Gruppo NC24

Corso di IS

Prof. C. Gravino

aa 2025-2026

Team Members

Mario Branca

Simone Sammartano

Paolo Visconti

Gabriele De Luca

Scheduling e comunicazione

- **Google Drive e Github**

  Abbiamo usato Google Drive per lavorare insieme alla documentazione in tempo reale, mentre GitHub è stato il punto di riferimento per la gestione del codice tramite una repository condivisa.

- **Trello**

  Per la pianificazione dei task abbiamo scelto Trello, che si è rivelato utile per assegnare e suddividere le attività tra i membri del team e monitorarne l'avanzamento.

- **Discord e Whatsapp**

  Per la comunicazione abbiamo utilizzato principalmente le chiamate vocali su Discord per riunirci e lavorare insieme a distanza, mentre WhatsApp è servito per aggiornamenti e comunicazioni rapide.



Documentazione

La fase di documentazione è stata la parte centrale del lavoro: da qui abbiamo costruito le basi su cui poi sviluppare tutto il progetto. Avere documenti chiari ci ha permesso di andare avanti con le fasi successive in modo più lineare e sicuro, senza perdere di vista i requisiti individuati e con un riferimento costante durante la realizzazione dei vari componenti dell'applicazione.

Documentazione Readify

Cos'è Readify?

Readify è un e-commerce dedicato ai libri, progettato per rendere l'acquisto online più semplice, rapido e accessibile.
La piattaforma permette agli utenti di consultare un catalogo organizzato, trovare facilmente i titoli desiderati tramite ricerca e filtri, e allo stesso tempo scoprire nuove letture in modo intuitivo.
Il processo di acquisto è pensato per essere lineare: in pochi passaggi l'utente può selezionare i libri, inserirli nel carrello, completare l'ordine e riceverli direttamente a casa, con feedback chiari lungo tutto il percorso.

Obiettivi del sistema

Ci siamo posti come obiettivi quelli di coprire tutte le funzionalità necessarie per la piattaforma, in particolare:

- Catalogo con ricerca filtrata
- Acquisto di prodotti
- Gestione del carrello
- Funzionalità amministrative
- Modifica del profilo

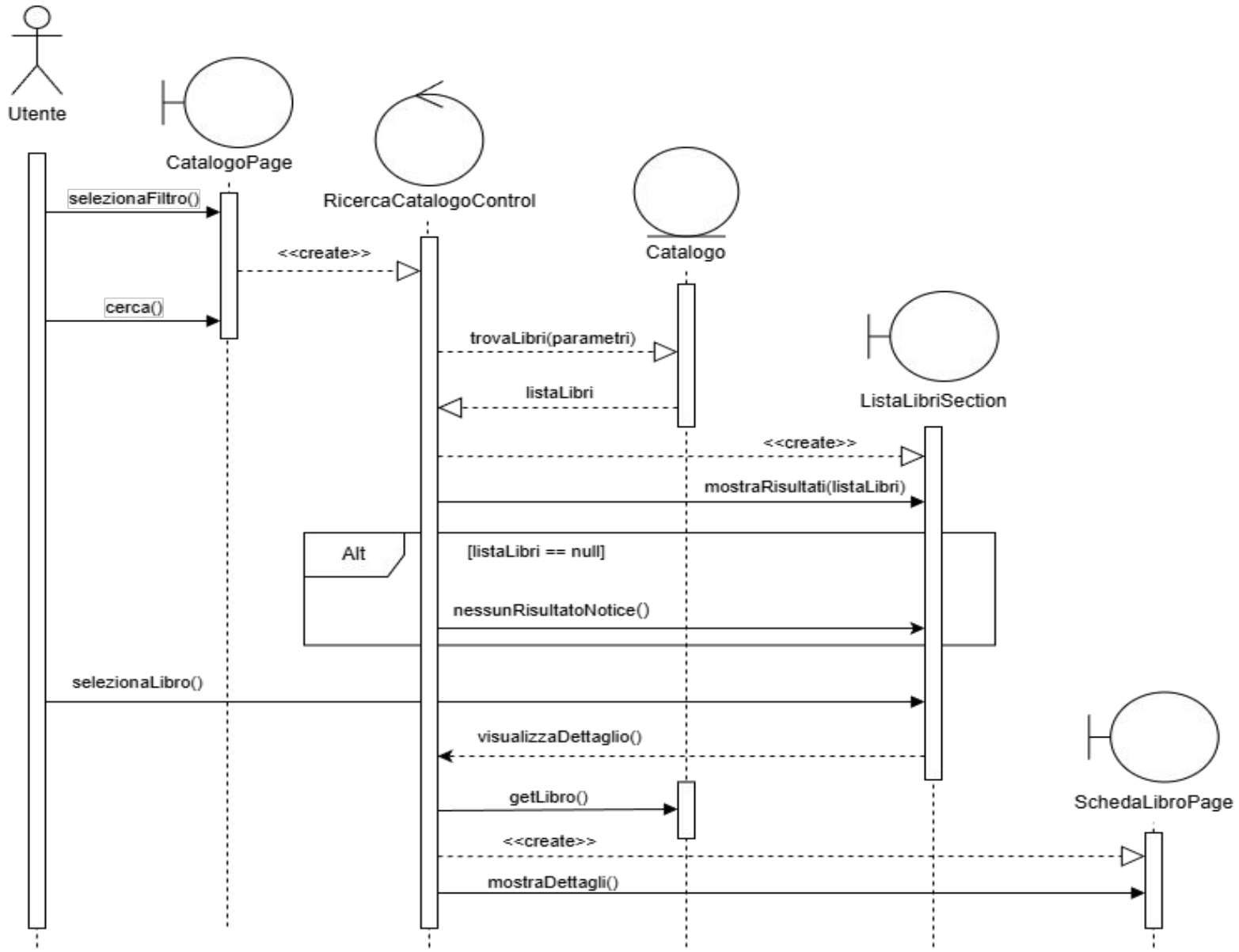
Casi d'uso

A partire dai requisiti individuati nella fase di raccolta e analisi, abbiamo definito e organizzato i principali casi d'uso del sistema, che descrivono le interazioni fondamentali tra gli utenti e la piattaforma.

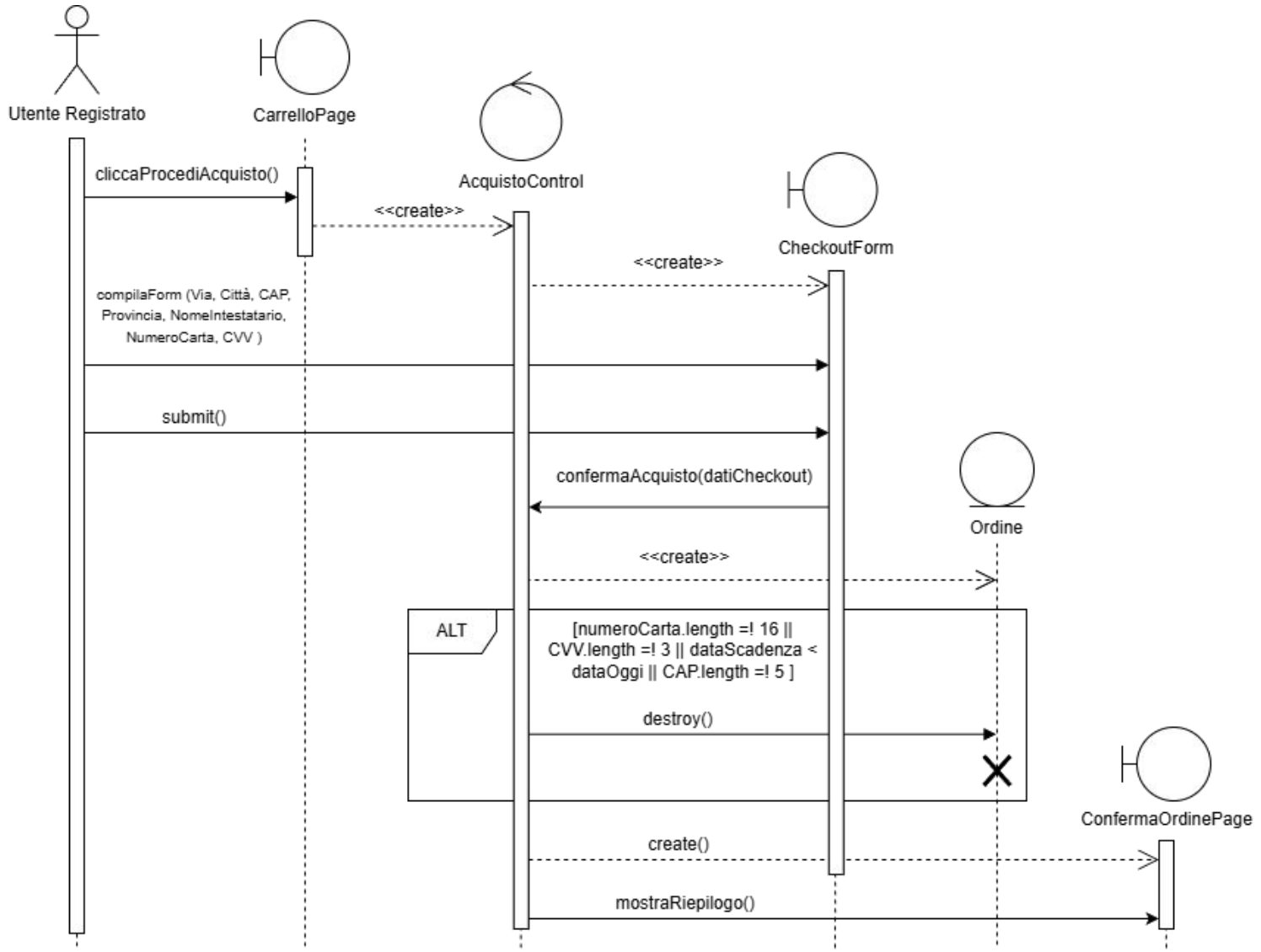
- **UC1_REG** - Registrazione utente
- **UC2_RLC** - Ricerca di un libro nel catalogo
- **UC3_ACQ** - Acquisto di uno o più libri
- **UC4_ALC** - Aggiunta di un libro al catalogo

Diagramma dei casi d'uso

SD_UC2_RLC

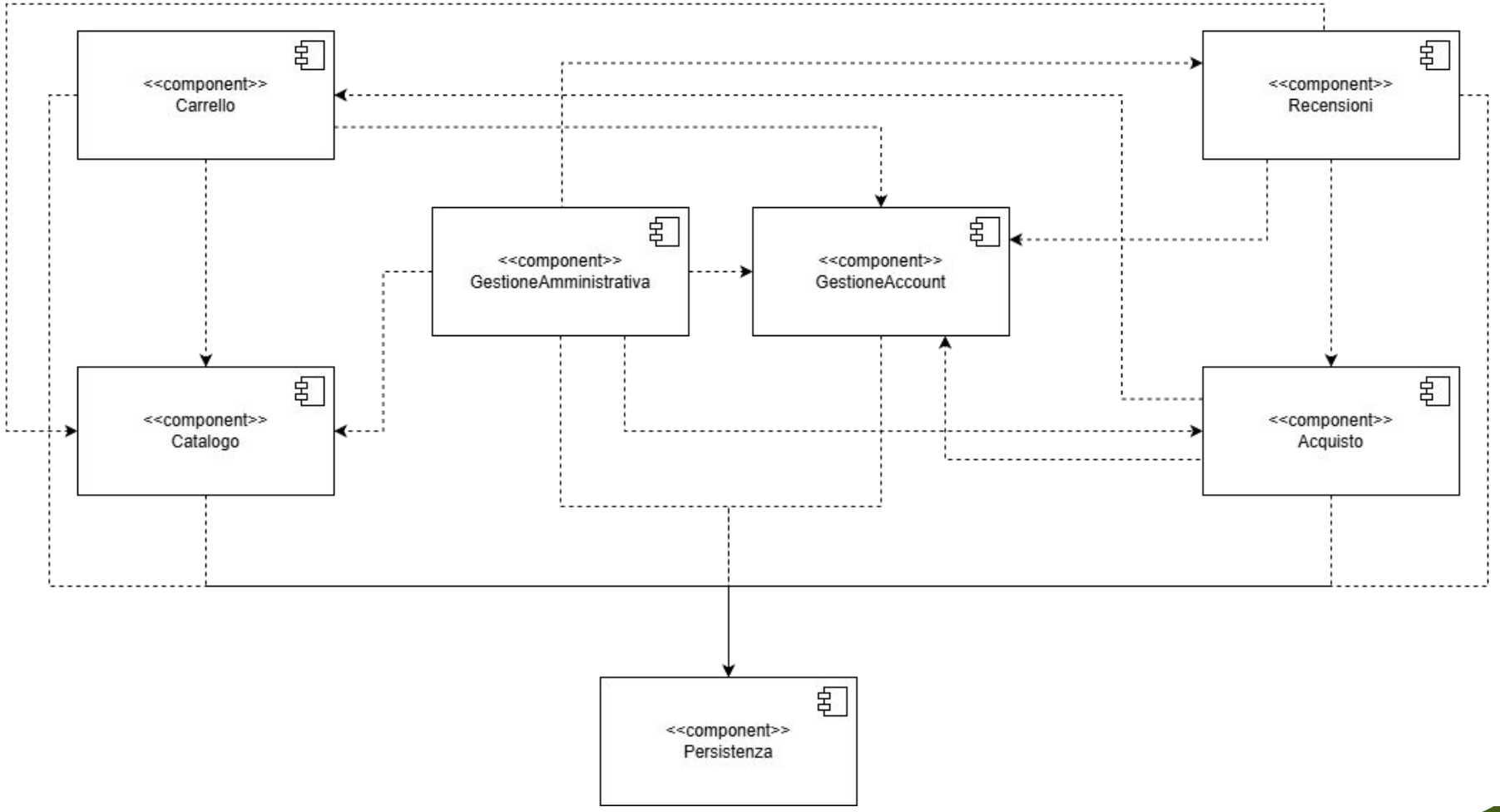

Utente
CatalogoPage
RicercaCatalogoControl
Catalogo
ListaLibriSection
SchedaLibroPage
selezionaFiltro()
<<create>>
cerca()
trovaLibri(parametri)
listaLibri
<<create>>
mostraRisultati(listaLibri)
Alt
[listaLibri == null]
nessunRisultatoNotice()
selezionaLibro()
visualizzaDettaglio()
getLibro()
<<create>>
mostraDettagli()

SD_UC3_ACQ

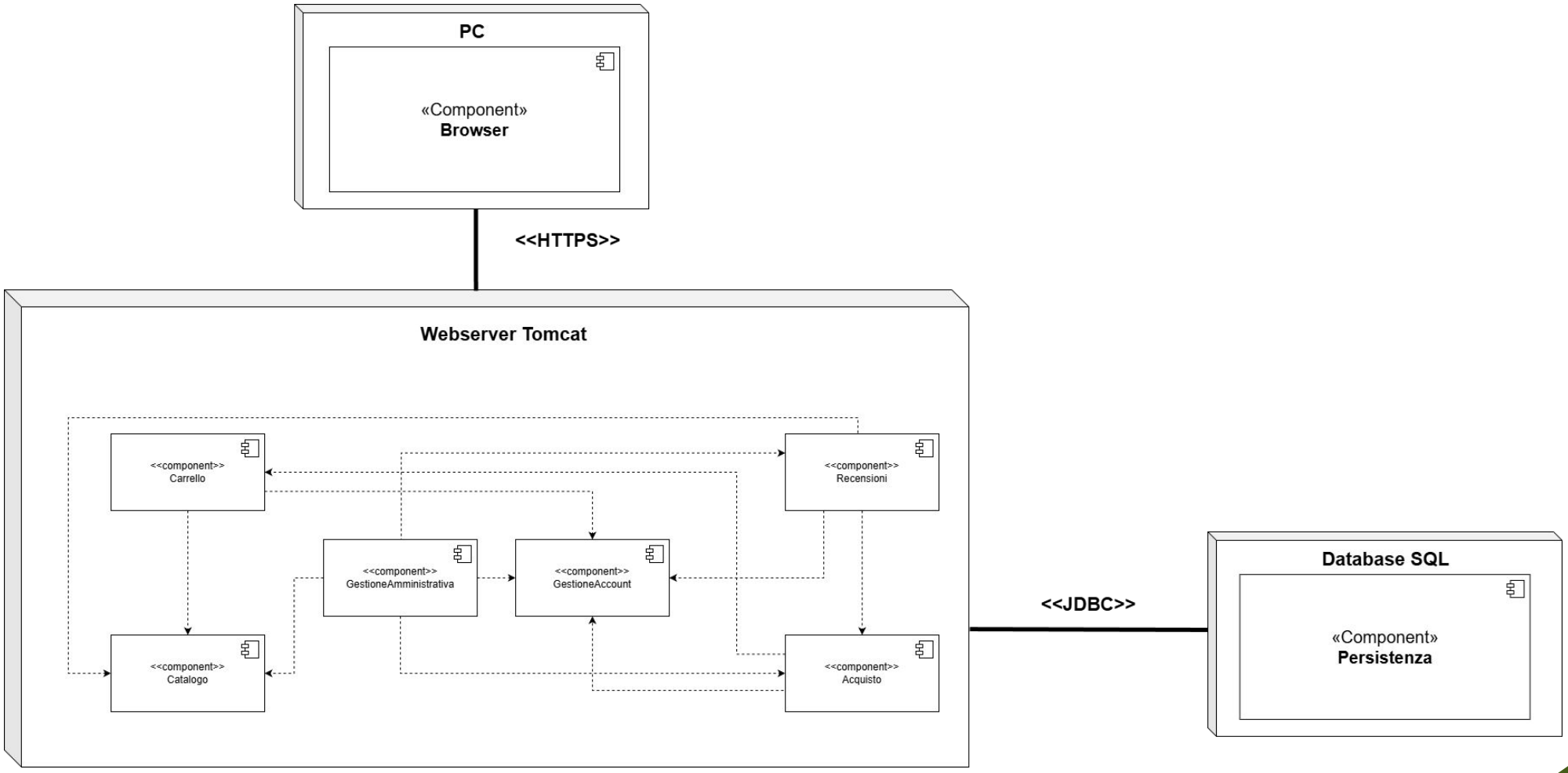

Utente Registrato
CarrelloPage
AcquistoControl
CheckoutForm
Ordine
ConfermaOrdinePage
cliccaProcediAcquisto()
<<create>>
<<create>>
compilaForm (Via, Città, CAP, Provincia, NomeIntestatario, NumeroCarta, CVV)
submit()
confermaAcquisto(datiCheckout)
<<create>>
ALT
[numeroCarta.length =! 16 || CVV.length =! 3 || dataScadenza < dataOggi || CAP.length =! 5]
destroy()
create()
mostraRiepilogo()

Sottosistemi del progetto

In seguito alla raccolta e analisi dei requisiti abbiamo suddiviso le funzionalità principali individuate in 7 sottosistemi:



Deployment Diagram



Scelta dell'architettura

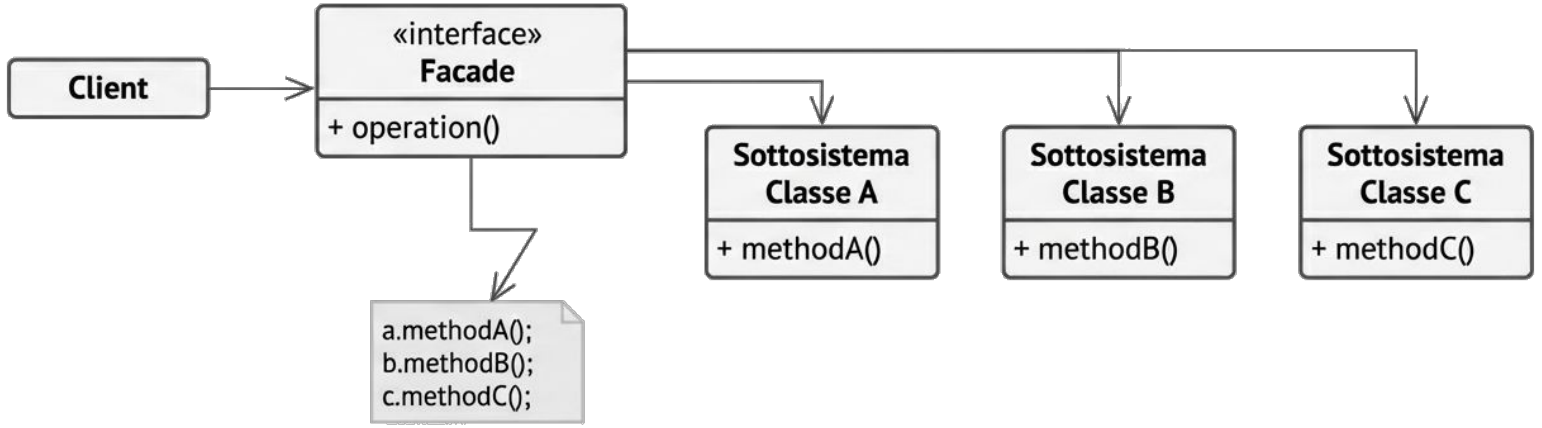
Abbiamo scelto un'architettura **Three-Tier** perché offre una struttura più chiara e organizzata del sistema. In particolare:

- migliora la **manutenibilità**, perché eventuali modifiche restano circoscritte a uno specifico livello;
- semplifica la **gestione dei ruoli** e dei **controlli di accesso**, concentrandoli nella logica applicativa;
- rende il sistema più **scalabile**, lasciando aperta la possibilità di distribuire i livelli su nodi distinti;
- facilita l'**evoluzione futura**, ad esempio integrando nuove interfacce o servizi esterni senza stravolgere l'intera applicazione.

Design Pattern: Facade

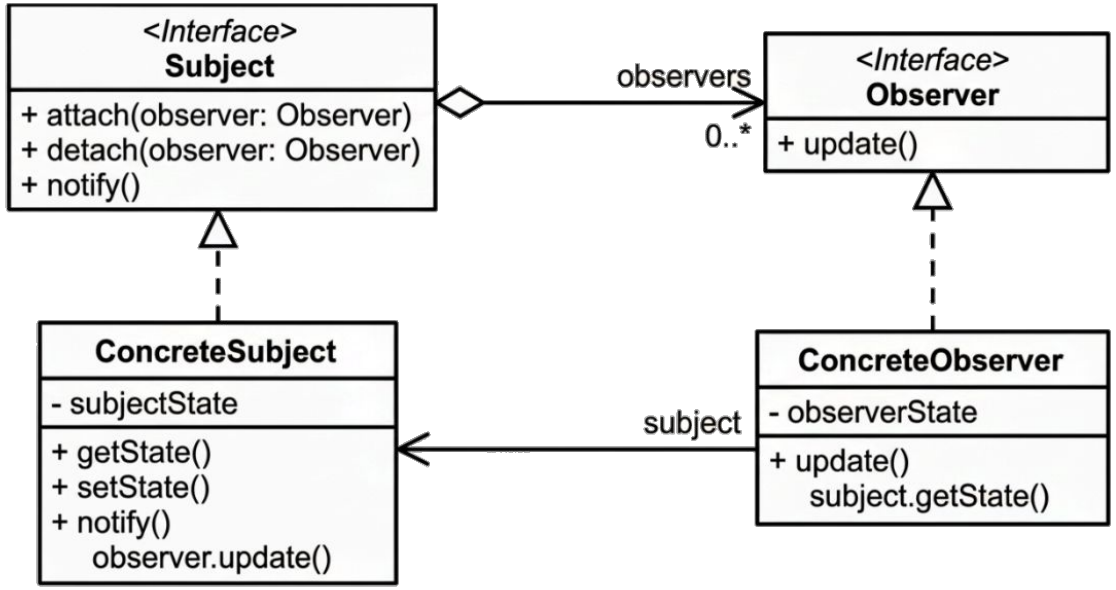
IL Facade è un design pattern che serve a semplificare l'accesso a un sottosistema complesso. Offre un'unica interfaccia attraverso cui utilizzare funzionalità che internamente coinvolgono più classi, nascondendo i dettagli e riducendo la complessità per il resto dell'applicazione.

Nel nostro sistema lo abbiamo applicato alla gestione del carrello: invece di far interagire direttamente i controller con le diverse componenti del sottosistema, abbiamo introdotto una Facade che espone un unico punto di accesso e si occupa di gestire tutta la logica del carrello. In questo modo centralizziamo il comportamento in una sola classe e manteniamo il resto dell'applicazione più ordinato e semplice da mantenere.



Design Pattern: Observer

L'Observer è un design pattern comportamentale che definisce una dipendenza uno-a-molti tra oggetti, in modo tale che quando un oggetto cambia stato, tutti gli oggetti che dipendono da esso vengano notificati automaticamente.
Abbiamo scelto di usare l'Observer per propagare automaticamente le modifiche del catalogo su tutti i carrelli persistiti, mantenendo consistenza dei dati senza spargere logica di aggiornamento in più punti del codice.



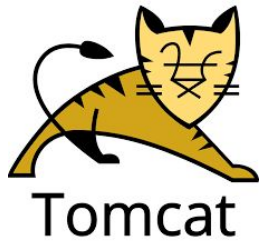
Implementazione

Avendo a disposizione un sistema già implementato, abbiamo scelto di partire da ciò che era presente, migliorando e ampliando le funzionalità esistenti e introducendo quelle emerse durante la fase di raccolta e analisi dei requisiti. In particolare ci siamo concentrati su:

- **Registrazione**
- **Login**
- **Ricerca nel catalogo**
- **Gestione del carrello**
- **Gestione Ordine**
- **Gestione Amministrativa**

Tecnologie utilizzate

- Server locale eseguito tramite **Tomcat**
- Database MySQL hostato in rete tramite **Microsoft Azure**
- **JUnit**, **Mockito** e **Jacoco** per il testing
- Implementazione gestita tramite repository **GitHub**, organizzando il lavoro tramite branch e pull request per integrare le modifiche in modo controllato.




Testing

Nella fase di testing ci siamo concentrati sui seguenti sottosistemi, ritenuti i più importanti perché sono quelli che gestiscono il maggior numero di input e interazioni, ovvero:

- **Gestione Account**
- **Catalogo**
- **Acquisto**

Per queste funzionalità abbiamo raggiunto l'obiettivo che ci eravamo prefissati, ottenendo almeno il 50% di branch coverage.

Cosa è andato bene?

- **Completezza del sistema:** le funzionalità principali previste in fase di analisi sono state implementate e integrate in modo coerente tra loro.
- **Raggiungimento degli obiettivi:** i requisiti e i vincoli principali definiti inizialmente sono stati rispettati.
- **Organizzazione del lavoro:** il coordinamento del team è stato efficace; siamo riusciti a suddividere i task e a gestire anche i ritardi senza compromettere l'avanzamento del progetto.

Cosa è andato male?

- **Gestione dei tempi morti:** in alcune fasi non siamo riusciti a distribuire il lavoro in modo uniforme, concentrando diverse attività in periodi brevi e generando piccoli ritardi.
- **Obiettivi fuori scala:** inizialmente alcune funzionalità erano state pensate in modo troppo ambizioso rispetto al tempo e alle risorse disponibili, e abbiamo dovuto ridimensionarle in corso d'opera.
- **Gestione del database online:** la scelta di hostare il database su Azure ha richiesto più tempo del previsto, soprattutto per le difficoltà legate alla configurazione di una tecnologia completamente nuova per noi.

Cosa avremmo fatto diversamente?

- **Maggiore flessibilità nella pianificazione:** avremmo lasciato più margine per eventuali imprevisti, evitando di concentrare troppe attività nelle fasi finali.
- **Implementazione del sistema:** partire da un sistema già implementato, che abbiamo deciso di modificare in gran parte, si è rivelato più complesso del previsto. L'integrazione delle nuove funzionalità ha richiesto di risolvere diversi conflitti tra componenti già presenti e quelle introdotte, rendendo il lavoro più articolato rispetto a uno sviluppo da zero.
- **Sistema di pagamento esterno:** avremmo voluto implementarlo in modo concreto, integrando servizi come Stripe per rendere l'acquisto più realistico. Tuttavia, per contenere il carico di lavoro ed evitare ulteriori ritardi, abbiamo scelto di simularlo. Resta comunque un'integrazione che consideriamo come possibile sviluppo futuro del progetto.

Come valutiamo il nostro lavoro?

Nel complesso siamo soddisfatti del risultato raggiunto. Sappiamo di non aver ottenuto un risultato eccellente, sia nella stesura della documentazione sia nello sviluppo del codice, e che ci sono aspetti che avremmo potuto curare meglio. Nonostante questo, siamo riusciti a rispettare gli obiettivi prefissati e a portare a termine un sistema coerente con i requisiti definiti.

Lezioni imparate

- **Lavoro di squadra:** abbiamo capito quanto sia importante coordinarci come team, valorizzando le competenze di ciascuno per raggiungere un obiettivo comune e affrontare insieme le difficoltà.
- **Pianificazione:** ci siamo resi conto di quanto sia fondamentale organizzare il lavoro in anticipo, definendo obiettivi chiari e scadenze realistiche per rispettare i tempi e mantenere la qualità.
- **Documentazione:** abbiamo compreso che una documentazione curata non serve solo durante lo sviluppo, ma è indispensabile per rendere il sistema chiaro e gestibile anche nel tempo.

In sintesi, questo progetto ci ha aiutato a sviluppare un metodo di lavoro più maturo e organizzato che porteremo con noi nei progetti futuri.

Grazie per l'attenzione!

Readify