



**UNIVERSITÀ
DEGLI STUDI
DI BERGAMO**






Progetto Ingegneria del Software

Anno accademico: 2021-2022

Gruppo di lavoro:

- Domenico Gaeni - matricola n° 1065107
- Fabio Palazzi - matricola n° 1066365
- Paolo Mazzoleni - matricola n° 1057949

Sommario

 Testo	4
 Project Plan	5
Introduction	5
Process model	6
Organization of the project	7
Standards, guidelines, procedures	7
Management activities	8
Risks	8
Staffing	8
Methods and techniques	9
Quality assurance	9
Work packages	10
Resources	10
Budget and schedule	11
Changes	11
Delivery	12
 Software Life Cycle	13
 Configuration Management	14
Struttura del progetto	14
Issue	14
Branch	15
 People Management	16
 Software Quality	17
Parametri riguardanti l'operatività del software	17
Parametri riguardanti la revisione del software	18
Parametri riguardanti la transizione verso un nuovo ambiente	18
 Requirements Engineering	19
Funzionali	19
Non Funzionali	20
 Modelling	21
Schema ER	21
Use-case diagram	24
Macchina a stati	25

 Software Architecture	26
Infrastruttura in locale	26
Deploy in produzione	27
 Software Design	29
Pattern utilizzato	29
 Software Testing e manutenibilità	30
Test statici	30
Test d'unità	30
CI / CD	33
 L'applicazione web	34
Panoramica dashboard	34
Pagina di log-in	35
Registrazione nuovo utente	35
Messa in vendita di un nuovo libro	36
Ricerca libri per autore	37
Ricerca libri per titolo	37
Ricerca libri per ISBN	37
Acquisto un libro	38
Consultazione storico degli acquisti	39
Recensione di un prodotto acquistato	39

Testo

Si gestisca una piattaforma software di vendita di libri fisici. Per accedere alla piattaforma, il cliente deve essere registrato. Inoltre il cliente, può essere contemporaneamente venditore e compratore. All'atto della registrazione dovranno essere memorizzati: il nome, il cognome, la email, la password, l'indirizzo di residenza, il numero di telefono e (opzionale) la carta di credito. La password dovrà essere memorizzata tramite cifratura HASH a 256 bit. All'atto della registrazione, deve essere verificato l'indirizzo email. Inoltre, al fine di rendere il sistema più sicuro, è richiesta la doppia autenticazione attraverso un codice OTP inviato al numero di telefono.

In caso di password dimenticata, il sistema deve mandare una mail di reset password contenente un link che attiverà la procedura di ripristino della password. Il link presente nell'email avrà scadenza di 1h dalla sua ricezione. In particolare prima di procedere con l'inserimento della nuova password il sistema deve chiedere l'OTP sul numero di telefono dell'utente coinvolto.

Il sistema offre le seguenti funzionalità ai propri utenti:

- Il venditore ha la possibilità di inserire un annuncio di vendita di un libro specificando: titolo, codice ISBN, autori, prezzo, genere.
- Un utente ha la possibilità di ricercare dei libri tramite i seguenti criteri: titolo, autore, ISBN, genere.
- Un utente ha la possibilità di acquistare un libro in vendita. In fase di checkout deve essere specificato l'indirizzo di consegna (in caso sia diverso da quello specificato all'atto della registrazione) e il metodo di pagamento (Paypal, carta di credito). Quando il sistema riceverà i soldi del libro, notificherà il venditore tramite un'email, in particolare sarà poi compito del venditore spedire il libro all'indirizzo indicato per email entro i giorni successivi all'acquisto. Inoltre, l'importo viene trattenuto dal sistema e il venditore riceverà la somma dovuta a consegna effettuata oppure se non verranno aperte contestazioni da parte dell'acquirente entro 30 giorni. Le contestazioni verranno aperte tramite un'apposita funzione del sistema e sarà poi compito del personale che gestisce la piattaforma contattare il venditore per informarsi sulla spedizione e/o procedere con il rimborso all'acquirente.
- Un utente potrà consultare il proprio storico di acquisti: dovranno essere memorizzati tutti i prodotti acquistati e venduti dal cliente per almeno 5 anni.
- Un utente dopo aver acquistato un libro potrà recensire il venditore: al fine di evitare truffe e venditori scorretti, il sistema deve permettere di valutare l'acquirente tramite una recensione (*punteggio da 1 a 5 con testo opzionale*).
- Un utente dopo aver acquistato un libro potrà recensire il prodotto: un acquirente può valutare il libro acquistato tramite una recensione (*punteggio da 1 a 5 con testo opzionale*).



Project Plan

Immaginiamo che il testo del problema, contenente i requisiti scritti in linguaggio naturale, sia una richiesta da parte di un ipotetico cliente indicato di seguito come Mario Rossi.

Mario ha idea di aprire una startup il cui prodotto sarà il sistema descritto nel testo ed è alla ricerca di possibili investitori, in quanto lui stesso non ha i fondi sufficienti per costituire la società e assumere degli sviluppatori. Gli investitori vogliono però vedere un prototipo di come sarà l'intero sistema a lavori finiti.

Mario si vuole quindi affidare alla nostra azienda costituita da Domenico, Fabio e Paolo al fine di progettare l'intero sistema e di sviluppare una prima versione, in modo che possa poi andare a presentare il prototipo a possibili investitori per ricevere un finanziamento.

La nostra azienda è una società di consulenza che ha solo noi tre come liberi professionisti, infatti ognuno di noi ha la propria partita IVA con regime forfettario. La nostra società emetterà fattura a Mario e noi emetteremo in seguito fattura verso la nostra società.

Inoltre, Mario ha la necessità di avere la documentazione riguardante il prodotto in quanto, alla fine della collaborazione, assumerà i propri sviluppatori. Sarà quindi la nuova società di Mario ad occuparsi di finire e mantenere l'intero sistema.

1. Introduction

Dopo un'attenta analisi dei requisiti scritti in linguaggio naturale con il nostro cliente Mario, di seguito saranno riportati i passi al fine di realizzare una prima versione del sistema richiesto.

Alla progettazione dell'intero sistema e allo sviluppo del primo prototipo ci lavoreranno tre persone: Domenico, Fabio e Paolo; di seguito saranno descritti i ruoli specifici per ognuno.

Dopo un primo incontro con il nostro cliente si è deciso insieme di sviluppare fisicamente un prototipo che implementi le seguenti funzionalità:

- registrazione di nuovi utenti, senza la verifica dell'indirizzo email e senza l'implementazione della doppia autenticazione;
- possibilità per un venditore di inserire un nuovo libro indicando gli attributi descritti nel testo;
- possibilità per un acquirente di consultare il catalogo dei libri disponibili tramite ricerca con parole chiavi quali titolo, autore, ISBN, genere;
- possibilità per un utente di acquistare un libro, specificando l'indirizzo di consegna senza effettuare il pagamento: in questa versione infatti non sono

gestiti i pagamenti, quindi l'utente per poter acquistare un libro dovrà specificare solo un indirizzo di spedizione;

- possibilità per un utente di consultare il proprio storico degli acquisti;
- possibilità per un utente di recensire un prodotto che ha acquistato e il suo venditore.

In particolare, insieme a Mario si è scelto di non implementare fisicamente l'integrazione con il servizio che manda l'email, il servizio che manda il codice OTP per la doppia autenticazione e il servizio per gestire i pagamenti, in quanto comportano dei costi elevati che il nostro cliente in almeno in questa prima versione non può finanziare.

In fase di progettazione dovrà comunque essere studiato l'intero sistema con tutte le integrazioni e le funzionalità descritte in precedenza.

Mario ha la necessità di avere una prima versione del prodotto entro la fine di gennaio 2022, quindi alla stesura di questo documento si hanno a disposizione circa 75 giorni.

Questi 75 giorni di tempo saranno suddivisi in due parti: una prima parte sarà dedicata completamente alla stesura della progettazione dell'intero sistema, mentre nella seconda verrà sviluppato concretamente il prototipo di cui sopra.

Ci saranno degli incontri intermedi con il cliente durante la fase di progettazione del sistema (prima di procedere con lo sviluppo), in modo da mantenerlo aggiornato e ricevere un suo feedback.

Entro la fine di Gennaio il prodotto da consegnare a Mario sarà una piattaforma web che implementa le funzionalità minime descritte in precedenza.

2. Process model

Per il processo di sviluppo del prototipo abbiamo optato per un approccio di tipo Agile, infatti puntiamo a rilasciare una versione di base del prodotto e integrarla mano a mano con piccoli aggiornamenti che porteranno con sé le funzionalità descritte sopra, in modo da facilitare anche le fasi di testing e debugging.

Fondamentale sarà il rapporto con Mario, che sarà invitato a partecipare alle riunioni settimanali sullo stato di sviluppo del software e in particolare nella fase di progettazione.

Per tenere traccia dello sviluppo delle funzionalità richieste, optiamo per l'utilizzo delle issue su Github, che saranno assegnate settimanalmente a ogni componente del gruppo in base a quello che si decide nei meeting. Mano a mano che ognuno andrà a completare i punti che gli sono stati assegnati, le issue verranno chiuse e i documenti

pubblicati sulla piattaforma.

3. Organization of the project

Le persone coinvolte nel progetto sono quattro: il cliente Mario, e il team di progettisti/sviluppatori, formato da Domenico, Fabio e Paolo. Di seguito saranno descritti i loro ruoli e le loro mansioni.

L'intera documentazione riguardante il progetto sarà riportata e tracciata su GitHub. In particolare, il team si incontra settimanalmente per analizzare il lavoro svolto dall'ultimo meet e pianificare le attività future, tracciando i task come issue su GitHub in modo che ogni persona del team sia al corrente di cosa stiano facendo gli altri. Alla repository GitHub inoltre avrà accesso anche il nostro cliente Mario che interagirà con noi e si assicurerà lui stesso che lo sviluppo vada come previsto.

4. Standards, guidelines, procedures

Il team ha deciso di sviluppare un'applicazione web per implementare il sistema descritto. L'applicazione sarà suddivisa in due parti:

- front-end - si occupa dell'interfaccia grafica e dell'interazione con il back-end;
- back-end - mette a disposizione delle API che verranno utilizzate dal front-end.

Il front-end verrà sviluppato utilizzando HTML 5 / CSS, in particolare utilizzando la libreria Bootstrap ([documentazione](#)) e la libreria di javascript JQuery ([documentazione](#)) per gestire le chiamate (API) al back-end e manipolare i dati da mostrare e salvare.

Abbiamo inoltre concordato, di utilizzare un database relazionale e abbiamo scelto in particolare un MySQL. Nonostante un database non relazionale offra tempi di risposta migliori (soprattutto al crescere delle dimensioni dei dati da memorizzare), abbiamo deciso di utilizzare un db relazionale perché più affidabile e perché non rilassa le proprietà ACID.

Il prototipo che consegneremo al cliente sarà hostato su Altervista, sia per motivi economici sia perché offre tutte le funzionalità utili per realizzarlo.

All'atto dell'autenticazione verrà creato un token univoco a livello di utente. Una volta autenticato, ad ogni chiamata al back-end verrà inserito nell'header della richiesta il token creato al fine di verificare se il cliente si è effettivamente loggato e sta usufruendo dei servizi cui può accedere.

5. Management activities

Abbiamo deciso di organizzarci secondo la filosofia dello "swat team" (Skilled Workers with Advanced Tools): gli incontri saranno settimanali e relativamente brevi, della durata di un'ora circa. Durante gli stessi potranno essere effettuate sessioni di brainstorming per condividere idee e risolvere problemi.

Il team di sviluppo si pone l'obiettivo di incontrarsi settimanalmente per aggiornarsi sullo stato dei lavori. In base alle esigenze si decide se effettuare la riunione virtualmente tramite la piattaforma Google Meet o se è necessario incontrarsi e svolgerla di persona.

Ad ogni incontro, i membri del gruppo espongono brevemente quali sono stati i punti portati avanti e quali sono state le problematiche incontrate sul proprio cammino. Mano a mano che si procede nell'esposizione, si aggiorna un documento che tiene traccia dei lavori svolti fino a quel momento. Alla fine di ogni meeting verrà scritta una breve relazione in merito a quanto discusso.

Alla fine del meeting ci si accorda su quali sono i punti da portare avanti; sarà il project manager ad assegnare a ciascun membro del gruppo, se stesso compreso, i compiti che dovranno essere eseguiti prima del prossimo incontro.

6. Risks

La durata del lavoro è di circa 75 giorni e l'obiettivo è quello di arrivare al termine con tutta la documentazione e il prototipo definito insieme a Mario. In particolare ci sono due scadenze da rispettare: il 10 gennaio per la consegna della documentazione e il 31 gennaio per la consegna del codice. Il rischio principale del progetto è che il nostro cliente non ci paghi la cifra pattuita insieme e di conseguenza di aver lavorato inutilmente. Per ridurre questo rischio abbiamo deciso insieme al cliente di suddividere il pagamento in due rate, i dettagli sono definiti nel punto 12 del presente documento. Facendo due tranches del pagamento abbassiamo il rischio in quanto se il cliente non ci paga la prima parte del lavoro possiamo fermare lo sviluppo del prototipo risparmiando così ore di lavoro.

7. Staffing

Utilizzando una modalità di sviluppo AGILE, non c'è una vera e propria distinzione fra i ruoli nel team. Inoltre, i task, decisi in comune accordo con il project manager, verranno inseriti come issue su Github e designati ad un programmatore. Nonostante questo, in una panoramica più generale possiamo definire i seguenti ruoli:

- Project Manager: Paolo Mazzoleni
- Progettista Software: Fabio Palazzi
- Progettista Database: Domenico Gaeni
- Back-end Developer: Domenico Gaeni
- Front-end Developer: Fabio Palazzi
- Tester: Paolo Mazzoleni

8. Methods and techniques

Per iniziare, il team si incontrerà per approvare lo USE-CASE DIAGRAM (precedentemente modellato da project manager e concordato con il cliente) in modo da definire i casi d'uso ovvero le iterazioni fra le varie componenti.

Per modellare il nostro sistema utilizziamo delle UML CLASS DIAGRAM. Questi diagrammi mostreranno le varie classi del nostro sistema ognuna delle quali conterrà attributi e metodi da implementare. Utilizzando tool esterni riusciremo, a partire dal UML CLASS DIAGRAM, a generare lo scheletro del nostro codice, che sarà OOP (e verrà implementato lato back-end).

Sempre sfruttando la potenza di UML, genereremo anche più SEQUENCE DIAGRAM in modo da modellare le varie operazioni del sistema come uno scambio di messaggi fra le componenti precedentemente definite. I vari diagrammi forniranno così una linea guida dei passi per implementare ogni singola funzionalità.

Per modellare i dati da salvare nel nostro db, utilizzeremo un ER CLASS DIAGRAM dove definiremo le varie tabelle da memorizzare e le varie relazioni (con le diverse cardinalità).

Nel caso durante la fase di implementazione ci accorgessimo dell'impossibilità di implementare funzionalità con la specifica approvata, il team si incontrerà nuovamente al fine di trovare una soluzione funzionante e concordata anche con il cliente.

9. Quality assurance

Il team punta a sviluppare un software che rispetti i parametri di qualità indicati dal modello di McCall:

- Correttezza
- Affidabilità
- Robustezza
- Integrità
- Usabilità

Per garantire la sicurezza utilizziamo librerie moderne, aggiornate frequentemente e poco inclini ad avere vulnerabilità.

Inoltre dividiamo il lato back-end da quello front-end, cosicché in futuro il cliente potrà assumere personale separato per lo sviluppo delle due parti.

Puntiamo a sviluppare un software che, oltre a rispettare i quality assurance, sia anche sicuro:

- verifica dell'indirizzo email: un utente, per completare il processo di registrazione, deve confermare i propri dati cliccando su un link apposito ricevuto via mail;
- doppia autenticazione tramite OTP: un utente, per poter accedere al proprio account, potrà richiedere che ogni accesso debba essere confermato tramite l'inserimento un codice univoco a tempo ricevuto per messaggio SMS.

10. Work packages

Il prototipo sviluppato sarà costituito da due parti principali, una front-end e una back-end. In questo modo tutta la parte relativa all'interfaccia grafica sarà presente nel codice del front-end e invece la parte di gestione dei dati e di interfaccia con il DB sarà gestita all'interno del back-end. Il back-end è sviluppato con il framework Lumen che si basa a sua volta sul framework Laravel che implementa il pattern MVC (Model-View-Controller) anche se in verità la parte di view sarà implementata all'interno del front-end.

Nel back-end ci saranno quindi delle classi (*model*) che modelleranno le varie tabelle presenti nel DB e delle classi (*controller*) che manipolano i modelli a seconda del comportamento desiderato. Quindi, per esempio, in fase di registrazione di un nuovo utente ci saranno due classi UserController e User. Un metodo della classe UserController si occuperà di ricevere i dati dal front-end, di validarli a seconda del tipo e di creare una nuova istanza del modello User.

Inoltre il mondo PHP è diverso per quanto riguarda la gestione dei packages, infatti in questo linguaggio i modelli saranno presenti in una apposita cartella sotto il nome di Model, mentre i controller sotto la cartella Controller.

11. Resources

Le risorse designate al progetto sono un project-manager più due sviluppatori che lo affiancano. Al team verrà inizialmente fornito un account [altervista.org](https://www.altervista.org) per rilasciare i primi delivery, poi all'atto della consegna, il sistema verrà migrato su un account AWS al fine di implementare anche funzionalità aggiuntive di sicurezza quali la doppia

autenticazione tramite OTP e la verifica dell'indirizzo email. Al team saranno forniti 3 computer portatili MacBook PRO, una connessione internet in fibra ottica, una stanza e un ufficio per l'incontro con il cliente.

12. Budget and schedule

La durata del progetto è di 75 giorni e durante questo periodo il team si concentrerà fondamentalmente su due grandi fasi, la prima di *progettazione* e la seconda di *sviluppo*. Il costo totale del lavoro sarà circa di 11.700€. Il prezzo è calcolato togliendo dal totale circa 10 giorni festivi (week-end e festività), quindi per un totale di 65 giorni. Il team è formato da 3 persone che lavorano part-time per un totale di 4 ore giornaliere. Ogni ora è pagata 15€ lordi.

Il costo totale è quindi definito dalla seguente espressione: $65 \text{ giorni} * 3 \text{ persone} * 4 \text{ ore} * 15 \text{ €/ora} = 11.700\text{€}$

Insieme a Mario si è scelto di fare due tranches di pagamento, il 60% (pari a 7.020€) al termine della progettazione e il restante 40% (4.680€) a conclusione dei lavori. Si è scelto di dare più importanza alla documentazione in quanto verrà progettato l'intero sistema in tutte le sue funzionalità e di dare invece meno importanza allo sviluppo in quanto verrà sviluppato un prototipo con solo alcune funzionalità. La consegna stimata della prima parte è attorno al 10 gennaio, mentre la data di fine lavori è attorno al 1 febbraio.

13. Changes

Oltre alle feature base incluse inizialmente nel prodotto e descritte al punto uno di questo documento, l'obiettivo sarà quello di mettere le basi per ulteriori funzionalità allo scopo di fornire un servizio più completo, in accordo con Mario. Queste verranno inserite nelle successive versioni del software e includeranno:

- un carrello, nel quale i clienti potranno inserire prodotti da acquistare ed effettuare un unico pagamento, a differenza di quello che avviene nella versione base del software nella quale è necessario effettuare un pagamento separato per ogni prodotto;
- liste desideri, nelle quali i clienti potranno inserire prodotti per eventuali acquisti futuri, con la possibilità di crearne più di una e ad ognuna assegnare un nome;
- una dashboard personalizzata per ogni cliente, che varierà in base all'autore/genere di libri acquistati in precedenza;
- la possibilità di salvare un metodo di pagamento (carta di debito/credito/prepagata) per poterla utilizzare negli acquisti futuri senza la necessità di inserirla ogni volta;

- la possibilità di acquistare buoni regalo con credito spendibile sul sito.

14. Delivery

L'incontro con il cliente è fissato ogni settimana, il giorno può variare in base agli impegni ma abbiamo comunque stabilito una frequenza settimanale in modo da rimanere in stretto contatto con il cliente ed essere allineati alle sue richieste e opinioni. Abbiamo inoltre concordato una volta a settimana in modo da riuscire a capire meglio le esigenze e consegnare un prodotto più vicino alle aspettative del cliente. Ad ogni incontro presenteremo dei prototipi ovvero delle schermate da noi sviluppate contenenti l'interfaccia grafica e poche funzionalità back-end così da mostrare le funzioni principali al cliente e in caso concordare dei cambiamenti. L'obiettivo mostrando questi prototipi sarà quello di discutere l'aspetto grafico con il cliente mostrandolo anche sui vari device (esempio Smartphone, Tablet, PC).

I prototipi mostrati al cliente non avranno solo lo scopo di mostrare l'interfaccia grafica finale, ma verranno poi implementati dal team che aggiungerà il software necessario a raggiungere i requisiti specificati per ogni singola schermata.

Software Life Cycle

Per il processo di sviluppo il team ha scelto un approccio di tipo Agile poiché meglio si adatta alla nostra metodologia di lavoro:

- Consideriamo importante il team, le abilità e le interazioni tra di noi. I lavori vengono assegnati in base alle capacità che ognuno possiede. Se uno della squadra si accorge che il compito va oltre le sue competenze, avvisa gli altri e se ne discute insieme.
- Nel team non c'è una struttura di tipo gerarchico: ci consideriamo tutti allo stesso livello. Questo ci porta ad essere più compatti e motivati.
- Consideriamo più importante un prodotto software funzionante che spendere troppo tempo sui documenti: a sviluppo iniziato, se ci accorgiamo che ci sono punti da implementare che non avevamo in precedenza considerato, li andiamo ad aggiungere poi alla documentazione.
- Utilizziamo la tecnica del pair programming: per lo sviluppo dei lati front-end e back-end, gli sviluppatori hanno l'opportunità di lavorare in coppia: uno dei due scrive il codice mentre l'altro funge da supervisore e revisore; i due possono scambiarsi di ruolo.
- Durante lo sviluppo del software utilizziamo la tecnica del timeboxing: suddividiamo lo sviluppo del codice in intervalli temporali entro i quali dobbiamo sviluppare determinate funzionalità.
- Diamo molta importanza alla collaborazione con il cliente, infatti Mario partecipa agli incontri settimanali, che siano virtuali (tramite Google Meet) o di persona: durante questi il team lo aggiorna sullo stato dei lavori per quanto riguarda la documentazione e il project plan e gli mostra fino a che punto è arrivato lo sviluppo delle funzionalità incluse nel prototipo.
- Il team è propenso al cambiamento e punta a rispondergli in maniera adeguata. Se il cliente avanzasse delle richieste in fase di implementazione o comunque dopo l'avvenuta stesura dei requisiti, cercheremo di fare in modo di accontentarlo stando in ogni modo attenti a quello che si è sviluppato fino a quel momento.
- In accordo con il cliente, il team si è prefissato l'obiettivo di sviluppare un prototipo di quello che sarà il sistema finale; sarà poi Mario a presentarlo a possibili investitori per ricevere un finanziamento. Saranno incluse solo le funzionalità base descritte nel project plan, per poter dare una dimostrazione del funzionamento del nostro prodotto.

Come approccio di progettazione software per lo sviluppo del sistema ci siamo attenuti alla model-driven architecture (MDA), o architettura guidata dal modello.



Configuration Management

Tutto il lavoro svolto che si tratti di documentazione o di codice viene salvato in una repository su GitHub in condivisione con tutti i membri del team e con il cliente Mario, che potrà osservare l'andamento dei lavori.

Struttura del progetto

All'interno della repository sono presenti 4 cartelle:

- Cartella meetings: conterrà un file per ogni incontro svolto dal team contenente i vari punti trattati nel corso delle riunioni.
- Cartella documentation: conterrà i file in merito alla progettazione, a partire dal project plan. In seguito verranno aggiunti i file che tratteranno i vari punti del progetto.
- Cartella front-end: conterrà il codice del front-end della piattaforma, sviluppato in HTML, CSS e JavaScript implementando la libreria Bootstrap.
- Cartella back-end: conterrà il codice del back-end della piattaforma, sviluppato in PHP utilizzando il framework Lumen.

Issue

Nei vari meeting settimanali vengono creati i macro task che dovranno essere sviluppati nel corso della settimana successiva. Le varie attività sono create come issue su GitHub, aggiungendo una breve descrizione del lavoro e la persona di riferimento che si occuperà di portarla a termine. Durante la settimana ogni membro del team è libero di creare ulteriori issue se lo ritiene necessario.

Le issue possono trovarsi in vari stati a seconda del loro avanzamento. Per tenere traccia dello stato si utilizza una classica board Kanban suddivisa in 5 colonne: To Do, Progress, Pull Request, Testing, Done.

- *To do*: contiene le issue che sono state create e che devono essere sviluppate.
- *Progress*: quando il responsabile della issue inizia a lavorarci, la sposta in questa colonna. In questo modo gli altri membri del team e il cliente stesso sanno a che punto si trova lo sviluppo di una determinata issue.
- *Pull Request*: quando il responsabile della issue apre una pull request per poter unire il proprio lavoro al branch master, la issue si trova in questo stato.
- *Testing*: una volta approvata la pull request, se è stato sviluppato del codice, l'issue passa in questo stato, altrimenti passa direttamente allo stato Done. Il nostro tester si occuperà di verificare che il codice sviluppato soddisfi i requisiti descritti all'interno del task. Se tutto funziona correttamente allora passa allo stato successivo (Done), altrimenti lo stato torna in To Do con un commento contenente una motivazione per tale scelta.

- *Done*: le issue che si trovano in questo stato sono state chiuse e riviste da almeno un'altra persona.

Branch

Il branch principale è il master. Su questo ci sarà tutta la documentazione e tutto il codice sviluppato dal team. Tutti i file presenti in questo branch sono stati rivisti da almeno due persone: l'autore e un altro membro del team.

Quando si sviluppa una issue, ovvero quando lo stato transita da *To Do* a *Progress*, viene aperto un nuovo un branch a partire dal master dandogli un nome significativo. Su questo branch si svilupperanno tutte le funzionalità necessarie al fine di completare la issue. Una volta terminato il lavoro, viene aperta una pull request e si sposta la issue nello stato *Pull Request*. A questo punto sarà compito del team andare a revisionare le issue che sono in questo stato. In caso di dubbi si lascia un commento, in modo che l'autore della request possa risolvere eventuali problemi. Se tutto torna, si approva la pr.

Una volta che la pr ha ottenuto almeno un'approvazione da un'altra persona l'autore può unire il codice nel master e chiudere il branch. Dopo il merge nel master, se le modifiche implementate devono essere testate allora la issue passa allo stato *Testing*, altrimenti allo stato *Done*.

L'approvazione di una pr da parte di un'altra persona serve a fare in modo che il codice scritto venga revisionato da almeno un altro membro del team in modo da avere la certezza che non si introducano errori.



People Management

La startup ha deciso di basarsi su una struttura organizzativa ad hoc. Utilizzando questa, abbiamo una grande capacità di adattamento e non abbiamo una vera e propria divisione prefissata del lavoro; è infatti il team a decidere come dividere e assegnare i vari task. Inoltre, in caso di difficoltà durante lo sviluppo, due membri del team possono affiancarsi per risolvere il problema in coppia. Il vantaggio di utilizzare l'ad hoc è che abbiamo un'organizzazione flessibile, adattabile e informale che è definita da una mancanza di struttura formale.

Il team è organizzato secondo il modello SWAT (skilled worker with advanced tools). Abbiamo quindi una squadra composta da tre persone che sviluppano utilizzando canali di comunicazione informali (issue su github). Le riunioni tra il team sono tutte informali, mentre abbiamo deciso di fissare con una frequenza settimanale una riunione formale con il cliente al fine di mostrare i delivery sviluppati.

A nostro parere, il vantaggio di rifarsi ad un'ad hoc e ad un modello SWAT è che ogni componente del team sviluppa software e implementa funzionalità di componenti che lo affascinano maggiormente. Vogliamo quindi non solo arrivare all'obiettivo finale di sviluppare software e svilupparlo bene, ma di costruirlo anche con piacere.

Anche se non abbiamo una vera e propria distinzione del lavoro, possiamo riassumere in maniera più generale i settori di sviluppo e le persone che ne dedicano più tempo:

	Project Manager	Progettista Software	Progettista Database	Front-end	Back-end	Testing
Domenico	V	V	V		V	V
Fabio		V		V		V
Paolo	V		V	V	V	V

Software Quality

Il team si è prefissato l'obiettivo di sviluppare un software che rispetti i parametri e gli attributi di qualità definiti da McCall-Richards-Walters nel documento da loro redatto nel 1977, di seguito elencati e suddivisi in categorie come indicato dagli autori.

Parametri riguardanti l'operatività del software

- *Correttezza* - Il prodotto software da noi realizzato soddisfa i requisiti e le specifiche indicate dal cliente. Le funzionalità da noi non sviluppate ma solo progettate saranno portate avanti dal team di sviluppo che Mario assumerà dopo che avremo consegnato il prototipo.
- *Affidabilità* - Il software è affidabile in quanto revisionato a livello di team e sottoposto a una lunga fase di testing prima del rilascio.
- *Efficienza* - Le risorse utilizzate dal prodotto sono limitate in quanto si tratta di un'applicazione web e molte delle operazioni sono effettuate lato back-end dal server. È quindi comunque necessaria una connessione internet e un browser per accedere ai servizi del software da noi sviluppato.
- *Integrità* - Il prodotto che sviluppiamo è sicuro:
 - Le password degli utenti sono crittografate nel database secondo il *Secure Hash Algorithm SHA-256*, quindi non visibili chiaramente anche a chi ha accesso al db.
 - All'atto dell'inserimento dei dati durante il processo di registrazione, per prevenire tentativi con indirizzi email errati e/o non di proprietà di chi effettivamente cerca di registrarsi, viene inviato un link di verifica alla mail specificata per completare il processo di registrazione: l'account non verrà attivato e non potrà essere utilizzato fino all'apertura del link.
 - Sarà possibile per un utente attivare la "verifica in due passaggi" per l'accesso all'account. Sarà sufficiente registrare sul sito web il proprio numero di telefono: dopo aver inserito indirizzo email e password, per poter proseguire con il login si riceverà tramite SMS un codice OTP da sei cifre da inserire nell'apposito campo.
 - L'utilizzo del protocollo HTTPS garantisce che lo scambio di dati tra client e server sia crittografato e quindi al sicuro da eventuali tentativi esterni di accesso ai dati.
- *Usabilità* - Il prodotto è semplice da utilizzare, infatti non sono richieste particolari abilità per poterne usufruire. L'uso dei servizi è facilitato su tutti i tipi di piattaforma grazie all'utilizzo della libreria Bootstrap che permette di avere componenti grafici semplici e facili da utilizzare.

I requisiti "base" includono:

- Avere una connessione internet.
- Avere un indirizzo email di proprietà per il processo di registrazione.
- Sapere come effettuare un pagamento online.

Parametri riguardanti la revisione del software

- *Manutenibilità* - La fase di individuazione degli errori è semplificata dalla nostra scelta di separare i lati front-end, sviluppato in HTML 5/CSS/javascript, e back-end, sviluppato in PHP. Il processo è inoltre facilitato poiché si ha la parte grafica separata dalla parte di gestione dei dati: è semplice fare manutenzione avendo persone specializzate in ambiti differenti.
- *Testabilità* - Tutte le feature incluse nel prototipo sviluppato sono testabili (tramite test manuali e test automatici) prima del rilascio del software al pubblico. Delle funzionalità che saranno invece da noi solo progettate si occuperanno in futuro i dipendenti della società del cliente: saranno loro a svilupparle e a effettuare la fase di testing.
- *Flessibilità* - Come per la manutenibilità, anche il processo di modifica, adattamento e perfezionamento del software è facilitato dalla scelta di separare i lati front e back-end.

Parametri riguardanti la transizione verso un nuovo ambiente

- *Portabilità* - È possibile utilizzare i servizi offerti dal nostro prodotto da qualsiasi dispositivo dotato di un browser e di una connessione internet. Dal punto di vista grafico invece, grazie all'utilizzo della libreria Bootstrap, il sito web sarà responsive, ovvero in grado di adattarsi a qualunque tipo di schermo su cui sarà aperto.
- *Riusabilità* - Per la parte grafica utilizziamo un template della libreria Bootstrap che il team aveva implementato per delle applicazioni sviluppate in precedenza. Il template sarà riutilizzabile anche per future applicazioni web in quanto si tratta di una base grafica.
- *Interoperabilità* - Nel futuro il nostro prodotto software potrà essere integrato con altri servizi in quanto sviluppiamo in modo separato front-end e back-end: il lato back-end potrà fornire delle API pubbliche utilizzabili da altri utenti. La scelta in tal senso sarà fatta dai dipendenti della nuova società di Mario.



Requirements Engineering

Uno dei passi più importanti prima di scrivere software è quello di analizzare i requisiti, ovvero definire con il cliente sia quelli funzionali (funzionalità del sistema) che quelli non funzionali (tempi di risposta, quantità di dati da immagazzinare ecc.). In questo caso abbiamo un'analisi custom driver perché è proprio il cliente stesso a guidare il team e a fornire i requisiti da implementare.

Oltre ai requisiti che si possono cogliere dal testo del problema, abbiamo organizzato ulteriori meeting per:

- Capire le funzionalità volute dal cliente (*elicitation*).
- Negoziare le funzionalità volute con quelle a nostro parere realizzabili (*negotiation*).
- Specificare i requisiti finali approvati sia dal team che dal cliente e contenuti nel documento REQUIREMENTS SPECIFICATIONS.

Il documento citato può subire variazioni a fronte di modifiche richieste dagli attori e concordate con i restanti altri. Queste richieste verranno presentate durante la fase di validazione in cui il team mostrerà al cliente parte del prodotto finale tramite prototipi. Qualora il team avesse fatto errate supposizioni, può concordare con il cliente una soluzione.

Inoltre, il project manager Paolo ha applicato un'ulteriore suddivisione dei requisiti secondo il modello MoSCoW.

Nel documento dei requisiti concordato con il cliente ci siamo accordati su quelli qui di seguito riportati.

Funzionali

- Must Have:
 - autenticazione cliente tramite mail e password;
 - registrazione di nuovi utenti, senza la verifica dell'indirizzo email e senza l'implementazione della doppia autenticazione;
 - il cliente per accedere alla piattaforma deve essere obbligatoriamente registrato;
 - alla chiusura della pagina web, il cliente dovrà autenticarsi nuovamente per usufruire dei servizi.
 - devono essere memorizzati per ogni cliente: il nome, il cognome, la email, la password, l'indirizzo di residenza, il numero di telefono e (opzionale) la carta di credito;
 - possibilità per un venditore di inserire un nuovo libro indicando gli attributi descritti nel testo;
 - possibilità per un acquirente di consultare il catalogo dei libri disponibili tramite ricerca con parole chiavi quali titolo, autore, ISBN, genere;

- possibilità per un utente di acquistare un libro, specificando l'indirizzo di consegna senza effettuare il pagamento: in questa versione infatti non sono gestiti i pagamenti, quindi l'utente per poter acquistare un libro dovrà specificare solo un indirizzo di spedizione.
- Should have:
 - possibilità per un utente di recensire un prodotto che ha acquistato e il suo venditore;
 - possibilità per un utente di consultare il proprio storico degli acquisti.
- Won't have:
 - recupero password tramite email e codice OTP sul numero di telefono;
 - doppia autenticazione tramite codice OTP inviato al numero di telefono.

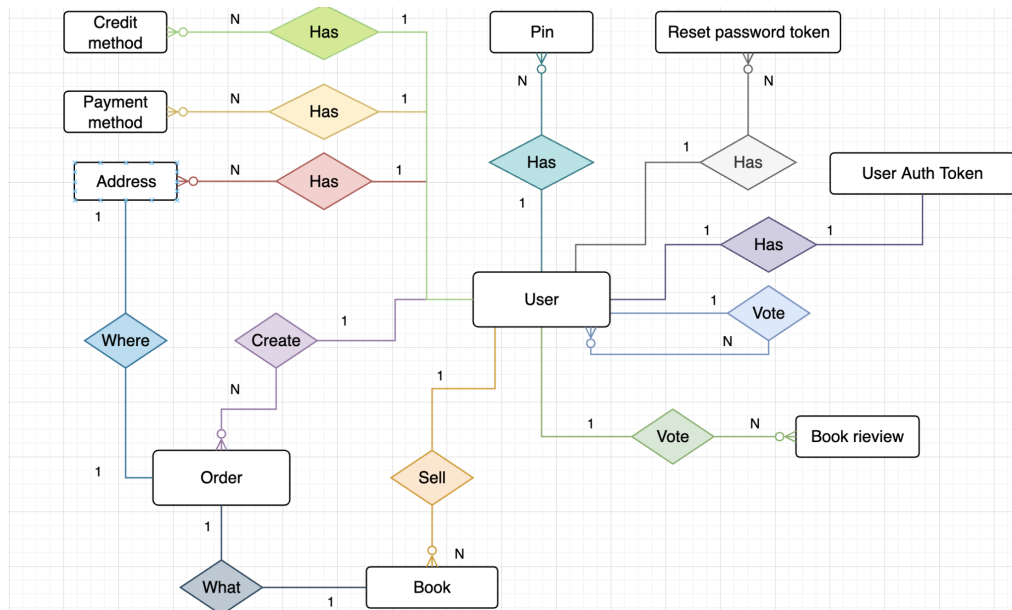
Non Funzionali

- Must have:
 - la password deve essere memorizzata nel db tramite cifratura hash a 256 bit;
 - la password deve essere lunga almeno 12 caratteri, contenente almeno un numero e un simbolo;
 - Il sistema implementato deve essere usufruibile da una vasta gamma di device quali smartphone, tablet e computer.
- Should have:
 - il tempo di risposta al login di un cliente deve essere inferiore ai 2 secondi;
 - il tempo di ricerca tramite parole chiave deve essere inferiore ai 2 secondi.
- Won't have
 - il codice OTP inviato al numero di telefono (in caso di reset password o di autenticazione) deve arrivare entro 30 secondi la richiesta;
 - il codice OTP inviato per email deve avere una scadenza di 5 minuti;
 - durante la registrazione, o la richiesta di reset delle password, la email deve essere stata inviata entro 1 minuto dalla richiesta.



Schema ER

Di seguito è rappresentato lo schema ER con tutte le entità e relazioni da modellare nell'applicazione. Come si può notare non sono presenti gli attributi delle singole entità in modo da non complicare ulteriormente lo schema: si è preferito dare importanza alle singole entità e alle relazioni tra di loro. Gli attributi saranno rappresentati in seguito nello schema delle tabelle della base di dati.



La rappresentazione in tabelle della base di dati è la seguente:

```
graph TD
    subgraph Table_Addresses [Addresses]
        id
        address
        user_id
        deleted_at
    end
    subgraph Table_Payment_Methods [Payment Methods]
        id
        card_number
        user_id
    end
    subgraph Table_Reset_Password_Tokens [Reset Password Tokens]
        id
        token
        expired_at
        user_id
    end
    subgraph Table_Users [Users]
        id
        name
        last_name
        email
        password
        phone
    end
    subgraph Table_Pins [Pins]
        id
        pin
        confirmed_at
        expired_at
        user_id
    end
    subgraph Table_Users_Auth_Tokens [Users Auth Tokens]
        id
        auth_token
        expired_at
        user_id
    end
    subgraph Table_Books [Books]
        id
        title
        isbn
        authors
        price
        description
        gender
        user_id
        deleted_at
    end
    subgraph Table_Orders [Orders]
        id
        status
        user_id
        book_id
        address_id
        created_at
    end
    subgraph Table_Reseller_reviews [Reseller reviews]
        id
        user_id
        description
        vote
        user_id_reviewed
    end
    subgraph Table_Book_reviews [Book reviews]
        id
        user_id
        book_id
        description
        vote
    end
    subgraph Table_Credit_Methods [Credit Methods]
        id
        iban
        user_id
    end
```

The diagram illustrates the database schema for a system, featuring 11 tables with their respective attributes:

- Addresses**: id, address, user_id, deleted_at
- Payment Methods**: id, card_number, user_id
- Reset Password Tokens**: id, token, expired_at, user_id
- Users**: id, name, last_name, email, password, phone
- Pins**: id, pin, confirmed_at, expired_at, user_id
- Users Auth Tokens**: id, auth_token, expired_at, user_id
- Books**: id, title, isbn, authors, price, description, gender, user_id, deleted_at
- Orders**: id, status, user_id, book_id, address_id, created_at
- Reseller reviews**: id, user_id, description, vote, user_id_reviewed
- Book reviews**: id, user_id, book_id, description, vote
- Credit Methods**: id, iban, user_id

Si è scelto di introdurre l'attributo `deleted_at` sulla tabella `addresses` e sulla tabella `books` perchè, siccome sono collegati tramite chiave esterna alla tabella `orders`, in caso di cancellazione di un indirizzo rimane comunque il riferimento nell'ordine effettuato.

Per quanto riguarda la tabella `books`, se il venditore elimina un libro che è presente in vari ordini, verrà eliminato dalla tabella `books` andando a impostare la data di `deleted_at`, in questo modo nella tabella `orders` ci sarà sempre il riferimento al libro venduto. Questo approccio verrà implementato lato back-end tramite la funzionalità `SoftDeletes`.

Di seguito c'è una breve descrizione per ogni tabella:

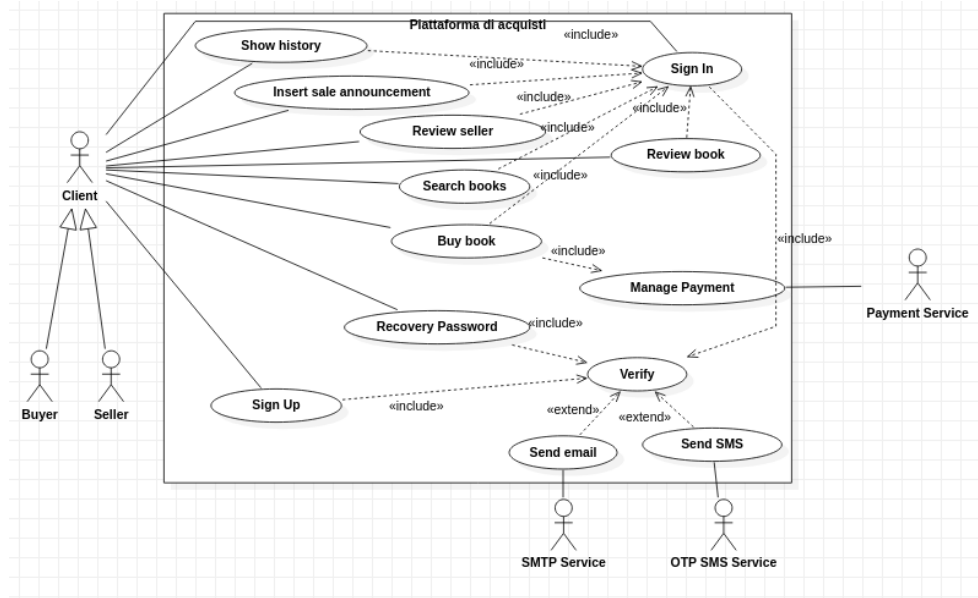
- **Users:** contiene le informazioni in merito all'utente che si registra alla piattaforma, in particolare il campo `password` contiene l'HASH della password.
- **Addresses:** contiene gli indirizzi dei vari utenti registrati.
- **Payment methods:** contiene i codici delle carte di credito delle persone, utilizzate per poter acquistare dei libri.
- **Credit methods:** contiene l'IBAN delle persone per poter ricevere il pagamento della vendita del libro.
- **Reset Password Tokens:** contiene i vari token che sono generati quando un utente richiede il reset della password. Il token avrà una scadenza di un'ora e sarà utilizzato poi per poter estrapolare le informazioni dell'utente che richiederà la password.
- **Pins:** contiene l'elenco dei PIN che saranno generati e mandati all'utente. Ogni PIN avrà una scadenza e nel momento in cui l'utente conferma il numero di telefono tramite il PIN valido verrà aggiunta la data attuale nella colonna `confirmed_at`.
- **Books:** contiene le varie informazioni in merito al singolo libro. Il campo `descrizione` conterrà una generica descrizione del libro, quale per esempio le condizioni in cui si trova (nuovo, ecc).
- **Reseller Reviews:** contiene le recensioni che gli utenti danno ai vari venditori dopo che hanno acquistato un loro libro, con un voto che andrà da 1 a 5 e una descrizione opzionale.
- **Book Reviews:** contiene le recensioni dei vari libri che sono stati acquistati da un utente, con un voto che andrà da 1 a 5 e una descrizione opzionale.
- **Users Auth Tokens:** contiene i token di autenticazione al back-end dei vari utenti. I token vengono generati ogni volta al login e hanno una durata di 1 giorno. Per poter utilizzare il back-end i token devono essere passati come Bearer Token nel header

Authorization della richiesta HTTP.

- Orders: contiene i dettagli del singolo ordine, in particolare contiene il riferimento di chi ha effettuato l'ordine (user_id), di cosa ha acquistato (book_id) e dove deve essere consegnato (address_id). C'è anche un'indicazione dello stato in cui si trova l'ordine in modo da sapere sempre in che situazione si trova. Lo stato può essere per esempio:
 - pending: in questo stato quando l'ordine è stato appena creato ed è in attesa che arrivi il pagamento;
 - processing: ci si trova in questo stato quando l'ordine è stato pagato e il venditore si deve occupare della spedizione;
 - shipped: ci si trova in questo stato quando l'ordine è stato spedito al destinatario;
 - delivered: l'ordine è arrivato a destinazione e l'acquirente segnala che il pacco è arrivato;
 - done: quando il pagamento sarà versato al venditore, l'ordine è da considerare completato.
- Infine nella tabella orders c'è un riferimento alla tabella books, questo significa che se il venditore deve modificare il prezzo di un libro che è già stato acquistato da altre persone il prezzo cambierebbe anche per le persone che hanno già pagato. Per far fronte a questa problematica si è deciso che ogni volta che il venditore modifica qualche attributo di un libro, come per esempio il prezzo, dietro le quinte si andrà ad eliminare il libro che deve essere modificato e si andrà a creare un nuovo elemento con l'attributo aggiornato, in questo modo tutti i libri che sono già presenti nei vari ordini non subiscono alcuna variazione.

Use-case diagram

Per modellare i casi d'uso, ovvero le funzionalità degli attori nel sistema, abbiamo modellato uno use-case diagram.



L'attore principale è il cliente che è esteso sia dall'attore compratore che venditore, infatti un cliente (che venditore o compratore sia), una volta registrato può sia vendere che comprare libri. Non abbiamo quindi funzionalità specifiche ma solo comuni ad entrambi i ruoli.

Un cliente ha molte funzionalità che possiamo dividere in base al contesto in cui si ubicano.

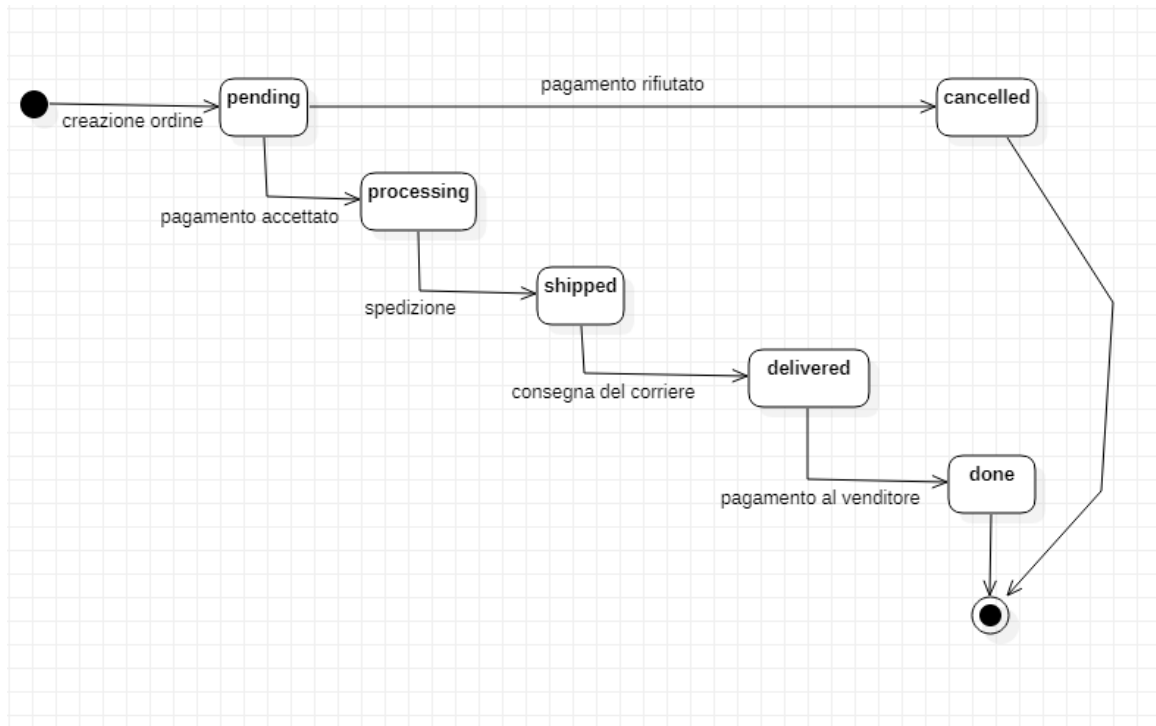
Abbiamo infatti un primo gruppo che comprende : Show History, Buy Book, Insert Sale Announcement, Review Seller, Review Book, Search Book che possono essere effettuate solo da clienti loggati. In particolare, questi Use Case includono Sign In, proprio per il motivo specificato sopra. Lo Use Case Buy Book include Manage Payment, infatti per comprare un libro si deve pagare il venditore. Inoltre rappresentiamo solamente il Payment Service ovvero il sistema che si occupa di gestire i pagamenti.

Il secondo tipo di funzionalità è relativo alle azioni eseguibili dal cliente per usufruire dei servizi specificati sopra. Tra esse abbiamo Sign in, Sign Up, Recovery Password. Entrambe includono il caso d'uso Verify in quanto per essere eseguite con successo necessitano una verifica di alcuni parametri.

Il terzo e ultimo tipo è Verify ovvero si occupa di verificare la correttezza dei dati inseriti nelle funzionalità del secondo tipo e inoltre, qualora fosse richiesto, di inviare email (Send Email) o SMS (Send SMS) al fine di implementare la cosiddetta *doppia autenticazione* gestita tramite gli attori: SMTP Service e OTP SMS Service.

Macchina a stati

Di seguito viene rappresentata la state machine riguardante lo stato di avanzamento di un ordine.



Nel suo ciclo di vita, un ordine si può trovare nei seguenti stati:

- pending: l'ordine è stato appena creato; si è in attesa del pagamento;
- processing: il pagamento dell'ordine è stato accettato; il venditore si deve occupare della spedizione;
- cancelled: l'ordine è stato annullato a causa del rifiuto del pagamento (ad es. saldo insufficiente sulla carta indicata come metodo di pagamento);
- shipped: l'ordine è stato spedito al destinatario;
- delivered: l'ordine è arrivato a destinazione e l'acquirente ha segnalato che il pacco è arrivato;
- done: il pagamento è stato versato al venditore: l'ordine è da considerare completato.



Software Architecture

L'architettura del nostro sistema è un'architettura client server basata sul pattern MVC (model - view - controller). Abbiamo quindi il front-end che ricopre il ruolo di view in quanto mostra risultati (statici e dinamici che siano) all'utente. Il back-end ricopre invece sia il ruolo di model, in quanto si occupa di astrarre e manipolare la suite di dati attraverso un database, che di controller, in quanto attraverso librerie di routing accetta richieste http (GET, POST, PUT, DELETE anche autenticate) e, tramite un'interazione con il database, ritorna i risultati all'view.

In questo modo abbiamo voluto creare un'architettura a servizi in cui ogni tipo di richiesta http accettata dal controller può essere vista come un servizio a sé e quindi eseguibile in una macchina diversa. Nel nostro caso, abbiamo deciso di integrare questa struttura in modo da migliorare la leggibilità e soprattutto la manutenibilità del codice. Ma è bene sottolineare che al momento la struttura del backend è più monolitica, infatti tutti i servizi vengono eseguiti sulla stessa macchina. Un ulteriore vantaggio nell'aver predisposto i servizi è che minimizziamo l'accoppiamento e nascondiamo la logica interna.

L'architettura è sicura: per accedere alla piattaforma viene creato all'atto dell'autenticazione un token (Bearer) che viene salvato sia nel nostro database che nella memoria locale `localStorage` gestita dal framework `jquery`. Questo token dovrà quindi essere passato ad ogni chiamata http formulata dopo essersi loggati, introducendo un doppio livello di sicurezza:

- il primo è a livello view (*front-end*): tramite codice `jquery` verifichiamo che il cliente sia correttamente autenticato verificando l'esistenza del token nel `localStorage` e passandolo al backend per controllarne la validità. Tuttavia, il codice `jquery` è eseguito localmente e quindi è vulnerabile in quanto l'utente potrebbe manipolare tutto il codice, cancellando ad esempio lo script che controlla la corretta autenticazione dell'utente loggato.
- il secondo è a livello controller-model (*back*-*end*) per risolvere il problema sopra citato: lato server andiamo a controllare che ogni richiesta http autenticata abbia nel campo `authorization`, contenuto nella header, un token bearer valido. In caso affermativo, viene ritornato un messaggio con uno stato 200 e il contenuto della richiesta, se no viene ritornato un messaggio con stato 401 ovvero *non autorizzato*.

La view è stata realizzata utilizzando HTML, la libreria BOOTSTRAP e il framework jQuery per gestire le richieste asincrone da inviare al backend.

Il back-end è stato realizzato utilizzando un database MySQL e il Framework Laravel con PHP 8 ed è stato sviluppato utilizzando Docker, di seguito sono descritti i dettagli dell'infrastruttura in locale e in produzione.

Infrastruttura in locale

I container utilizzati per eseguire correttamente il progetto in locale sono descritti nel file `docker-compose.yml`.

Di seguito vengono descritti brevemente i singoli servizi:

- database: si basa su un'immagine mysql 8, vengono specificate le credenziali di accesso al db. Esse (DB_PASSWORD e DB_DATABASE) sono specificate nel .env e possono essere modificate a piacere.
- lumen: si basa su un'immagine apache con PHP 8.0 customizzata. La ricetta del container si può trovare nella cartella environments, all'interno della quale sono presenti 3 sottocartelle, ognuna contenente l'immagine Docker utilizzata a seconda del contesto:
 - base: ricetta base che viene estesa da dev e prod.
 - dev: ricetta utilizzata per lo sviluppo in locale. Estende l'immagine base e ci aggiunge le estensioni per Xdebug (per il debug di PHP). Viene utilizzata all'interno di docker-compose.yml.
 - prod: ricetta utilizzata per l'ambiente di produzione. Estende l'immagine base ed effettua alcune operazioni di ottimizzazione, come per esempio cancellare i file e le cartelle inutili, copiare il .env con tutte le credenziali per l'accesso ai vari servizi, ed infine eseguire le migrazioni che andranno ad allineare il database di produzione con le nuove modifiche effettuate. Su questo container viene eseguito il codice che scriviamo nella root del progetto, quindi le API che sviluppiamo "vivono" all'interno del container apache. Esse saranno raggiungibili in locale all'indirizzo: localhost:{PHP_HOST_PORT} (dove PHP_HOST_PORT è una variabile definita nel file .env, di default sarà 80). All'avvio del container vengono eseguiti alcuni comandi, quali:
 - composer install: scarica le dipendenze del progetto descritte nel composer.json
 - php artisan migrate --force: esegue le migrazioni in modo forzato. Questo significa che ogni volta che vengono avviati i container con docker-compose up -d, questo comando distruggerà tutte le tabelle e tutti i dati presenti nel DB in locale e li creerà nuovamente. Chiaramente se il container viene lasciato attivo, questo comando viene eseguito solo la prima volta. In alternativa si può omettere -force e verranno eseguite solo le migrazioni che non sono state mai eseguite. Questo comando può essere modificato a seconda delle esigenze.
 - apache2-foreground: esegue il servizio di apache2 in foreground in modo che il container rimanga attivo, altrimenti esso si spegnerebbe nel momento in cui non c'è un processo attivo in esecuzione.

Deploy in produzione

Ogni volta che viene fatto un push sul branch master viene eseguito il deploy in produzione, eseguendo la Github Action presente nella repo descritta nel file `.github/workflows/deploy.yml`.

Molto semplicemente all'interno della Github Action viene eseguito il checkout del codice, viene buildata l'immagine Docker di produzione, descritta all'interno del file `./environments/prod/Dockerfile` e viene caricata su Heroku. Per eseguire l'upload del codice

vengono passate le credenziali di Heroku e le varie informazioni del DB che sono presenti come secrets su Github.

Il database utilizzato è un PostgreSQL ed è incluso nel piano gratuito scelto per il progetto.

Il link pubblico per le api è raggiungibile a questo url:

<https://ingegneria-software.herokuapp.com/public/>

Un domani si possono scegliere soluzioni più avanzate per l'ambiente di produzione, quale per esempio AWS o Google Cloud con i loro servizi. Utilizzando un'immagine Docker dove è presente il codice di produzione, il processo di deploy del codice è molto semplice sulle varie infrastrutture, infatti l'unico requisito che serve è che sia presente il Docker Engine sulla macchina.

Software Design

Per lo sviluppo dell'applicazione web il team si è basato sul pattern Model-view-controller (MVC), allo scopo di suddividere il codice in parti che abbiano funzionalità distinte tra loro.

I dati, la loro gestione e le interazioni con il database sono gestite dai livelli Model-Controller a lato back-end, che si occupa di gestire le query e la connessione alla base di dati.

Il livello che corrisponde alla View è l'output che presentiamo in front-end, ovvero le pagine vere e proprie che appaiono all'utente. Il contenuto, generato dal codice PHP in back-end, è restituito da un Controller che lo modella basandosi sui dati acquisiti dal database (livello Model).

Il team si è inoltre preposto di scrivere codice basandosi sui concetti di astrazione e modularità, con l'obiettivo finale di avere un programma che abbia:

- basso accoppiamento, ovvero avere moduli altamente indipendenti tra di loro; ad esempio:
 - common coupling: non ci sono variabili globali condivise tra i moduli;
 - control coupling: nessun modulo va ad utilizzare codice che appartiene a un altro modulo;
 - external coupling: l'unico "oggetto" esterno al codice che i moduli condividono tra di loro è il database, per il quale però esiste un modulo apposito addetto all'invio e alla ricezione dei dati; per il resto non ci sono file condivisi.
- alta coesione, ovvero avere una alta correlazione delle funzionalità presenti dentro un singolo modulo; ad esempio:
 - procedural cohesion: per ogni procedimento ci sono delle azioni che vengono sempre eseguite una dopo l'altra; ad esempio, quando un utente inserisce il nome di un autore per cercare i suoi libri disponibili alla vendita, il modulo opportuno andrà a leggere la stringa inserita dall'utente, poi cercherà i libri correlati nella tabella del database e infine si occuperà di stamparli secondo determinate regole grafiche.

Pattern utilizzato

Nello sviluppo del codice, in particolare lato back-end, il team ha utilizzato il pattern "factory". Nello specifico, è stato utilizzato per generare dati fittizi in base alle varie tabelle presenti nel database, utili nella fase di testing del codice per controllare che tutto funzioni correttamente (maggiori dettagli sul loro funzionamento nel capitolo di Software testing e manutenibilità).

Le factories che sono state realizzate sono visibili a questo link: [back-end/database/factories/](#).



Software Testing e manutenibilità

Durante lo sviluppo del back-end sono stati implementati dei test statici e dei test di unità.

Test statici

Per i test statici si è utilizzata la libreria Psalm ([documentazione](#)) che offre una serie di test statici che vanno a verificare la corretta stesura del codice php in tutto il progetto.

Per esempio di seguito si commentano alcuni esempi di test statici che danno errore:

```
<?php

/**
 * @return array<string>
 */
function takesAnInt(int $i) {
    return [$i, "hello"];
}

$data = ["some text", 5];
takesAnInt($data[0]);
```

- Nel primo caso il test darà errore perché nella PHPDOC è dichiarato che la funzione ritorni un array di stringhe mentre nella realtà ritorna un array con un intero e una stringa.
- Nel secondo caso ci sarà un errore perché una stringa viene passata come parametro ad una funzione che si aspetta un intero.

In conclusione si può dire che i test statici si assicurano che non ci siano sviste da parte del programmatore e che il codice è scritto rispettando la sintassi.

Test d'unità

Oltre ai test statici sono stati scritti anche dei test ad hoc per verificare la validità di tutte le chiamate esposte. Per questa tipologia di test si è utilizzata la libreria PHPUNIT ([documentazione](#)).

I test sono scritti all'interno della directory: back-end/tests/.

Per poter funzionare nel modo migliore i test hanno bisogno di dati sui quali poter fare le varie operazioni. Senza scrivere nel codice i dati fissi, si può utilizzare un generatore di dati fake in modo che ogni volta cambino, rispettando però un pattern definito a priori.

Lumen mette a disposizione la libreria FakerPHP/Faker ([link alla repo](#)) che permette di generare dei dati fake a seconda delle necessità. Se per esempio si deve generare un nome e un email si può fare così:

```
<?php
require_once 'vendor/autoload.php';

// use the factory to create a Faker\Generator instance
$faker = Faker\Factory::create();
// generate data by calling methods
echo $faker->name();
// 'Vince Sporer'
echo $faker->email();
// 'walter.sophia@hotmail.com'
```

All'interno della directory: /back-end/database/factories sono presenti le varie classi associate alle tabelle del database nel quale sono definite le regole per la generazione di dati. In questo modo quando si avrà bisogno di una certa entità per poter eseguire i test posso crearla con dati fake che cambiano ogni volta, andando a “simulare” i dati reali.

Ogni test scritto verifica che le chiamate esposte al pubblico rispettino il comportamento atteso, testando il tutto sia in caso di successo ma anche in caso di errore.

Per esempio di seguito è descritto il test per la verifica della chiamata per la *creazione di un nuovo utente*:

```
public function testNewUser()
{
    // Viene generato un nuovo utente con dati fake.
    $user = User::factory()->raw();
    $name = $user['name'];
    $lastName = $user['last_name'];
    $email = $user['email'];
    $password = $user['password'];
    $phone = $user['phone'];
    // Viene generato un indirizzo.
    $address = Address::factory()->raw()['address'];
    // Viene generata una carta di credito.
    $cardNumber = PaymentMethod::factory()->raw()['card_number'];
    // Viene generata un iban.
    $iban = CreditMethod::factory()->raw()['iban'];

    // Controlliamo che sul database nella tabella users non sia presente
    // un utente con lo stesso nome, cognome, email e telefono.
    $this->notSeeInDatabase('users', [
        'name' => $name,
        'last_name' => $lastName,
        'email' => $email,
        'phone' => $phone,
    ]);
}
```

```

// Chiamiamo la richiesta di creazione di un nuovo utente,
// passando le varie informazioni.
$this->post('auth/register', [
    'name' => $name,
    'last_name' => $lastName,
    'email' => $email,
    'password' => $password,
    'phone' => $phone,
    'address' => $address,
    'card_number' => $cardNumber,
    'iban' => $iban,
]);
// Verifichiamo che la richiesta ritorni codice 200 (tutto ok)
$this->seeStatusCode(200);
$response = json_decode($this->response->original)->data;

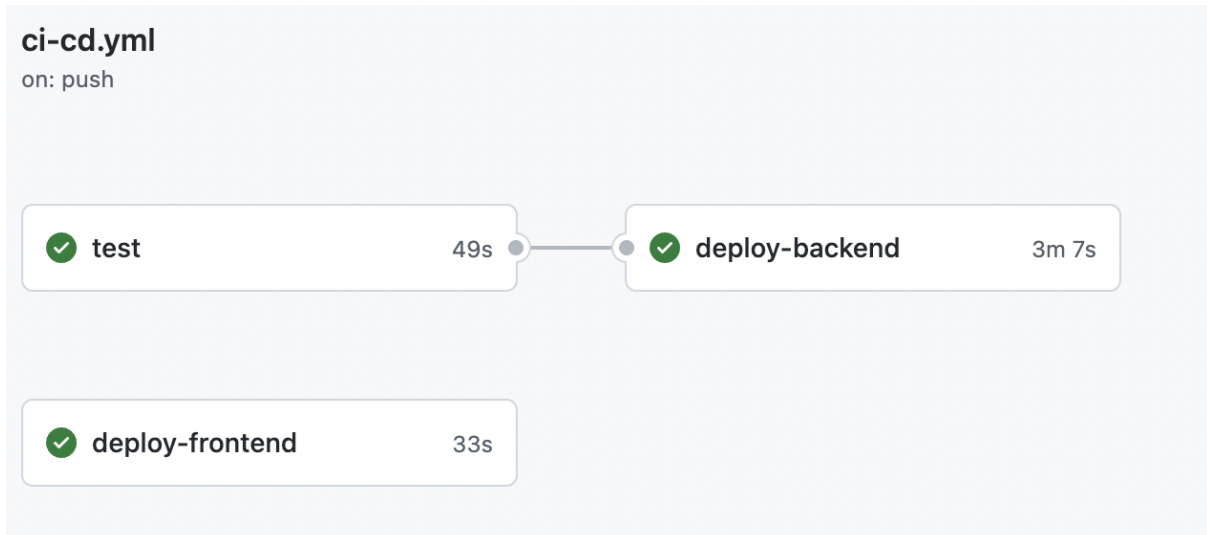
// Verifichiamo il formato e il contenuto della risposta
// che viene ritornata come risposta alla chiamata.
$this->seeJson([
    'data' => [
        'id' => $response->id,
        'name' => $name,
        'last_name' => $lastName,
        'email' => $email,
        'phone' => $phone,
        'address' => $address,
        'auth_token' => [
            'auth_token' => $response->auth_token->auth_token,
            'expired_at' => $response->auth_token->expired_at,
        ],
        'card_number' => $cardNumber,
        'iban' => $iban,
    ],
]);

// Infine controlliamo che nel database sia presente l'utente
// con i dati corretti che abbiamo generato.
$this->seeInDatabase('users', [
    'name' => $name,
    'last_name' => $lastName,
    'email' => $email,
    'phone' => $phone,
]);
}

```


CI / CD

Infine abbiamo implementato una Github action (`/.github/workflows/ci-cd.yml`) che esegue ad ogni push/merge sul branch master i seguenti tre job:



- **test**: questo job è necessario per poter fare il deploy del backend correttamente, infatti esegue in un ambiente di test dedicato i vari test, sia *statici* con `psalm` e *sia di unità*. Nell'ambiente di test vengono inizializzate tutte le tabelle e grazie poi ai dati fake si è in grado di eseguire i vari test in modo semplice. Se tutti i test sono corretti e non ci sono problemi allora si procede con il deploy del backend.
- **deploy-backend**: viene buildata l'immagine Docker di produzione con tutto il codice e poi viene deployato su Heroku.
- **deploy-frontend**: viene caricato tutto il codice relativo alla parte del front-end su Heroku che verrà esposto poi tramite NodeJS.

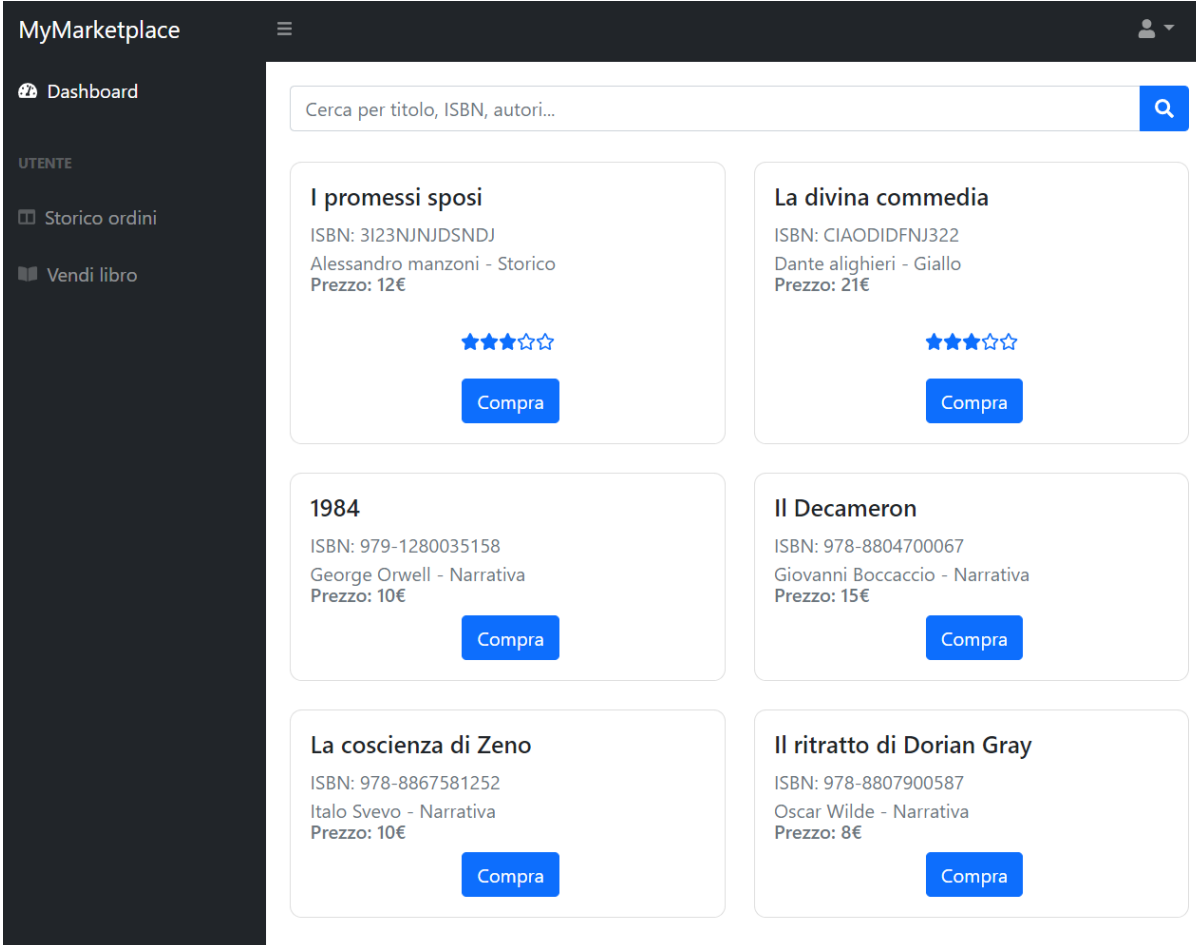
Avendo implementato un meccanismo di *continuous integration*, se un domani qualcuno mette mano al codice del back-end e rompe qualcosa, nel momento in cui viene eseguita la Github action e i test falliscono nulla verrà deployato in produzione, garantendo sempre il funzionamento di tutto quello presente sul branch master.

L'applicazione web

Di seguito riportiamo alcuni screenshot dell'applicazione web che il team ha realizzato, elencati secondo le funzionalità riportate nel project plan.

Il sito web è raggiungibile [questo link](#).

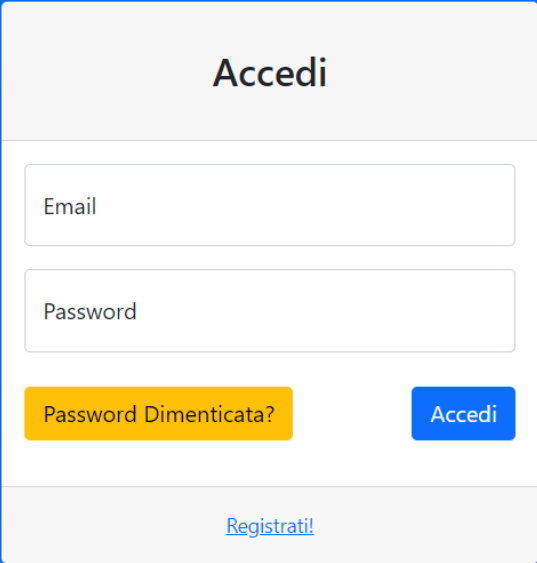
❖ Panoramica dashboard



The screenshot displays the 'MyMarketplace' dashboard. On the left is a dark sidebar with the following menu items: 'Dashboard' (with a user icon), 'UTENTE', 'Storico ordini' (with a list icon), and 'Vendi libro' (with a book icon). The main content area features a search bar at the top with the placeholder text 'Cerca per titolo, ISBN, autori...' and a magnifying glass icon. Below the search bar is a grid of six book listings, each in a white card with a light gray border. Each card contains the book title, ISBN, author, genre, price, a five-star rating, and a blue 'Compra' button.

Book Title	ISBN	Author	Genre	Price	Rating
I promessi sposi	3123NJNJSNDJ	Alessandro manzoni	Storico	12€	★★★★☆
La divina commedia	CIAODIDFNJ322	Dante alighieri	Giallo	21€	★★★★☆
1984	979-1280035158	George Orwell	Narrativa	10€	★★★★☆
Il Decameron	978-8804700067	Giovanni Boccaccio	Narrativa	15€	★★★★☆
La coscienza di Zeno	978-8867581252	Italo Svevo	Narrativa	10€	★★★★☆
Il ritratto di Dorian Gray	978-8807900587	Oscar Wilde	Narrativa	8€	★★★★☆

❖ Pagina di log-in



Accedi

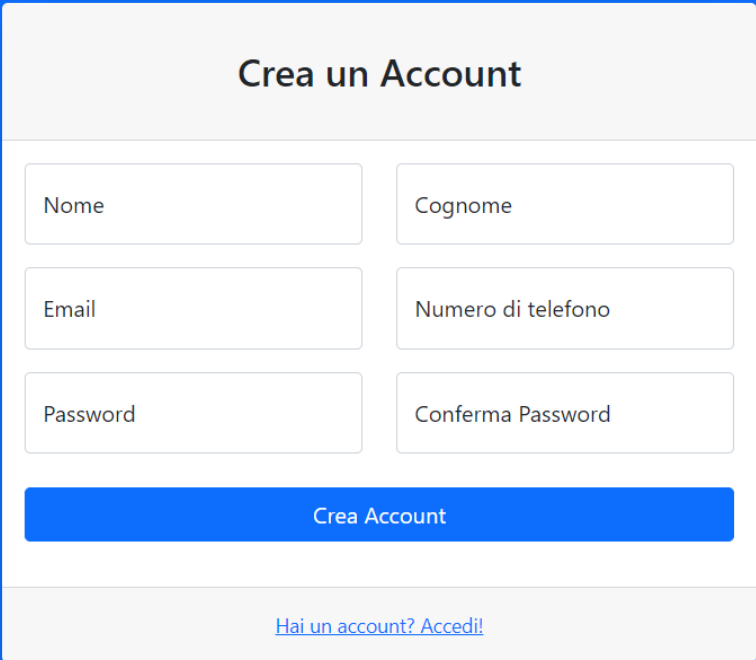
Email

Password

[Password Dimenticata?](#) [Accedi](#)

[Registrati!](#)

❖ Registrazione nuovo utente



Crea un Account

Nome

Cognome

Email

Numero di telefono

Password

Conferma Password

[Crea Account](#)

[Hai un account? Accedi!](#)

❖ Messa in vendita di un nuovo libro

MyMarketplace

Dashboard

UTENTE

Storico ordini

Vendi libro

Vendita

Inserisca il prodotto da vendere

Titolo

I promessi sposi

Autore

Alessandro Manzoni

Genere

Narrativa

ISBN

979-1259911483

Prezzo

15

Chiudi

Vendi

MyMarketplace

Dashboard

UTENTE

Storico ordini

Vendi libro

Vendi

Libri attualmente in vendita

La coscienza di Zeno

ISBN: 978-8867581252

Italo Svevo - Narrativa

Prezzo: 10€

Interrompi Vendita

Il ritratto di Dorian Gray

ISBN: 978-8807900587

Oscar Wilde - Narrativa

Prezzo: 8€

Interrompi Vendita

La fattoria degli animali

ISBN: 978-8822749765

George Orwell - Narrativa

Prezzo: 5€

Interrompi Vendita

I promessi sposi

ISBN: 979-1259911483

Alessandro Manzoni - Narrativa

Prezzo: 15€

Interrompi Vendita

❖ Ricerca libri per autore

MyMarketplace

Dashboard

UTENTE

Storico ordini

Vendi libro

Orwell

1984

ISBN: 979-1280035158

George Orwell - Narrativa

Prezzo: 10€

Compra

La fattoria degli animali

ISBN: 978-8822749765

George Orwell - Narrativa

Prezzo: 5€

Compra

❖ Ricerca libri per titolo

MyMarketplace

Dashboard

UTENTE

Storico ordini

Vendi libro

I promessi SPOSI

I promessi sposi

ISBN: 3123N1NJDSNDJ

Alessandro manzoni - Storico

Prezzo: 12€

★★★★☆

Compra

❖ Ricerca libri per ISBN

MyMarketplace

Dashboard

UTENTE

Storico ordini

Vendi libro

978-8867581252

La coscienza di Zeno

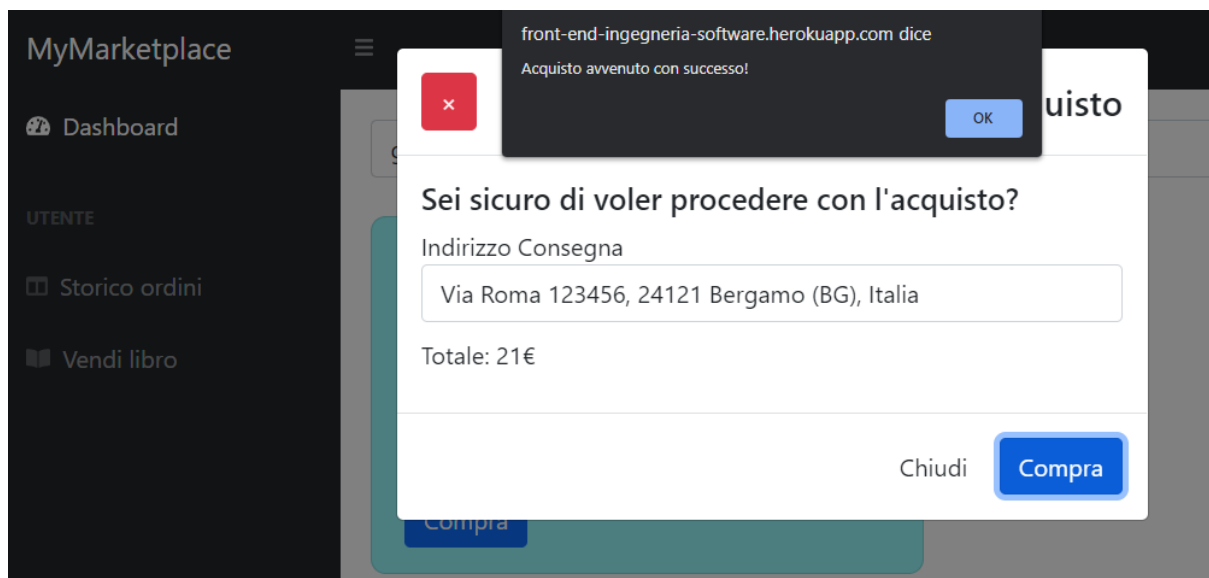
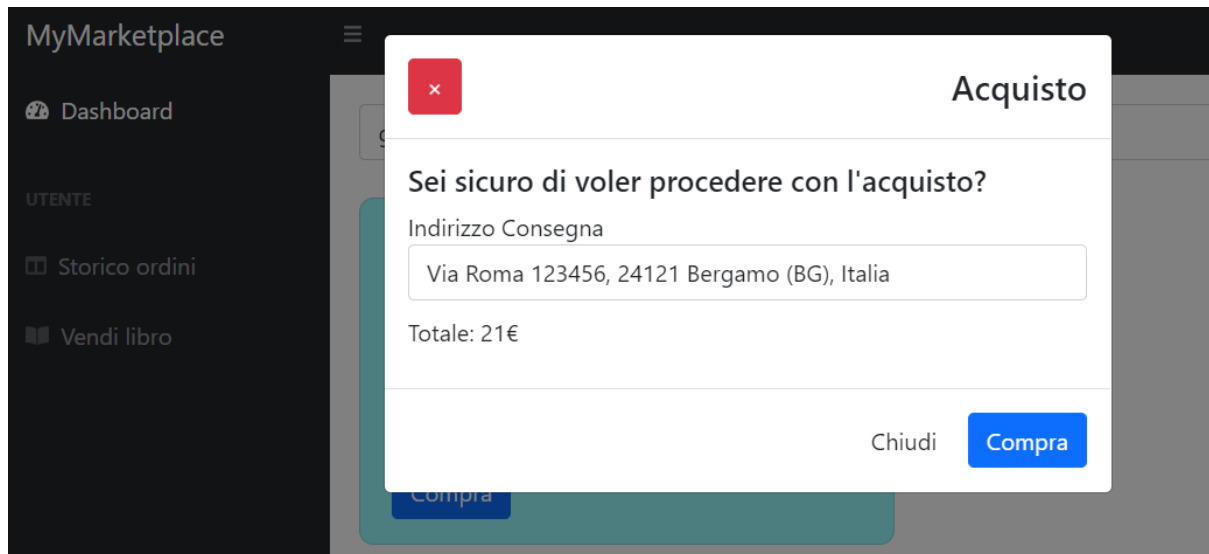
ISBN: 978-8867581252

Italo Svevo - Narrativa

Prezzo: 10€

Compra

❖ Acquisto un libro



❖ Consultazione storico degli acquisti

MyMarketplace

☰

Dashboard

UTENTE

Storico ordini

Vendi libro

Libri Comprati

La divina commedia

Dante alighieri - Giallo

Prezzo: 21€

Comprato da: domenico gae

☆☆☆☆☆

Vota

❖ Recensione di un prodotto acquistato

MyMarketplace

☰

Dashboard

UTENTE

Storico ordini

Vendi libro

Libri Comprati

La divina commedia

Dante alighieri - Giallo

Prezzo: 21€

Comprato da: domenico gae

★★★★☆

Vota