Documentazione

UniMarket

Progetto di Ingegneria del Software

Di Davide Dell'Anno, Davide Carissoni, Francesca Corrente

Università degli Studi di Bergamo
Facoltà di Ingegneria Informatica
2024-2025

Indice

1 Project Plan

1.1 Introduzione

UniMarket è stato progettato con l'obiettivo di semplificare la gestione della spesa per gli studenti.
Si tratta di un'applicazione che consente di selezionare facilmente i prodotti desiderati e creare un carrello virtuale, ottimizzando il processo di ricerca e acquisto.
L'applicazione non solo semplifica la gestione della spesa quotidiana, ma offre anche funzionalità pensate per le esigenze specifiche degli studenti, come la possibilità di salvare le proprie liste della spesa, monitorare i costi in tempo reale per rispettare il budget mensile e ricevere notifiche su offerte speciali o sconti.
UniMarket mira a migliorare l'esperienza di spesa online, offrendo una soluzione pratica ed efficiente per le esigenze quotidiane degli studenti.

1.2 Modelli di Processo

Lo sviluppo del progetto seguirà un processo **RAD** (Rapid Application Development), con l'obiettivo di realizzare un'applicazione funzionante entro i tempi stabiliti dal *time box*. Verrà adottato il **modello di Kano** per classificare i requisiti del progetto in base alla loro capacità di soddisfare le preferenze del cliente.

1.3 Organizzazione del Progetto

Per questo progetto è stata creata una **Squadra SWAT**, ovvero un team agile e versatile, composto da membri con competenze complementari, che si adatta rapidamente ai cambiamenti del progetto. In questo modo è più facile rispondere prontamente alle esigenze del cliente e dell'ambiente di sviluppo, mantenendo un'elevata efficienza nella gestione delle attività.
I ruoli sono stati assegnati come segue:

- **Team Leader: Davide Carissoni**
  Ha il compito di coordinare le attività, pianificare i moduli per gli sprint e assicurarsi il rispetto delle scadenze.
- **Sviluppatori, Progettisti, Tester: Francesca Corrente, Davide Carissoni, Davide Dell'Anno**
  Vista le piccole dimensioni del team, tutti i membri del progetto si occupano di tutte e tre le mansioni.

1.4 Standard, Linee Guida, Procedure

È molto importante che all'interno del progetto ogni componente del team coinvolto segua gli standard, le linee guida e le procedure concordate al fine di garantire un risultato facilmente comprensibile a tutti.
Gli standard Java Oracle definiscono le regole e le pratiche per la scrittura del codice, assicurando che il software sviluppato rispetti tali linee guida. In questo modo, l'approccio allo sviluppo adottato dal team garantirà uniformità e qualità.
Il linguaggio di programmazione scelto per lo sviluppo del software gestionale è Java.

Lo sviluppo avviene all'interno dell'**IDE Eclipse**, che permette la generazione automatica della documentazione tecnica tramite JavaDoc.
La Stesura della Documentazione del progetto è scritta utilizzando **LaTeX**, una scelta che semplifica la gestione della formattazione e il merge delle modifiche, garantendo un output professionale e ben strutturato. Per semplificare e velocizzare il processo di scrittura, è stato utilizzato il plugin LaTeX Workshop.
Per garantire una comprensione chiara e accurata del progetto, vengono utilizzati diagrammi **UML** (Unified Modelling Language) che permettono di visualizzare in modo intuitivo la struttura del sistema.
Per la gestione delle versioni e la condivisione del codice viene utilizzato **GitHub**, uno strumento essenziale per la collaborazione tra sviluppatori. GitHub consente di tracciare ogni modifica e mantenere il progetto sempre aggiornato. Inoltre, è stato utilizzato anche per la condivisione della documentazione e la gestione delle risorse correlate.
Per lo sviluppo dell'interfaccia grafica, verrà utilizzato **Vaadin**.

1.5 Attività di Gestione

Per garantire una gestione efficace delle attività e dello sviluppo del progetto, sono stati definiti i seguenti obiettivi:

- **Incontri settimanali:** monitorare l'avanzamento del lavoro, affrontare eventuali problemi riscontrati, discutere strategie risolutive e valutare la direzione dello sviluppo.
- **Kanban board su GitHub:** tracciare il progresso delle attività, organizzare il lavoro e assegnare incarichi ai membri del team.
- **Gestione del progetto con GitHub:** tenere traccia delle modifiche, monitorare gli aggiornamenti e facilitare la collaborazione.

1.6 Rischi

I rischi a cui bisogna prestare più attenzione sono:

- **Compatibilità dei sistemi utilizzati:** Il progetto viene sviluppato per essere utilizzato su desktop; pur usando maven, che consente il download automatico delle librerie richieste, potrebbero comunque esserci problemi di visualizzazione o calcolo su mobile o altri S.O. desktop.
- **Limitate quantità e tipologie di prodotto:** UniMarket è indipendente da reali supermarket, non dispone di tante opzioni per ogni genere di prodotto.
- **Bugs:** Ci potrebbero essere problemi generati dall'uso scorretto e imprevisto del programma da parte del cliente, è possibile che alcuni bug agli estremi dell'utilizzo improprio rimangano pur dopo i test.

1.7 Personale

Il team è composto da tre membri:

- **Francesca Corrente**

- **Davide Carissoni**
- **Davide Dell'Anno**

Ogni membro avrà funzioni simili all'interno del team, con ruoli assegnati in base alle esigenze, senza l'intervento di personale esterno.

1.8 Metodi e Tecniche

I metodi e le tecniche adottati per le diverse fasi dello sviluppo sono i seguenti:

- **Pianificazione dei requisiti:** si basa sull'elicitazione e sull'analisi dei requisiti per definire in modo chiaro le funzionalità richieste dall'applicazione.
- **Progettazione dell'applicazione:** prevede la definizione del funzionamento dell'applicazione, delle funzionalità incluse e degli strumenti software da utilizzare.
- **Programmazione:** consiste nello sviluppo del codice sorgente dell'applicazione seguendo le specifiche progettuali.
- **Testing:** include la scrittura e l'esecuzione di test, utilizzando strumenti appositi, per verificare il corretto funzionamento del codice e individuare eventuali anomalie.

Queste attività saranno supportate da un'adeguata gestione del controllo di versione e della configurazione dei componenti software, documentando ogni fase con precisione.

1.9 Garanzie di Qualità

Al fine di assicurare la qualità durante il processo di sviluppo e la realizzazione del software, verranno utilizzati i seguenti criteri:

- **Correttezza:** Il software deve funzionare come previsto e soddisfare i requisiti specificati, senza errori o comportamenti imprevisti. La correttezza implica che il sistema esegua tutte le operazioni correttamente, restituendo i risultati attesi in ogni condizione di input.
- **Affidabilità:** Il software deve essere stabile e operare in modo continuo e senza interruzioni. Un software affidabile deve resistere a guasti o malfunzionamenti, gestendo situazioni impreviste e mantenendo la sua funzionalità nel tempo, anche sotto carichi elevati o condizioni di stress.
- **Integrità:** L'integrità riguarda la protezione dei dati e il mantenimento della coerenza e correttezza durante le operazioni. Un software con buona integrità assicura che i dati non vengano alterati o danneggiati involontariamente, né durante l'elaborazione né durante la memorizzazione.
- **Usabilità:** Il software deve essere intuitivo e facile da usare per gli utenti finali. Una buona usabilità garantisce che gli utenti possano navigare nell'interfaccia senza difficoltà, riducendo il numero di errori operativi e migliorando l'esperienza complessiva.
- **Manutenibilità:** Un software manutenibile è facilmente aggiornabile e modificabile nel tempo per adattarsi a nuove esigenze o correggere eventuali difetti. Ciò implica che il codice sia chiaro, ben strutturato e documentato, facilitando interventi futuri senza introdurre regressioni o problemi aggiuntivi.

1.10 Pacchetti di lavoro

Nel team, il lavoro sarà suddiviso e gestito attraverso l'uso della Kanban Board, uno strumento che offre una visione chiara e immediata del flusso di lavoro.
La board consente di monitorare le milestone, visualizzare l'avanzamento delle attività che vengono completate o che sono in fase di sviluppo, e assegnare tasks.

1.11 Risorse

Le risorse relative allo sviluppo software sono le seguenti:

- **Eclipse IDE:** per lo sviluppo e la modifica del codice Java.
- **Maven:** per la gestione e l'automazione dei progetti Java.
- **SQLite:** per la creazione e gestione di un database embedded.
- **Vaadin:** per la progettazione iniziale dell'interfaccia grafica dell'applicazione.
- **JUnit:** per la sccrittura ed esecuzione di test.

Strumenti aggiuntivi:

- **LaTeX:** per la redazione della documentazione tecnica.
- **Papyrus:** per la modellazione di diagrammi UML.

Le risorse hardware si limitano ai dispositivi personali dei membri del team.

1.12 Budget

Il budget del progetto, essendo privo di spese finanziarie, è relativo al tempo di lavoro necessario, ipotizzato intorno alle 60 ore per ciascun membro.

1.13 Cambiamenti

Le modifiche principali saranno apportate durante la fase di ingegneria dei requisiti, per garantire un prodotto il più possibile aderente alle richieste. Eventuali modifiche al codice, derivanti dalla scrittura di test o da cambiamenti delle funzionalità dell'applicazione, verranno gestite attraverso la creazione di un nuovo branch nel repository, accompagnata dall'aggiornamento della documentazione correlata.

1.14 Consegna

La documentazione e l'applicazione finale saranno consegnate condividendo il progetto sul repository GitHub.

2 Gestione del progetto

2.1 Software Life Cycle

Il modello di processo scelto per la realizzazione del progetto è basato su un approccio **RAD** (Rapid Application Development), integrato con tecniche **Agile**, per garantire velocità, flessibilità e un'elevata capacità di adattamento alle esigenze degli utenti.
L'obiettivo di questo tipo di processo è fare uso delle tecniche **Agile** per bilanciare la mancanza di esperienza dei membri del team, dando priorità alla comunicazione e collaborazione, al fine di garantire velocità di realizzazione, qualità del sistema, flessibilità, iterazione.

2.2 Processo

Il processo è suddiviso in diverse fasi, ciascuna caratterizzata da attività iterative e collaborazione costante tra i membri del team:

1. **Analisi dei requisiti e prototipazione rapida:** Inizialmente vengono raccolti i requisiti fondamentali per comprendere appieno le esigenze degli utenti e le aspettative del cliente. Successivamente, vengono creati prototipi preliminari che permettono di raccogliere feedback, consentendo così di perfezionare e verificare le funzionalità sin dalle fasi iniziali del progetto.

2. **Sviluppo iterativo:** Ogni iterazione comprende le fasi di pianificazione, progettazione, implementazione e test. Le funzionalità vengono sviluppate in blocchi incrementali, consentendo il rilascio continuo di versioni con qualità elevata.

3. **Test continuo e revisione:** I test vengono eseguiti frequentemente durante tutto il processo per garantire un elevato standard qualitativo e per identificare tempestivamente eventuali problematiche. Il feedback continuo è fondamentale nell'orientare lo sviluppo del software. Questo processo permette di identificare le aree di miglioramento e di adattare il software a nuove esigenze in modo rapido e mirato.

4. **Timebox:** Il team ha suddiviso il lavoro in *timebox* della durata di una o due settimane, durante le quali ogni membro si è focalizzato su uno specifico obiettivo, contribuendo in modo mirato e collaborativo all'avanzamento del progetto. Essendo il gruppo composto da tre membri, la comunicazione dei processi e le eventuali revisioni del progetto sono state tempestive e su base giornaliera.

   Calendario timebox alla fine del documento.

2.3 Gestione della configurazione

Per la gestione della configurazione è stato adottato **GitHub**, utilizzato in combinazione con il tool **GitHub Desktop** per il controllo dei repository locali e le operazioni di *pull*, *commit* e *push* eseguite dai membri del team.
La distribuzione del lavoro è organizzata tramite una **Kanban board** integrata su GitHub, con cui, attraverso la creazione di *issues*, quando necessario, vengono definiti i compiti da svolgere e assegnati ai relativi membri del team, al fine di sistematizzare e ottimizzare il processo di sviluppo. L'uso di **milestones** permette di monitorare lo stato di avanzamento degli *issues* e del completamento delle *task*.

Durante l'implementazione, eventuali errori e problematiche sono notificati e vengono tempestivamente segnalati e gestiti tramite l'uso di *issues*. Ogni *issue* viene aperta dal membro del team che individua il problema e successivamente assegnata a uno o più colleghi competenti, che si occuperanno di risolverla.
La gestione degli *issues* e l'utilizzo di **branch** dedicati per le nuove versioni del prodotto hanno l'obiettivo di assicurare un approccio strutturato e accurato durante tutte le fasi dello sviluppo del progetto. L'implementazione e la revisione del codice saranno effettuate mediante la creazione di *pull request* da parte dei membri del team, prima di procedere al *merge* sul *branch* di destinazione.
La repository del progetto è stata organizzata nel seguente modo:

- **Documentazione**: contenente la documentazione del progetto.
- **Media**: contiene il logo del progetto.
- **UML**: contiene i diagrammi UML del progetto.
- **UniMarket**: contiene il codice sorgente dell'applicazione.

2.4 People Management

Il team presenta caratteristiche simili a quelle di un'unità **SWAT**, ma il livello di competenze tecniche dei membri non è particolarmente avanzato. Di conseguenza, vengono adottati principi propri di un **team Agile**, con particolare attenzione alla divisione del lavoro e alla comunicazione tra i membri.
Tutti i membri possiedono competenze tecniche simili, pertanto i ruoli non vengono assegnati in base a specifiche abilità, ma ogni membro partecipa attivamente a tutte le fasi dello sviluppo.
La suddivisione delle attività è decisa collettivamente durante i *timebox*, con la possibilità di ridefinire i ruoli, se necessario.
Non viene istituita una gerarchia formale all'interno del gruppo, sebbene sia presente un **leader** il cui unico compito è garantire la correttezza e la puntualità del lavoro svolto.

3 Requisiti

3.1 Software Quality

Per garantire la qualità nello sviluppo del progetto, verrà adottato lo standard **IEEE 830**, che definisce linee guida per la stesura e la gestione delle specifiche dei requisiti software. In particolare, saranno rispettate le seguenti caratteristiche di qualità:

- **Correttezza**: i requisiti saranno valutati e convalidati rispetto alle esigenze effettive degli utenti, garantendo che il software soddisfi le aspettative.
- **Univocità**: i requisiti saranno formulati in modo chiaro e preciso, in modo da evitare ambiguità.
- **Coerenza**: si assicurerà che non vi siano conflitti logici o temporali tra i requisiti.
- **Verificabilità**: ogni requisito sarà espresso in termini misurabili, consentendo un processo finito e oggettivo per verificare se sia stato soddisfatto.
- **Modificabilità**: il software deve evolversi, consentendo la possibilità di cambiamenti e modifiche, senza compromettere l'integrità del sistema.

3.2 Requirement engineering

Durante la fase di ingegneria dei requisiti l'obiettivo è quello di ottenere una descrizione chiara e completa dei problemi da risolvere, le caratteristiche del sistema e i vincoli da soddisfare; ciò avverrà tenendo conto delle funzioni da integrare e dei requisiti imposti dall'ambiente.
Per la stesura della documentazione relativa e l'effettiva definizione dei requisiti, viene seguito lo standard stabilito dall'**IEEE 830**, secondo il seguente modello:

1. **Introduzione:**
   (a) **Obiettivi e scopo**:

   L'obiettivo di *UniMarket* è fornire agli utenti un'applicazione che permetta di gestire e facilitare la selezione e l'ordinazione dei prodotti, semplificandone la ricerca e l'acquisto.

2. **Descrizione generale:**
   (a) **Prospettiva del prodotto:**
   Progettato per essere utilizzato su un PC standard con supporto per Java, l'applicazione offre un'interfaccia semplice e intuitiva, compatibile con i principali sistemi operativi.
   (b) **Funzioni del prodotto:**
   UniMarket offre una vasta gamma di funzionalità per semplificare l'esperienza di acquisto. Gli utenti possono registrarsi, navigare tra i prodotti, aggiungerli al carrello e scegliere modalità di pagamento sicure.

L'amministratore ha il controllo completo sulla gestione dei prodotti, degli ordini e del magazzino, mentre funzionalità aggiuntive come sconti, programmi fedeltà e modalità scura migliorano l'esperienza complessiva.
Tutte le funzionalità sono descritte in dettaglio nel paragrafo 3.

(c) **Caratteristiche dell'utente:**
Si assume che gli utenti siano in grado di usare in maniera corretta le funzioni di base del sistema, nonostante ciò, si cercherà di guidare il cliente durante l'utilizzo dell'applicazione.

(d) **Vincoli:**

i. **Modifiche al Database e al Codice:** Gli utenti non hanno la possibilità di modificare il database o il codice dell'applicazione. Le funzionalità sono predefinite e non modificabili dall'utente.
ii. **Disponibilità dei Prodotti:** Non è possibile acquistare prodotti che risultano esauriti. L'app garantisce solo la possibilità di selezionare articoli disponibili in magazzino.
iii. **Lingua:** L'app è progettata per un pubblico specifico e non è destinata a utenti internazionali. La lingua dell'interfaccia e i contenuti sono localizzati per un'area geografica definita.
iv. **Compatibilità con i Dispositivi:** L'app potrebbe non essere compatibile con tutti i dispositivi.
v. **Orari di ritiro Limitati:** Le opzioni di ritiro sono vincolate a orari specifici, con possibili limitazioni durante i giorni festivi e nel weekend.
vi. **Funzionalità di Ricerca Limitata:** La ricerca nell'app è limitata al catalogo di prodotti disponibile, che comprende un numero ristretto di articoli (es. 1000 prodotti). Le opzioni di filtraggio e ricerca potrebbero essere meno precise rispetto ad altre piattaforme più ampie.

(e) **Funzionalità Utente**

1. **Registrazione**: Permettere agli utenti di creare un account con e-mail e password o tramite social login.
2. **Login**: Accesso all'account personale per gestire ordini e informazioni.
3. **Selezione prodotti**: Possibilità di navigare e cercare tra categorie o direttamente tramite barra di ricerca.
4. **Aggiunta al carrello**: Funzione per aggiungere prodotti al carrello con possibilità di specificare quantità.
5. **Controllo carrello e modifica**: Revisione degli articoli selezionati con opzioni per rimuovere o modificare quantità.
6. **Opzioni di consegna**: Selezione di modalità e tempi di consegna (consegna a domicilio, ritiro in negozio, fascia oraria).
7. **Pagamenti:** Supporto a metodi di pagamento sicuri come carta di credito, PayPal, e wallet digitali.
8. **Tracciamento ordine:** Monitoraggio dello stato dell'ordine (in preparazione, in consegna, consegnato).
9. **Cronologia ordini:** Visualizzazione degli ordini passati per semplificare riacquisti o richiedere assistenza.

10. **Gestione del profilo:** Modifica delle informazioni personali, indirizzi di consegna e metodi di pagamento.
11. **Sconto applicabile:** L'utente può visualizzare eventuali sconti applicabili (per esempio, promozioni temporanee o sconti per nuovi utenti) e calcolare il prezzo finale con il dettaglio delle offerte attive.

(f) **Funzionalità Amministratore:**

12. **Gestione prodotti**: Aggiunta, modifica e rimozione di prodotti dal catalogo, compresi prezzi, descrizioni e immagini.
13. **Modifica disponibilità prodotti:** Aggiornamento delle quantità disponibili o sospensione temporanea di prodotti esauriti.
14. **Gestione ordini:** Visualizzazione e gestione degli ordini ricevuti con opzioni per modificare lo stato (es. in lavorazione, spedito).
15. **Gestione magazzino:** Monitoraggio dello stato delle scorte in tempo reale, con la possibilità di generare avvisi quando un prodotto è in esaurimento.

(g) **Funzionalità Aggiuntive** (non viene garantita la loro implementazione):

16. **Lista della spesa personalizzata:** Creazione di liste salvate per velocizzare gli acquisti futuri.
17. **Notifiche push o e-mail:** Informazioni su offerte, aggiornamenti di ordini, e nuovi prodotti disponibili.
18. **Filtri avanzati:** Filtraggio dei prodotti per prezzo, marca, disponibilità, o preferenze alimentari (bio, vegano, ecc.).
19. **Programma fedeltà:** Accumulo punti per ogni acquisto con possibilità di riscattare sconti o premi.
20. **Visualizzazione dettagliata dei prodotti:** L'utente può cliccare su ciascun prodotto per visualizzarne dettagli, descrizioni, immagini e prezzo.
21. **Salvataggio preferiti:** L'utente può aggiungere articoli alla lista dei desideri o "preferiti" per acquistarli in un secondo momento.
22. **Gestione resi e reclami:** L'amministratore può ricevere e gestire richieste di reso, rimborsi e reclami, garantendo un processo semplice e veloce per i clienti. (amministratore)
23. **Dark Mode:** L'utente può attivare la modalità scura per un'esperienza di navigazione più comoda in ambienti poco illuminati.

3.3 Modello Kano

Il modello Kano classifica i requisiti di un prodotto o servizio in base alla loro capacità di influenzare la soddisfazione del cliente, suddividendoli in sei categorie distinte:

i. **Must-Be:** requisiti essenziali.
ii. **Attractive:** requisiti interessanti.
iii. **One-Dimensional:** soddisfazione del cliente proporzionale al numero di requisiti soddisfatti.
iv. **Indifferent:** requisiti che non influenzano la soddisfazione del cliente.

v. **Reverse:** requisiti la cui soddisfazione del cliente è opposta a quella prevista. Nel sistema non sono previsti requisiti in questa categoria, in quanto punta a migliorare l'esperienza dell'utente.

vi. **Questionable:** preferenze del cliente non chiare.

TIPOLOGIA	**REQUISITI**
MUST BE	1, 2, 4, 7 12
ATTRACTIVE	3, 11, 14, 16, 18
ONE DIMENSIONAL	5, 6, 8, 9, 10 13, 15, 22
INDIFFERENT	20, 23
REVERSE	
QUESTIONABLE	17, 29, 21

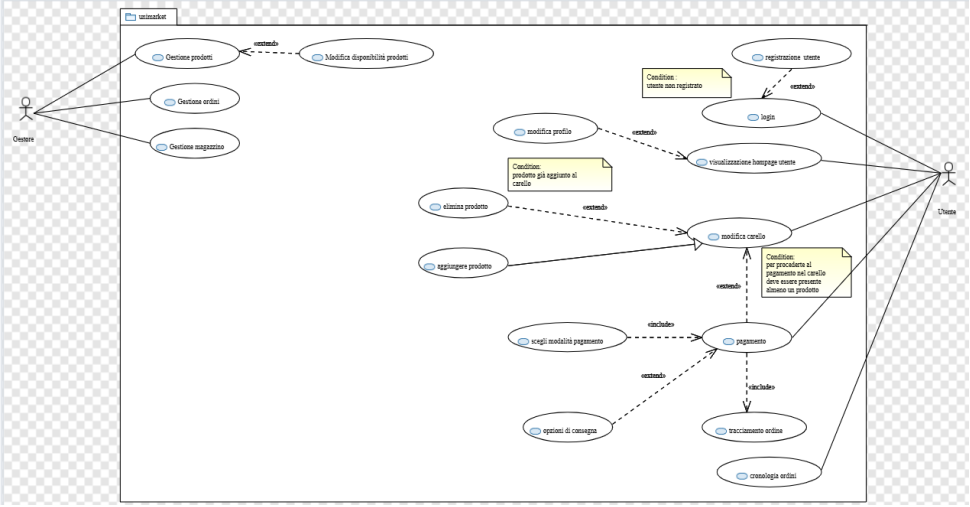
4 Design

4.1 UML

Di seguito i diagrammi UML; per una visione migliore, fare riferimento alle immagini contenute nel repository UML.

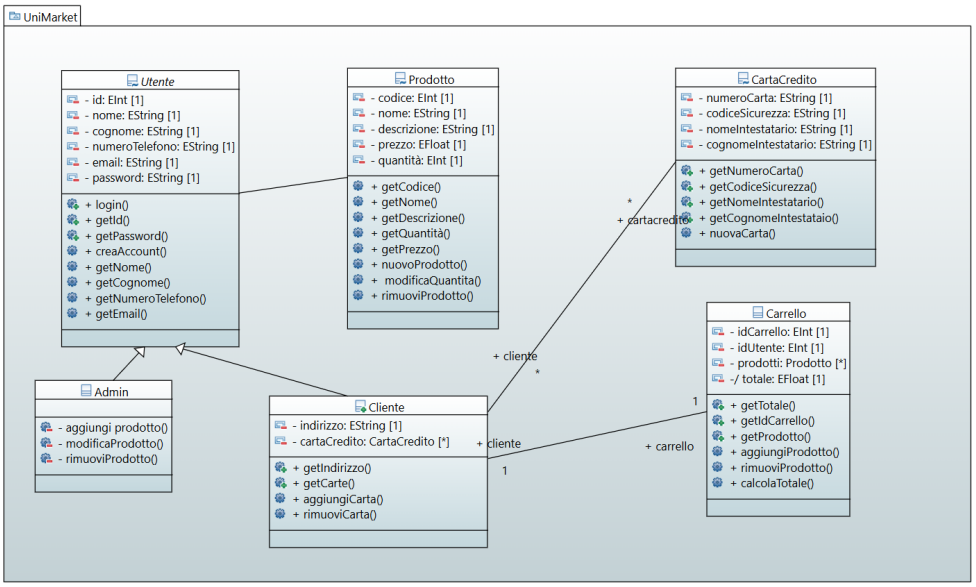
4.1.1 Use Case Diagram

Il diagramma dei casi d'uso modella l'interazione tra gli attori (Utenti e gestori) e il sistema, mostrando le funzionalità supportate.

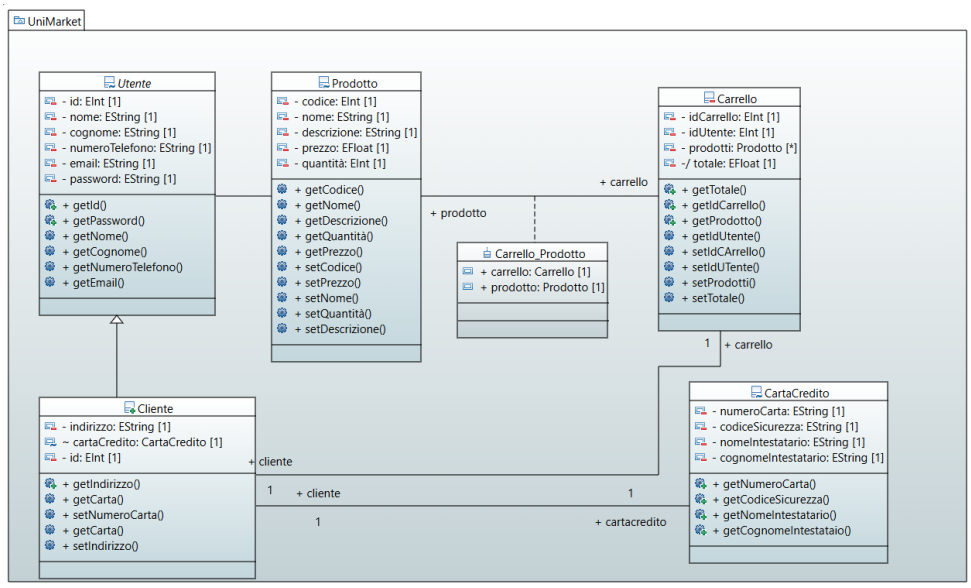


4.1.2 Class Diagram

Il diagramma delle classi modella le classi contenute nel sitema, con i relativi metodi e attributi.

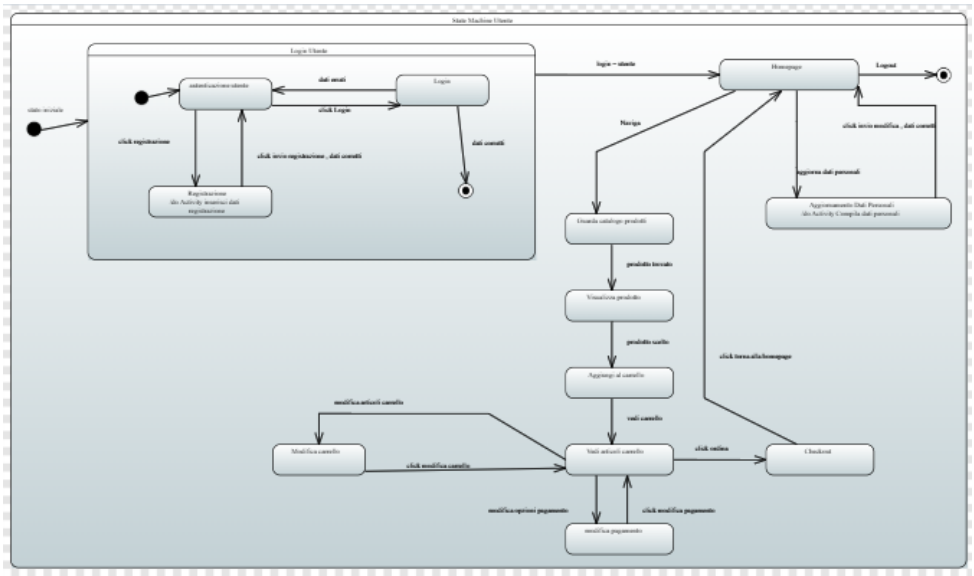


Le classi hanno subito varie evoluzioni durante lo sviluppo del progetto, il diagramma seguente mostra lo stato delle classi al momento della prima pubblicazione del sistema:



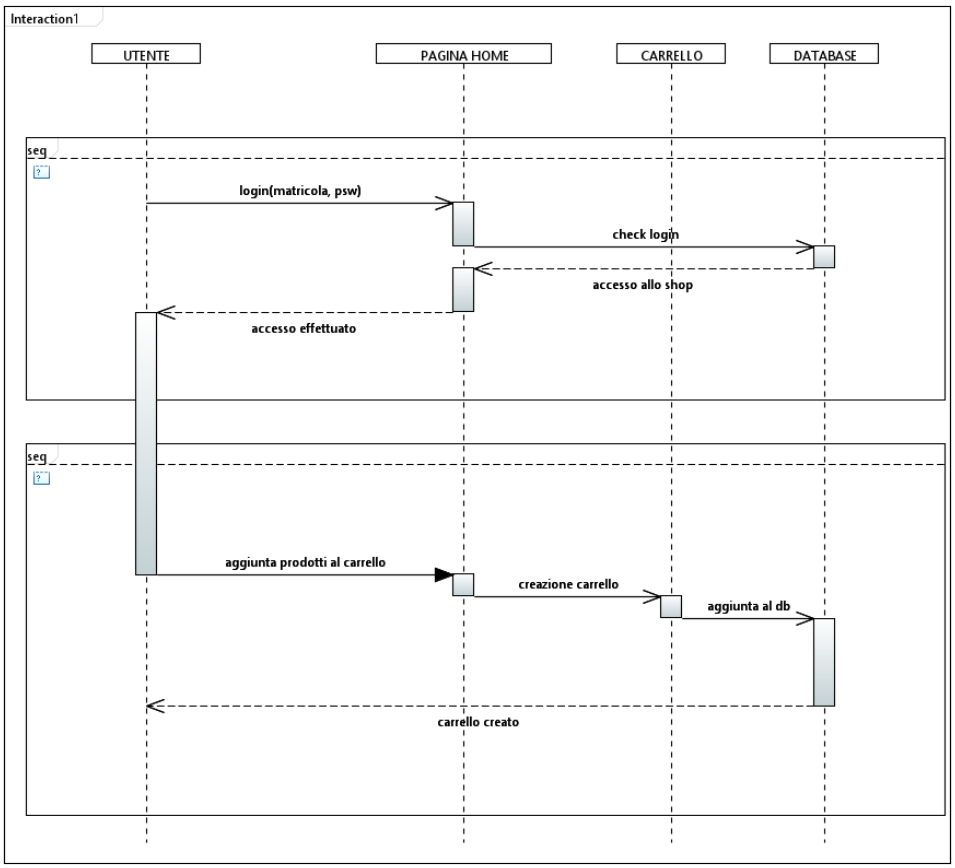
4.1.3 State Machinne

Il diagramma degli stati modella il comportamento di un oggetto in base allo stato in cui si trova. Nello specifico, il seguente diagramma modella gli stati attivi durante l'utilizzo del sistema da un utente.



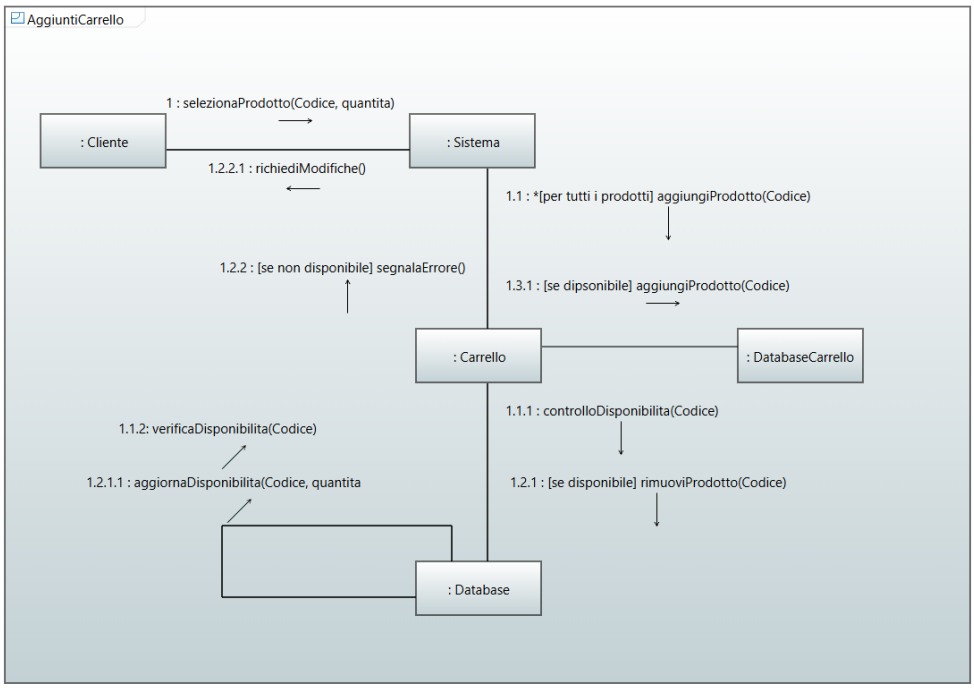
4.1.4 Sequence Diagram

Il diagramma delle sequenze modella l'interazione tra gli oggetti del sistema, mostrando l'ordine in cui i messaggi vengono scambiati. Nel diagramma viene mostrata l'interazione tra l'utente e il sistema nelle operazioni di login e agiunta di prodotti al carrello.



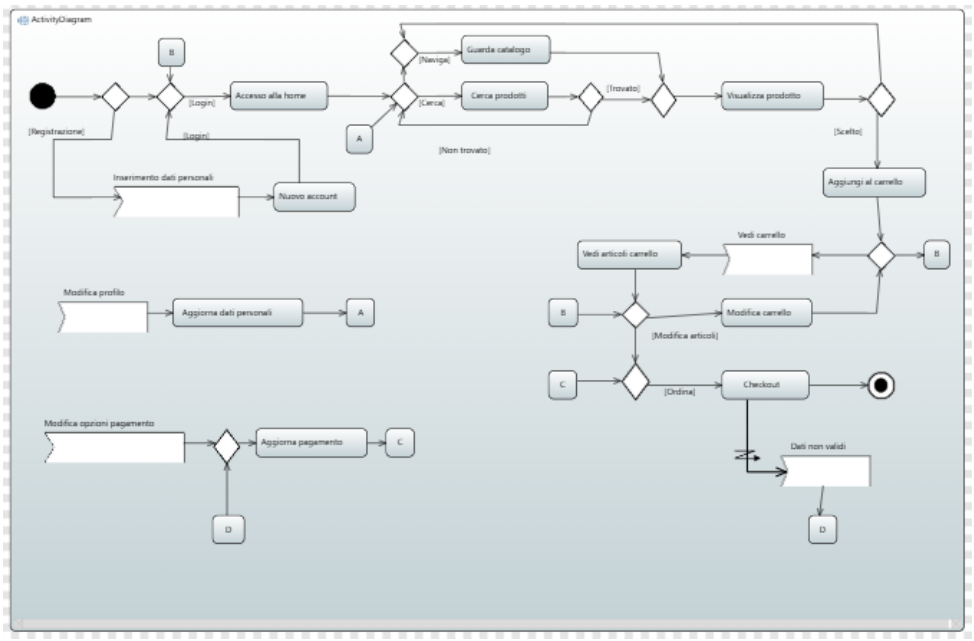
4.1.5 Communication Diagram

Il diagramma di comunicazione modella lo scambio di messaggi tra le componenti del sistema durante l'aggiunta, da parte dell'utente, di prodotti al carrello.



4.1.6 Activity Diagram

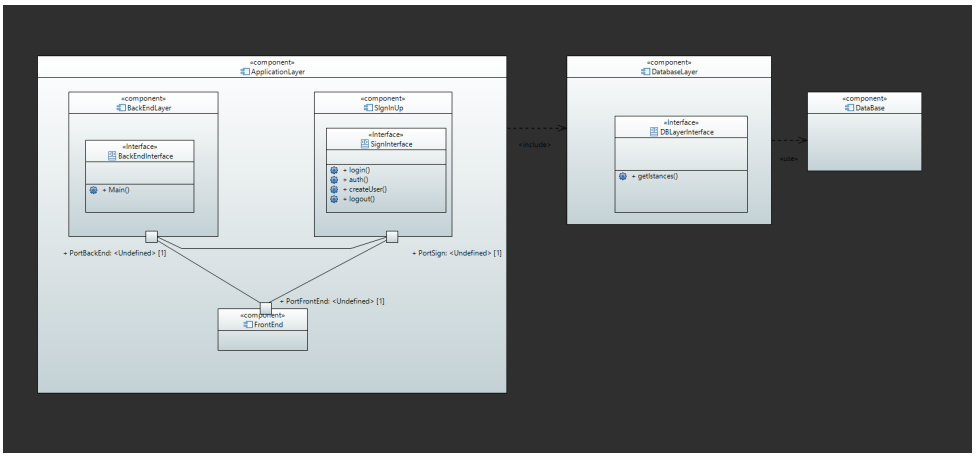
Il diagramma delle attività modella il flusso di lavoro all'interno del sistema, mostrando le attività e le decisioni che vengono prese durante l'interazione con l'applicazione. Il diagramma seguente mostra il flusso di lavoro per l'aggiunta di un prodotto al carrello.



Nota: i connettori(A,B,C,D) sono stati rappresentati usando blocchi azione per l'assenza di un blocco dedicato da parte di Papyrus.

4.1.7 Component Diagram

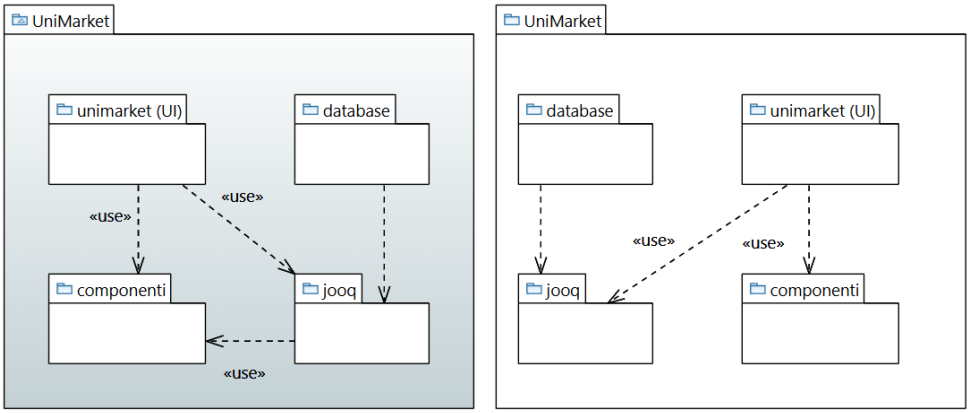
Il diagramma delle componenti ha lo scopo di mostrare le relazioni tra le diverse componenti di un sistema. Nel diagramma seguente vengono mostrate le relazioni tra i tre livelli del sistema.



4.1.8 Package Diagram

Il diagramma dei pacchetti mostra la struttura del programma e le interazioni tra i vari package.

Nel digramma riportato, a sinistra è il diagramma concettuale, mostra l'architettura ideata all'inizio dello sviluppo; a destra la rappresentazione effettiva delle interazioni, fornita da un'analisi effettuata con STAN.



4.2 Architettura del software

4.2.1 Viste architettoniche

Le viste architettoniche e i punti di vista utilizzati sono:

- **Modulo:** È stato utilizzato il punto di vista **a strati**, che permette di visualizzare il sistema su livelli, i cui elementi possono utilizzare elementi di un livello inferiore.
- **Componenti e connettori:** È stato utilizzato il punto di vista **client-server**, che descrive un sistema costituito da client e server cooperanti. In questo progetto, il client rappresenta l'interfaccia utente dell'applicazione e il server gestisce la logica di business e l'accesso ai dati.

- **Realizzativa:** È stato utilizzato il punto di vista dell'**implementazione**, che descrive come i componenti logici e le funzionalità di alto livello vengono tradotti in strutture fisiche e tecniche.
  - Il progetto è organizzato seguendo un approccio **maven**, con una struttura standard: **src/main/java** per il codice sorgente, **src/main/resources** per le risorse, **src/test/java** per i test.
  - I framework utilizzati sono: **Vaadin** per l'interfaccia utente, **JOOQ** per l'accesso ai dati, **SQLite** per il database.

4.2.2 Stile architettonico

Il progetto è basato sullo stile architettonico a **tre livelli**, che prevede la suddivisione del sistema in tre strati distinti: livello database, livello di persistenza e livello di presentazione. Questo approccio consente di separare le funzionalità e i compiti del sistema, semplificando la gestione e la manutenzione del codice.

- **Livello Database:** Il livello database gestisce la memorizzazione e l'accesso ai dati, garantendo la persistenza delle informazioni. In questo progetto, è implementato un *database embedded* utilizzando *SQLite*.
- **Livello di Persistenza:** Il livello di persistenza si occupa di gestire la comunicazione tra il livello database e il livello di presentazione, fornendo un'interfaccia per l'accesso ai dati. In questo progetto, il livello di persistenza è implementato utilizzando *JOOQ*, un framework Java per la creazione di query SQL.
- **Livello di Presentazione:** Il livello di presentazione gestisce l'interfaccia utente e l'interazione con l'utente. In questo progetto, il livello di presentazione è implementato utilizzando *Vaadin*, un framework Java per lo sviluppo di applicazioni web.

4.2.3 Librerie esterne

Il progetto utilizza diverse librerie per gestire l'interfaccia utente, il database, la persistenza dei dati e il logging. Di seguito sono descritte alcune delle più rilevanti.

- **Vaadin:** Framework Java per lo sviluppo di applicazioni web, utilizzato per la creazione dell'interfaccia utente. Fornisce componenti UI reattivi e un'integrazione fluida con Spring Boot.Fornisce componenti UI reattivi e un'integrazione fluida con Spring Boot.
- **JOOQ:** Framework Java per la creazione di query SQL, utilizzato per la gestione della persistenza dei dati.
- **SQLite:** Database embedded utilizzato per la memorizzazione dei dati.
- **log4j e slf4j:** Framework per la gestione dei log, utilizzati per il debugging e per registrare eventi e informazioni sul funzionamento dell'applicazione.
- **hibernate** *Object-Relational Mapping* (ORM) framework, utilizzato per la gestione della persistenza dei dati. Permette l'integrazione con Spring Data JPA per un accesso semplificato ai dati.

4.3 Software Design

4.3.1 Complessità e misurazioni

Analisi strutturale

L'analisi strutturale del codice ha lo scopo di valutare la complessità e la qualità del codice sorgente, identificando eventuali punti deboli e aree di miglioramento. Questo processo include la misurazione di metriche chiave come la complessità ciclomatica, la coesione e il grado di accoppiamento tra i moduli.

Nel progetto è stata svolta un'analisi strutturale tramite **STAN**, che permette di valutare le interazioni e dipendenze tra classi e pacchetti. Tramite STAN è stato possibile individuare cicli interni al sistema e risolverli.

Ad esempio:

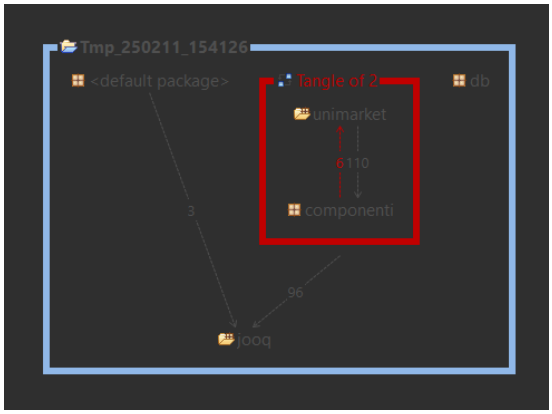


Figure 1: Implementazione scorretta dei pattern

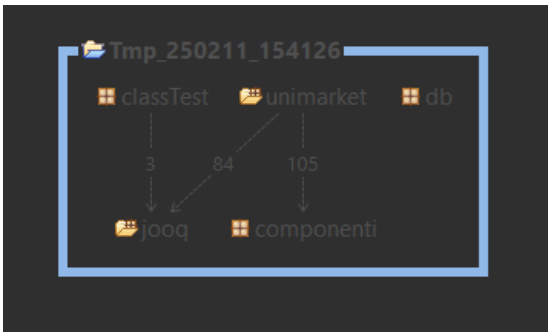


Figure 2: Pacchetti corretti

L'analisi ha permesso di identificare un ciclo tra i pacchetti *unimarket*, il livello di presentazione, e *componenti*. Questo errore era causato dalla scorretta implementazione dei pattern. Una volta individuato è stato possibile correggerlo per ottenere un'interazione senza cicli:

4.3.2 Design Patterns

I pattern utilizzati nel progetto sono:

- **Factory pattern**: Implementato tramite il servizio *CarrelloService*, permette di garantire l'associazione di un carrello ad ogni utente al momento del login; se un utente ha già un carrello, questo viene recuperato dal database, altrimenti ne viene creato uno. Permette di evitare duplicazioni, migliora la manutenibilità e separa la logica dalla gestione.
- **Observer pattern**: Implementato tramite due interfacce *Observer*: una per la gestione e una per l'interfaccia. L'oggetto osservato è il *Carrello*, tramite la gestione e notifica degli observer in *CarrelloService*, permette di aggiornare in tempo reale le classi e interfacce che ne mostrano il contenuto.

5 Testing

5.1 Testing e Verifica del Sistema

Durante il ciclo di vita del progetto, oltre allo sviluppo del codice e alla scrittura della documentazione, un ruolo fondamentale è stato svolto dalla fase di testing. L'obiettivo principale è stato garantire la correttezza e l'affidabilità del sistema, prevenendo errori e malfunzionamenti che avrebbero potuto comprometterne l'utilizzo.

È noto che molti problemi emergono nelle prime fasi dello sviluppo software, ma se individuati in ritardo possono risultare più complessi e costosi da correggere. Per questo motivo, il testing è stato integrato sin dalle prime fasi del progetto, adottando un approccio progressivo e iterativo.

5.2 Testing in fase di sviluppo

La prima fase del testing è avvenuta in parallelo con la scrittura del codice. Per verificare il corretto funzionamento delle funzionalità integrate, sono stati adottati strumenti di logging e controlli temporanei, facilitando il debugging e l'individuazione tempestiva di errori. Il controllo è stato eseguito manualmente, testando direttamente le funzionalità con valori di input specifici e verificando il comportamento del sistema tramite i log di esecuzione.

5.3 Testing automatizzato

Per garantire una maggiore copertura del sistema e validare il corretto funzionamento delle diverse componenti, è prevista l'implementazione di test automatici. L'obiettivo è scrivere test unitari e di integrazione che verifichino il comportamento del codice in modo sistematico. Tuttavia, per ragioni di tempistiche, questa fase è stata parzialmente posticipata e non è ancora completamente integrata nel processo di sviluppo.

L'implementazione dei test automatici avviene all'interno della directory *src/test/java* utilizzando JUnit, un framework per il testing unitario in Java. I test vengono scritti per verificare il corretto funzionamento dei metodi e delle classi, simulando l'input e confrontando l'output con quello atteso.

Un esempio di test automatico implementato è la classe di test *CartaServiceTest*, che verifica il metodo *salvaCarta*. Questo metodo si occupa di inserire nel database un nuovo oggetto *CartaCredito*, salvando i dati del numero di carta, codice di sicurezza e intestatario. Il test controlla che l'inserimento avvenga correttamente e gestisce eventuali errori di persistenza.

6 Maintenance

6.1 Manutenzione del Software

La manutenzione del software comprende diverse attività mirate a garantire il corretto funzionamento del sistema nel tempo, migliorandone stabilità, efficienza e adattabilità. Si suddivide in quattro categorie principali:

1. **Manutenzione correttiva**: riguarda la risoluzione di bug e malfunzionamenti emersi durante lo sviluppo, come incompatibilità tra i tipi di dati gestiti da Java e quelli supportati dal database.
2. **Manutenzione adattiva**: consiste nell'aggiornamento del software per adeguarlo a cambiamenti tecnologici o nuove esigenze operative. Pur non essendo ancora necessaria in questa fase, potenziali interventi futuri potrebbero includere l'ottimizzazione delle prestazioni, l'aggiornamento delle librerie o l'adattamento a nuove piattaforme.
3. **Manutenzione perfettiva**: si concentra sul miglioramento delle funzionalità esistenti e sull'introduzione di nuove feature per ottimizzare l'interazione con il sistema.
4. **Manutenzione preventiva**: ha l'obiettivo di ridurre il rischio di problemi futuri attraverso verifiche periodiche, mantenendo test e log aggiornati per semplificare la gestione del software senza complicarne inutilmente la struttura.

6.2 Best Practices

Per garantire una manutenzione efficace e prevenire problemi derivanti da codice disorganizzato o documentazione insufficiente, sono state seguite le seguenti linee guida:

- **Commenti e documentazione**:Commentare il codice per facilitarne la comprensione e la futura modificabilità.
- **Versionamento**: Utilizzare un sistema di versionamento per tenere traccia delle modifiche e facilitare il rollback in caso di errori.
- **Revisione**: Sottoporre le modifiche alla revisione del team per migliorare la qualità del codice e ridurre il rischio di errori.
- **Layering**: Strutturare il codice in moduli e servizi distinti, favorendo la separazione delle responsabilità e semplificando eventuali interventi futuri.

Queste attività permettono di mantenere un software facilmente gestibile e modificabile, riducendo il rischio di errori e semplificando la manutenzione nel tempo.

6.3 Refactoring

Il refactoring non è stato concentrato in una singola fase, ma è stato integrato durante tutto il processo di sviluppo con modifiche graduali per garantire un codice più leggibile e manutenibile. Ogni intervento ha interessato sezioni specifiche del sistema, riducendo il rischio di conflitti e garantendo il funzionamento complessivo.

Un'area chiave di intervento è stata l'ottimizzazione del codice autogenerato da Vaadin, inizialmente disordinato e poco comprensibile. È stata migliorata la struttura delle classi, separando le responsabilità e rendendo il codice più chiaro e facile da mantenere.

Altri interventi hanno riguardato la gestione dell'interfaccia utente, con particolare attenzione al flusso di navigazione, al sistema di login e alle interfacce riservate agli amministratori per il controllo dei dati salvati.

6.4 miglioramenti futuri

Alcune funzionalità non sono ancora state implementate, ma potrebbero rappresentare possibili sviluppi futuri per arricchire il sistema. Tra queste:

- **Gestione dei prodotti**: migliorare la gestione dei prodotti da parte degli amministratori, con la possibilitià di modificarne i dettagli.
- **Logout**: fornire l'opzione di logout per poter cambiare account.
- **Modificabilità account**: permettere agli utenti di eliminare il proprio account o modificare i dati salvati.

L'integrazione di queste funzionalità potrà migliorare ulteriormente l'esperienza d'uso del sistema.

Calendario Timebox

SETTIMANA/E	OBIETTIVO	INCONTRI
4/11/2024 - 17/11/2024	Project Plan	6
18/11/2024 - 24/11/2024	Documentazione, gestione del progetto	3
25/11/2024 - 29/11/2024	Documentazione, Specifica dei requisiti	3
2/12/2024 - 5/2/2025	UML	2
6/1/2025 - 26/1/2025	Database, documentazione	1
27/1/2025 - 2/2/2025	Interfaccia	2
3/2/2025 - 11/2/2025	Interfaccia, manutenzione, documentazione	6