale, affacciandomi per la prima volta al mondo del lavoro.

Ho avuto la possibilità di conoscere meglio me stessa, le mie capacità e i miei limiti in campo lavorativo. In questo modo ho potuto migliorare le mie conoscenze informatiche, le mie capacità pratiche e il mio approccio con i colleghi.

Ho apprezzato molto l'ambiente lavorativo in cui ho lavorato: sia i tutor che gli altri dipendenti dell'azienda hanno creato un ambiente tranquillo dove poter lavorare. Posso ritenermi quindi soddisfatta della scelta compiuta qualche mese fa su dove iniziare questa esperienza.

Appendice A

Appendice A

A.1 Confronto tra framework di sviluppo

Durante il mio progetto di stage un altro stagista ha sviluppato le stesse maschere front end come quelle sviluppate da me, ma tramite un [framework](#) differente: [Angular.js](#). Grazie a questo, nell'ultima settimana in cui stavamo ultimando le diverse implementazioni e l'integrazione, abbiamo potuto fare un confronto fra i due [framework](#). Le differenze che abbiamo trovato possono essere riassunte nella seguente tabella.

Tabella A.1: Tabella riassuntiva delle differenze tra Vue.js ed Angular.js

	Vue.js	Angular.js
Curva di apprendimento	Bassa	Alta
Linguaggio	Javascript	Typescript
Comunity	Poco ampia ma in crescita	Molto ampia ma sta morendo un po' alla volta
Two-way data binding	Assente	Presente
Organizzazione dei file	Per ogni componente esiste un solo file contenente il codice della parte logica, html e stile	Per ogni componente si crea una cartella contenente un file per parte logica, html e stile
Peso alla creazione del progetto	Nel momento in cui si esegue per creare il progetto viene creata una cartella di peso pari molto esiguo	Nel momento in cui si esegue per creare il progetto viene creata una cartella di peso consistente

Inoltre il [framework](#) Vue.js viene utilizzato maggiormente per lo sviluppo di [web application](#) destinate ad un bacino di utente ampio mentre [Angular.js](#) viene utilizzato

per lo sviluppo *enterprise*, ovvero per lo sviluppo di [web application](#) da utilizzare all'interno di aziende.

Abbiamo notato come, a lato pratico, Vue.js fosse molto più veloce e semplice da utilizzare rispetto ad [Angular.js](#), ad esempio per creare una finestra di dialogo si riscontra meno difficoltà su Vue.js rispetto che su [Angular.js](#).

Inoltre Vue.js possiede il *plugin* Vuetify che aiuta a creare delle [web application responsive](#) ed utilizzabili anche su risoluzioni inferiori a quelle dello schermo da computer senza scrivere codice di stile. Questa cosa abbiamo notato essere assente in [Angular.js](#).

Glossario

Angular.js framework sviluppato da Google per lo sviluppo delle applicazioni web. [6](#), [61](#), [62](#)

API in informatica con il termine *Application Programming Interface API* (ing. interfaccia di programmazione di un'applicazione) si indica ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. [65](#)

back end parte del programma non visibile all'utente che permette il funzionamento delle interazioni con il front end. [1](#), [5](#), [6](#), [9](#), [37](#), [39–44](#), [46](#), [47](#), [51–53](#), [59](#), [60](#)

blockchain struttura dati condivisa definita come un registro le cui voci sono concatenate in ordine cronologico, la cui integrità si basa sulla crittografia. [1](#), [5](#), [6](#), [48](#), [57](#), [58](#)

CRUD in informatica CRUD, acronimo di create read update delete, sono le quattro operazioni di un database persistente. Nel contesto dello sviluppo di applicazioni web corrispondono alle quattro operazioni HTTP: put, get, post e delete.. [57](#), [65](#)

Ethereum piattaforma digitale che utilizza la blockchain ed opera nel mercato delle criptovalute. Con essa è possibile costruire applicazioni decentralizzate.. [5](#), [58](#)

framework architettura logica di supporto sulla quale un software può essere progettato e sviluppato, facilitandone lo sviluppo da parte del programmatore. [v](#), [1](#), [5](#), [6](#), [11](#), [33–35](#), [38](#), [53](#), [54](#), [56](#), [57](#), [60](#), [61](#)

front end parte del programma visibile all'utente con cui egli può interagire, tipicamente è un'interfaccia utente. [v](#), [1](#), [6](#), [9](#), [37](#)

NFT è un tipo speciale di token crittografico che rappresenta qualcosa di unico, i token non fungibili non sono interscambiabili. Questo concetto va in contrasto con le criptovalute che sono di natura fungibili, ovvero sostituibili.. [1](#), [5](#), [39](#), [65](#)

Scrum è un framework agile utilizzato per la gestione del ciclo di vita del software, iterativo ed incrementale, con lo scopo di gestire progetti, prodotti software o applicazioni di sviluppo. [6](#)

stakeholder in ingegneria del software uno stakeholder è una persona a vario titolo coinvolta nel ciclo di vita di un software, che ha influenza sul prodotto o sul processo. [6](#)

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. [65](#)

web application traduzione in inglese di applicazione web ed indica tutte le applicazioni distribuite, ovvero applicazioni accessibili via web cioè attraverso un'architettura di tipo client-server. [1](#), [4](#), [5](#), [11](#), [18](#), [34](#), [40](#), [60–62](#)

Acronimi

API [Application Program Interface](#). [37](#), [63](#)

CRUD [Create Read Update Delete](#). [63](#)

NFT [Non-Fungible Token](#). [5](#), [58](#), [63](#)

UML [Unified Modeling Language](#). [11](#), [64](#)

Bibliografia

Siti web consultati

Documentazione ufficiale Visual Studio Code. URL: <https://code.visualstudio.com/docs>.

Documentazione ufficiale Vue Test Utils. URL: <https://vue-test-utils.vuejs.org/>.

Documentazione ufficiale Vue.js. URL: <https://vuejs.org/>.

Documentazione ufficiale Vuetify. URL: <https://vuetifyjs.com/en/>.

Sito ufficiale Sync Lab. URL: <https://www.synclab.it/>.