

Applicativo per la gestione di un B2B di abbigliamento

Ingegneria del software a.a. 2023/2024

Leonardo Borsi



Sommario

Questo è il documento di progetto riguardante la realizzazione di un applicativo software per la gestione di un B2B di abbigliamento

Indice

1 Analisi Requisiti	3
1.1 Il progetto	3
1.1.1 Introduzione	3
1.1.2 Ciclo di vita di un ordine	3
1.2 Use Case	4
1.2.1 Diagrams	4
1.2.2 Templates	6
1.3 Activity Diagrams	8
1.3.1 Cliente effettua un ordine	8
1.3.2 Cliente richiede l'annullamento di un ordine	9
1.3.3 Cliente richiede il reso di un ordine	9
1.4 Mockups	10
1.4.1 Creazione di un ordine	10
1.4.2 Annullamento di un ordine	14
1.4.3 Reso di un ordine	15
2 Implementazione	16
2.1 Realizzazione	16
2.2 Struttura	16
2.3 Domain Model	17
2.3.1 Cliente	18
2.3.2 Fornitore	18
2.3.3 Articolo - Capo, Variante e Taglia	18
2.3.4 ArticoloMagazzino	18
2.3.5 ArticoloOrdineCliente	18
2.3.6 ArticoloOrdineFornitore	18
2.3.7 Stati	19
2.3.8 OrdineCliente	19
2.3.9 OrdineFornitore	19
2.3.10 ResoOrdine	19
2.4 Database e Object-Relational Mapping	19
2.4.1 Database	19
2.4.2 Object-Relational Mapping	21
2.5 Business Logic	23
2.5.1 ClienteController	23
2.5.2 RepartoCommercialeController	24
2.5.3 MagazzinoController	24
2.5.4 FornitoreController	24
2.6 Testing	25

1 Analisi Requisiti

1.1 Il progetto

1.1.1 Introduzione

Si vuole realizzare un applicativo per gestire un B2B di abbigliamento.

L'azienda è suddivisa in reparto commerciale e magazzino, e si affida a dei fornitori esterni per la produzioni degli articoli.

Il reparto commerciale si occupa della gestione degli ordini effettuati da parte dei clienti e di inoltrare ai relativi fornitori gli ordini degli articoli non presenti in magazzino.

I fornitori producono gli articoli che vengono ordinati dal reparto commerciale e li spediscono al magazzino.

Il magazzino contiene gli articoli ricevuti dai fornitori e prepara i colli degli ordini da spedire ai clienti una volta che ha tutti gli articoli necessari.

Il cliente ha la possibilità di effettuare ordini, annullare ordini (se i fornitori non hanno già spedito gli articoli) ed effettuare resi (a seguito di approvazione).

Ciascun ordine è composto da uno o più articoli, i quali vengono identificati dalla ternaria capo-variante-taglia, questo perché in fase di ordine il cliente seleziona per ciascun capo la taglia e la variante.

Quando viene effettuato un nuovo ordine da un cliente, prima di ordinare la produzione degli articoli ai fornitori, il reparto commerciale richiede al magazzino di verificare quelli già presenti in azienda (esiste infatti una "scorta" in magazzino generalmente formata da articoli appartenuti ad ordini resi o spediti erroneamente dai fornitori), in modo da utilizzarli per il nuovo ordine; successivamente il reparto commerciale ordina la produzione degli articoli mancanti ai fornitori.

L'annullamento di un ordine da parte di un cliente può essere effettuato solamente se i fornitori non hanno ancora iniziato a spedire gli articoli prodotti, egli deve quindi prima inviare una richiesta di annullamento al reparto commerciale, il quale potrà annullare l'ordine a seguito di una verifica.

Lo stesso flusso avviene nel caso di un reso, per il quale il cliente deve prima effettuare una richiesta (con tanto di motivazione), la quale deve essere approvata dal reparto commerciale; se questa viene approvata, il cliente spedirà indietro la merce al magazzino.

1.1.2 Ciclo di vita di un ordine

L'ordine di un cliente si attiene al seguente flusso:

1. Il reparto commerciale registra l'ordine del cliente
2. Il magazzino controlla se e quali articoli dell'ordine sono già presenti (se tutti presenti si passa allo step 6)
3. Il reparto commerciale effettua gli ordini verso i fornitori per gli articoli mancanti
4. I fornitori producono e spediscono gli articoli
5. Il magazzino registra gli articoli man mano che arrivano dai fornitori
6. Quando ha tutti gli articoli necessari, il magazzino prepara i colli per spedire l'ordine al cliente
7. Il magazzino spedisce l'ordine al cliente

Il flusso principale può deviare solo in caso di annullamento dell'ordine, il quale viene accettato solamente nei seguenti casi:

- Dallo step 2 siamo passati direttamente al 6, in quanto non sono stati effettuati ordini verso i fornitori
- Siamo nello step 3 e nessuno fornitore ha ancora iniziato a spedire i capi verso il magazzino

Il flusso principale può invece essere esteso in caso di richiesta di reso

1.2 Use Case

1.2.1 Diagrams

Ho realizzato i seguenti Use Case Diagrams tramite Creately, un apposito tool online.

Cliente

1. Effettua un ordine
2. Richiede l'annullamento di un ordine
3. Richiede il reso di un ordine
4. Effettua il reso di un ordine

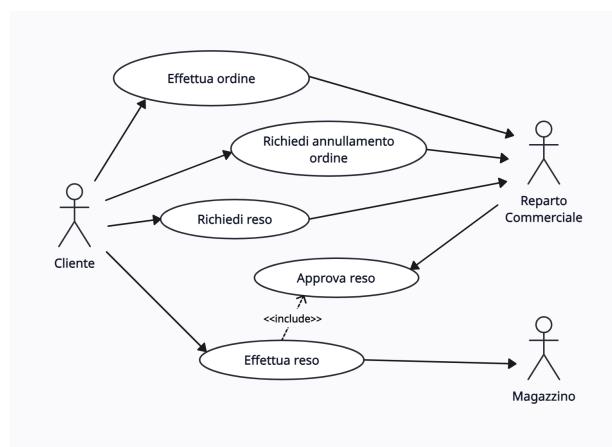


Figura 1: Diagramma use-case del Cliente

Reparto commerciale

1. Ordina gli articoli mancanti di un ordine del cliente
2. Verifica se i fornitori hanno già spedito degli articoli relativi ad un ordine cliente
3. Annulla un ordine
4. Approva il reso di un ordine

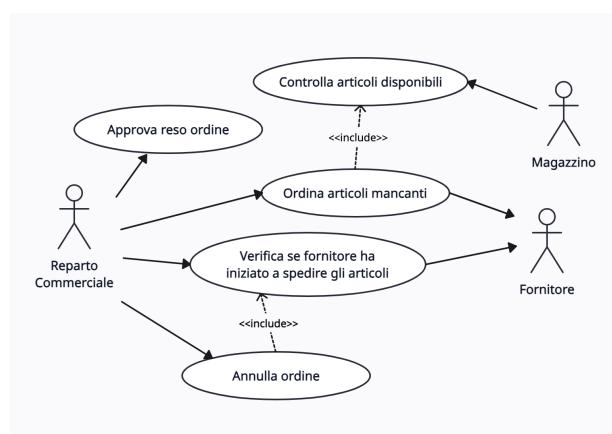


Figura 2: Diagramma use-case del Reparto commerciale

Fornitori

1. Producono un articolo
2. Spediscono un articolo

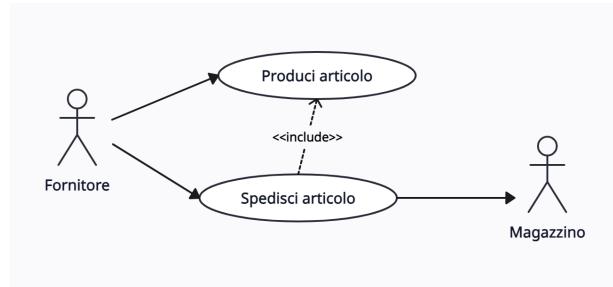


Figura 3: Diagramma use-case dei Fornitori

Magazzino

1. Aggiunge un articolo alla scorta del magazzino
2. Controlla gli articoli disponibili in magazzino
3. Registra un articolo spedito da un fornitore
4. Verifica di avere tutti gli articoli necessari alla preparazione di un ordine
5. Prepara un ordine
6. Spedisce un ordine

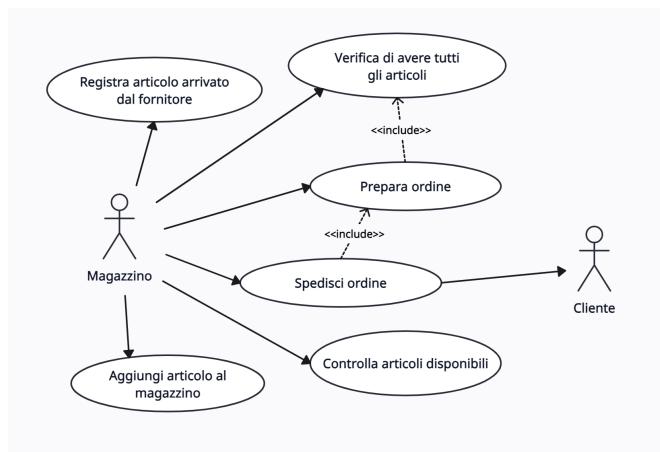


Figura 4: Diagramma use-case del Magazzino

1.2.2 Templates

Reparto commerciale - Use case N°1	Ordina articoli mancanti
Description	Il reparto commerciale ordina gli articoli mancanti dell'ordine di un cliente
Level	User Goal
Actors	Reparto comemrciale, Magazzino
Andamento Regolare	<ol style="list-style-type: none"> 1. Arriva un nuovo ordine al reparto commerciale 2. Il magazzino controlla se e quali articoli dell'ordine sono già presenti 3. Il magazzino non trova articoli disponibili 4. Il reparto commerciale ordina la produzione di ciascun articolo dell'ordine al relativo fornitore
Andamento Alternativo	<ol style="list-style-type: none"> 3a. Il magazzino trova alcuni articoli disponibili 4a. Il reparto commerciale ordina la produzione degli articoli mancanti ai relativi fornitori
Andamento Alternativo	<ol style="list-style-type: none"> 3b. Il magazzino trova tutti gli articoli 4b. Il reparto commerciale non deve effettuare alcun ordine di produzione ai fornitori ed il magazzino può iniziare a preparare i colli da spedire

Vedi mockups a Figura 10 e Figura 11

Cliente - Use case №2	Richiedi annullamento ordine
Description	Il cliente richiede l'annullamento di un ordine effettuato
Level	User Goal
Actors	Cliente, Reparto commerciale
Andamento Regolare	<ol style="list-style-type: none"> 1. Il cliente richiede l'annullamento di un ordine al reparto commerciale 2. L'ordine contiene articoli per i quali sono stati effettuati ordini di produzione verso i fornitori 3. Il reparto commerciale verifica se almeno un fornitore ha iniziato a spedire gli articoli 4. Nessun fornitore ha ancora inviato articoli 5. Il reparto commerciale annulla l'ordine del cliente
Andamento Alternativo	<ol style="list-style-type: none"> 4a. Uno o più fornitori hanno iniziato a spedire gli articoli 5a. Il reparto commerciale nega la richiesta di annullamento
Andamento Alternativo	<ol style="list-style-type: none"> 2b. L'ordine non contiene articoli per i quali sono stati effettuati ordini di produzione verso i fornitori 3b. Il reparto commerciale annulla l'ordine del cliente

Vedi mockup a Figura 17

Cliente - Use case N°4	Effettua il reso di un ordine
Description	Il cliente vuole effettuare un reso di un ordine
Level	User Goal
Actors	Cliente, Reparto commerciale, Magazzino
Andamento Regolare	<ol style="list-style-type: none"> 1. Il cliente manda la richiesta di reso di un ordine al reparto commerciale 2. Il reparto commerciale controlla e approva la richiesta 3. Il cliente effettua il reso rispedendo la merce al magazzino
Andamento Alternativo	<ol style="list-style-type: none"> 2a. Il reparto commerciale controlla e rifiuta la richiesta 3a. Il cliente non può effettuare il reso

Vedi mockup a Figura 19

1.3 Activity Diagrams

1.3.1 Cliente effettua un ordine

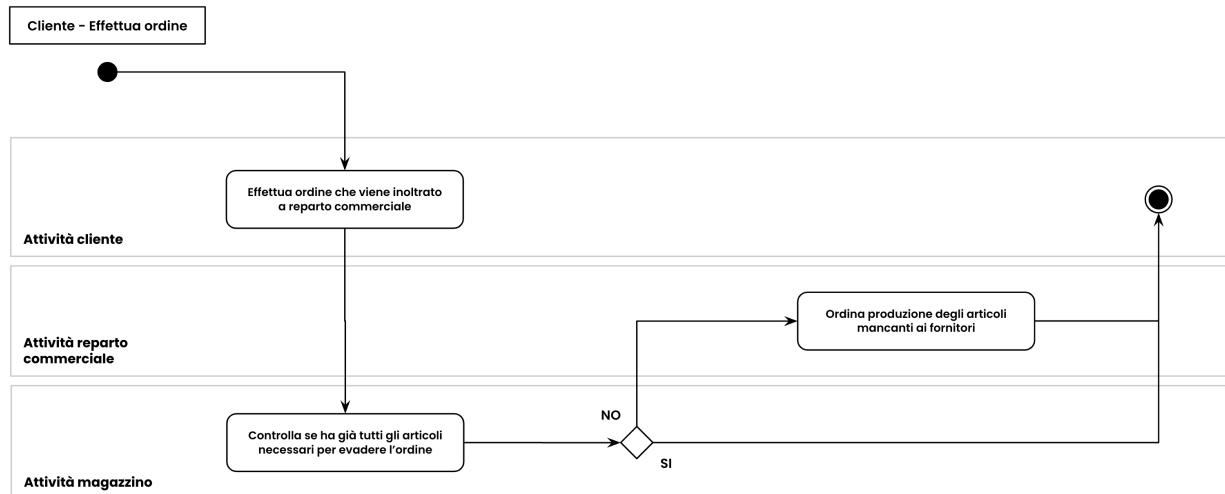


Figura 5: Diagramma attività del cliente che effettua un ordine

1.3.2 Cliente richiede l'annullamento di un ordine

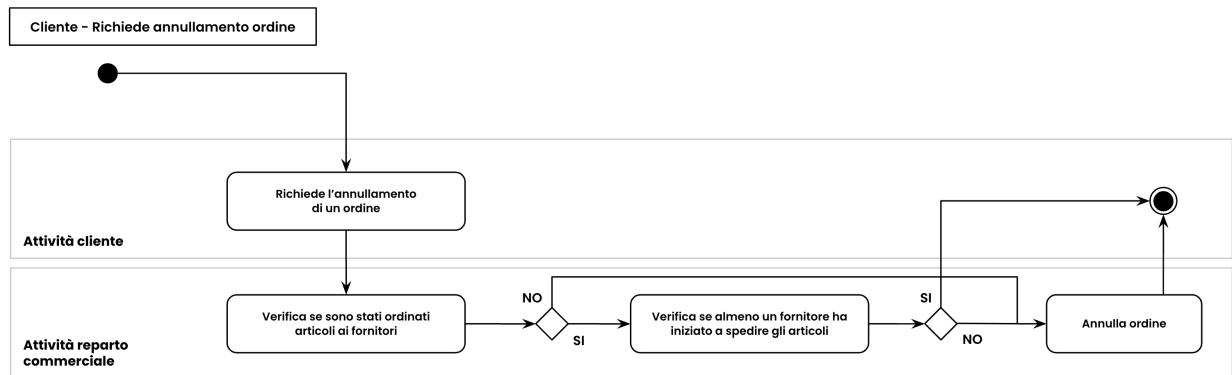


Figura 6: Diagramma attivita' del cliente che richiede l'annullamento di un ordine

1.3.3 Cliente richiede il reso di un ordine

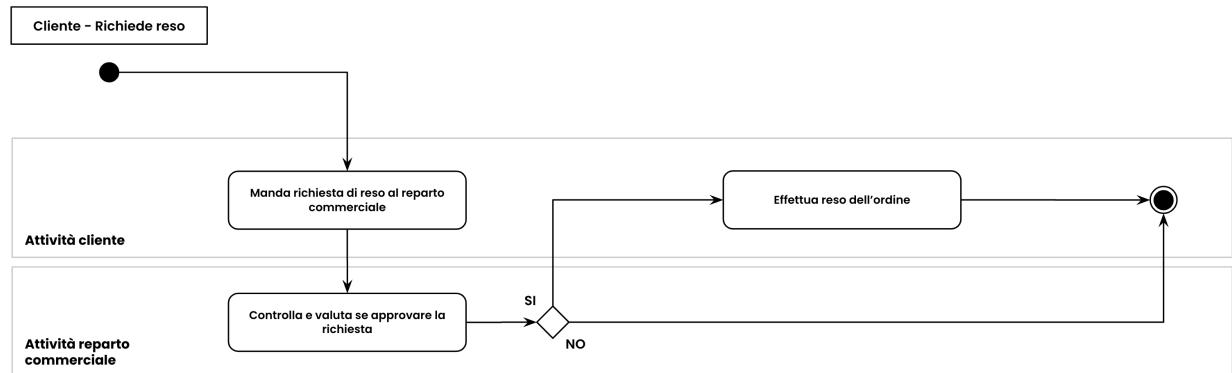


Figura 7: Diagramma attivita' del cliente che richiede di effettuare un reso

1.4 Mockups

Ho realizzato tramite Balsamiq Wireframes dei mockups della parte Front-End dell'applicativo riguardanti i flussi di creazione, annullamento e reso di un ordine (descritti precedentemente nei templates dei casi d'uso e nei diagrammi di attività)

1.4.1 Creazione di un ordine

Il cliente si interesserà con l'applicativo sottoforma di shop, e potrà quindi formare il suo carrello aggiungendo gli articoli dalle rispettive pagine di dettaglio. Ipotizziamo quindi che egli arrivi alla schermata di riepilogo dell'ordine dopo aver composto il carrello:

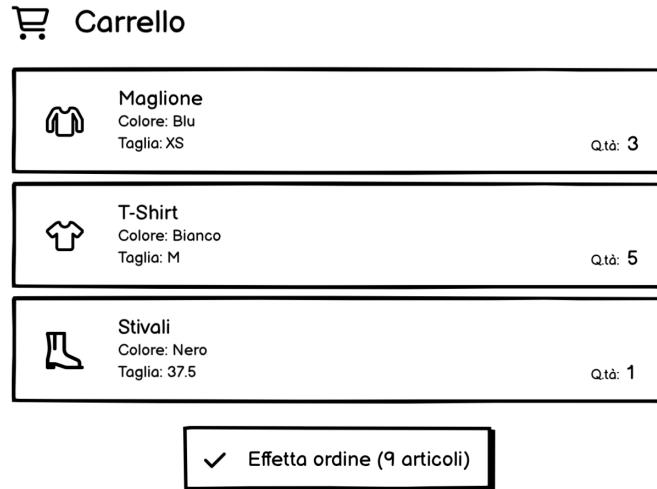


Figura 8: Vista del cliente per effettuare l'ordine

Ordini Cliente			Filtra per stato: Da controllare ▾
Ordine #24545	Stato: Da controllare	Cliente: Luigi Bianchi	^
Ordine #24546	Stato: Da controllare	Cliente: Mario Rossi	▼
Maglione Colore: Blu Taglia: XS	Q.tà ordinata dal cliente: 3 Q.tà in magazzino: 0	Q.tà ordinata al fornitore: 0 Q.tà arrivata dal fornitore: 0	
T-Shirt Colore: Bianco Taglia: M	Q.tà ordinata dal cliente: 5 Q.tà in magazzino: 0	Q.tà ordinata al fornitore: 0 Q.tà arrivata dal fornitore: 0	
Stivali Colore: Nero Taglia: 37.5	Q.tà ordinata dal cliente: 1 Q.tà in magazzino: 0	Q.tà ordinata al fornitore: 0 Q.tà arrivata dal fornitore: 0	
Ordine #24547	Stato: Da controllare	Cliente: Federica Verdi	^

Figura 9: Vista del reparto commerciale con gli ordini appena arrivati

Ordini da controllare

Ordine #24545	Stato: Da controllare	Cliente: Luigi Bianchi	^
Ordine #24546	Stato: Da controllare	Cliente: Mario Rossi	▼
Conferma quantità magazzino			

Ordine #24547 Stato: **Da controllare** Cliente: Federica Verdi ^

Figura 10: Vista del magazzino con gli ordini da controllare

Ordini Cliente

Filtra per stato:	<input type="button" value="Controllato ▼"/>
Ordine #24545	Stato: Controllato Cliente: Luigi Bianchi ^
Ordine #24546	Stato: Controllato Cliente: Mario Rossi ▼
Ordina articoli ai fornitori	
Ordine #24547	Stato: Controllato Cliente: Federica Verdi ^

Figura 11: Vista del reparto commerciale con gli ordini controllati e di cui vanno effettuati gli ordini ai fornitori

Ordini ricevuti				Filtra per stato:	Tutti ▾
Ordine Fornitore #13974595 Stato: Da produrre					
 T-Shirt Colore: Blu Taglia: XS	Q.tà ordinata: <input type="text" value="1"/>	Q.tà spedita: <input type="text" value="0"/>			
<hr/>					
				Avvia produzione	
Ordine Fornitore #13974596 Stato: Produzione avviata					
 T-Shirt Colore: Blu Taglia: XS	Q.tà ordinata: <input type="text" value="5"/>	Q.tà spedita: <input style="background-color: #f0f0f0; border: 1px solid #ccc; width: 20px; height: 20px; vertical-align: middle;" type="text" value="1"/> 			
	Q.tà prodotta: <input style="background-color: #f0f0f0; border: 1px solid #ccc; width: 20px; height: 20px; vertical-align: middle;" type="text" value="3"/> 				
<hr/>					
				Salva modifiche	
Ordine Fornitore #13974597 Stato: Completamente evaso					
 T-Shirt Colore: Blu Taglia: XS	Q.tà ordinata: <input type="text" value="3"/>	Q.tà spedita: <input type="text" value="3"/>			
	Q.tà prodotta: <input type="text" value="3"/>				
<hr/>					

Figura 12: Vista del fornitore con gli ordini fornitore da produrre e spedire

Ordinati ai fornitori

Ordine #24545	Stato: Ordinato ai fornitori	Cliente: Luigi Bianchi	^
Ordine #24546 Stato: Ordinato ai fornitori Cliente: Mario Rossi			
 Maglione Colore: Blu Taglia: XS	Q.tà ordinata dal cliente: <input type="text" value="3"/>	Q.tà ordinata al fornitore: <input type="text" value="2"/>	
<hr/>			
 T-Shirt Colore: Bianco Taglia: M	Q.tà ordinata dal cliente: <input type="text" value="5"/>	Q.tà ordinata al fornitore: <input type="text" value="3"/>	
	Q.tà in magazzino: <input type="text" value="2"/>	Q.tà arrivata dal fornitore: <input style="background-color: #f0f0f0; border: 1px solid #ccc; width: 20px; height: 20px; vertical-align: middle;" type="text" value="2"/> 	
<hr/>			
 Stivali Colore: Nero Taglia: 37.5	Q.tà ordinata dal cliente: <input type="text" value="1"/>	Q.tà ordinata al fornitore: <input type="text" value="1"/>	
	Q.tà in magazzino: <input type="text" value="0"/>	Q.tà arrivata dal fornitore: <input style="background-color: #f0f0f0; border: 1px solid #ccc; width: 20px; height: 20px; vertical-align: middle;" type="text" value="0"/> 	
<hr/>			
		Aggiorna quantità arrivate	
Ordine #24547 Stato: Ordinato ai fornitori Cliente: Federica Verdi			
<hr/>			

Figura 13: Vista del magazzino con gli ordini di cui sono stati effettuati gli ordini ai fornitori

Ordini da preparare

Ordine #24545	Stato: Da preparare	Cliente: Luigi Bianchi	^
Ordine #24546	Stato: Da preparare	Cliente: Mario Rossi	▼
Termina preparazione			
 Maglione Colore: Blu Taglia: XS	Q.tà ordinata dal cliente: 3 Q.tà in magazzino: 1	Q.tà ordinata al fornitore: 2 Q.tà arrivata dal fornitore: 2	
 T-Shirt Colore: Bianco Taglia: M	Q.tà ordinata dal cliente: 5 Q.tà in magazzino: 2	Q.tà ordinata al fornitore: 3 Q.tà arrivata dal fornitore: 3	
 Stivali Colore: Nero Taglia: 37.5	Q.tà ordinata dal cliente: 1 Q.tà in magazzino: 0	Q.tà ordinata al fornitore: 1 Q.tà arrivata dal fornitore: 1	
Termina preparazione			
Ordine #24547	Stato: Da preparare	Cliente: Federica Verdi	^

Figura 14: Vista del magazzino con gli ordini da preparare

Ordini da spedire

Ordine #24545	Stato: Da spedire	Cliente: Luigi Bianchi	^
Ordine #24546	Stato: Da spedire	Cliente: Mario Rossi	▼
Conferma spedizione			
 Maglione Colore: Blu Taglia: XS	Q.tà ordinata dal cliente: 3 Q.tà in magazzino: 1	Q.tà ordinata al fornitore: 2 Q.tà arrivata dal fornitore: 2	
 T-Shirt Colore: Bianco Taglia: M	Q.tà ordinata dal cliente: 5 Q.tà in magazzino: 2	Q.tà ordinata al fornitore: 3 Q.tà arrivata dal fornitore: 3	
 Stivali Colore: Nero Taglia: 37.5	Q.tà ordinata dal cliente: 1 Q.tà in magazzino: 0	Q.tà ordinata al fornitore: 1 Q.tà arrivata dal fornitore: 1	
Conferma spedizione			
Ordine #24547	Stato: Da spedire	Cliente: Federica Verdi	^

Figura 15: Vista del magazzino con gli ordini da spedire

1.4.2 Annullamento di un ordine

Ordine #34452

	Maglione Colore: Blu Taglia: XS	Q.tà: 3
	T-Shirt Colore: Bianco Taglia: M	Q.tà: 5
Richiedi annullamento		

Figura 16: Vista del cliente per richiedere l'annullamento dell'ordine

Una volta richiesto l'annullamento, l'interfaccia del reparto commerciale mostrerà se è possibile annullare l'ordine (esito della verifica effettuata sugli articoli spediti dai fornitori) oppure se la richiesta deve essere rifiutata, riportando quindi l'ordine nel normale flusso di elaborazione.

Ordini Cliente			Filtra per stato:	Richiesto annullamento
Ordine #34452	Stato: Richiesto annullamento	Cliente: Mario Rossi		▼
	Maglione Colore: Blu Taglia: XS	Q.tà ordinata dal cliente: 3 Q.tà ordinata al fornitore: 2 Q.tà in magazzino: 1 Q.tà arrivata dal fornitore: 0		
	T-Shirt Colore: Bianco Taglia: M	Q.tà ordinata dal cliente: 5 Q.tà ordinata al fornitore: 3 Q.tà in magazzino: 2 Q.tà arrivata dal fornitore: 3		
● Impossibile annullare l'ordine				Rifiuta annullamento
Ordine #34478	Stato: Richiesto annullamento	Cliente: Luigi Bianchi		▼
	Stivali Colore: Nero Taglia: 37.5	Q.tà ordinata dal cliente: 1 Q.tà ordinata al fornitore: 1 Q.tà in magazzino: 0 Q.tà arrivata dal fornitore: 0		
● Possibile annullare ordine				Annulla ordine

Figura 17: Vista del reparto commerciale con gli ordini di cui è stato richiesto l'annullamento

1.4.3 Reso di un ordine

Nel progetto ho semplificato la richiesta di reso utilizzando solamente il campo motivazione di tipo testuale come informazione fornita dal cliente per l'approvazione del reso; in realtà andrebbero forniti altri dati per poter effettuare un reso, come ad esempio foto o video di merce danneggiata.

Ordine #24546

	Maglione Colore: Blu Taglia: XS	Q.tà: 3
	T-Shirt Colore: Bianco Taglia: M	Q.tà: 5
	Stivali Colore: Nero Taglia: 37.5	Q.tà: 1
Motivazione reso <input type="text" value="Articoli difetti"/>		
Apri richiesta di reso dell'ordine		

Figura 18: Vista del cliente per richiedere il reso dell'ordine

Ordini Cliente				Filtra per stato:	Richiesto reso	▼
Ordine #24546	Stato: Richiesto reso	Cliente: Mario Rossi	▼			
	Maglione Colore: Blu Taglia: XS	Q.tà ordinata dal cliente: 3	Q.tà ordinata al fornitore: 2			
	Q.tà in magazzino: 1	Q.tà arrivata dal fornitore: 2				
	T-Shirt Colore: Bianco Taglia: M	Q.tà ordinata dal cliente: 5	Q.tà ordinata al fornitore: 3			
	Q.tà in magazzino: 2	Q.tà arrivata dal fornitore: 3				
	Stivali Colore: Nero Taglia: 37.5	Q.tà ordinata dal cliente: 1	Q.tà ordinata al fornitore: 1			
	Q.tà in magazzino: 0	Q.tà arrivata dal fornitore: 1				
Motivazione: Articoli difettosi				Approva reso		

Figura 19: Vista del reparto commerciale con gli ordini di cui è stato richiesto il reso

2 Implementazione

2.1 Realizzazione

Sono andato a sviluppare la componente Back-End dell'applicativo in Java utilizzando Eclipse come IDE, ed il progetto si trova su un repository GitHub:
<https://github.com/LeonardoBorsi/fashion-b2b>.

Tutti gli snippet del codice e dei packages sono screenshots catturati su Eclipse, mentre i diagrammi di classe e di pacchetti sono stati creati tramite StarUML.

2.2 Struttura

Ho suddiviso il progetto in due macro pacchetti:

- Il package main che contiene tutte le classi utili alla realizzazione del gestionale, suddivise nei 3 subpackages:
 1. logic (Business Logic)
 2. domain (Domain Model)
 3. orm (Object-Relational Mapping)
- Il package test che contiene tutte le classi utili al testing delle componenti definite nel main

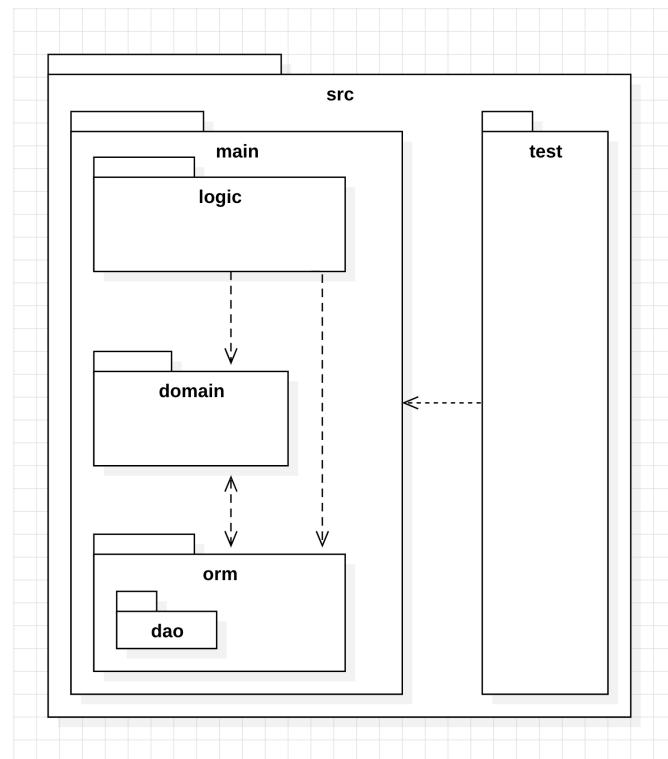


Figura 20: Struttura pacchetti

2.3 Domain Model

Il Domain Model è definito nel package domain, il quale contiene le entità del dominio da dover rappresentare. Seguono il diagramma delle classi e la struttura del pacchetto:

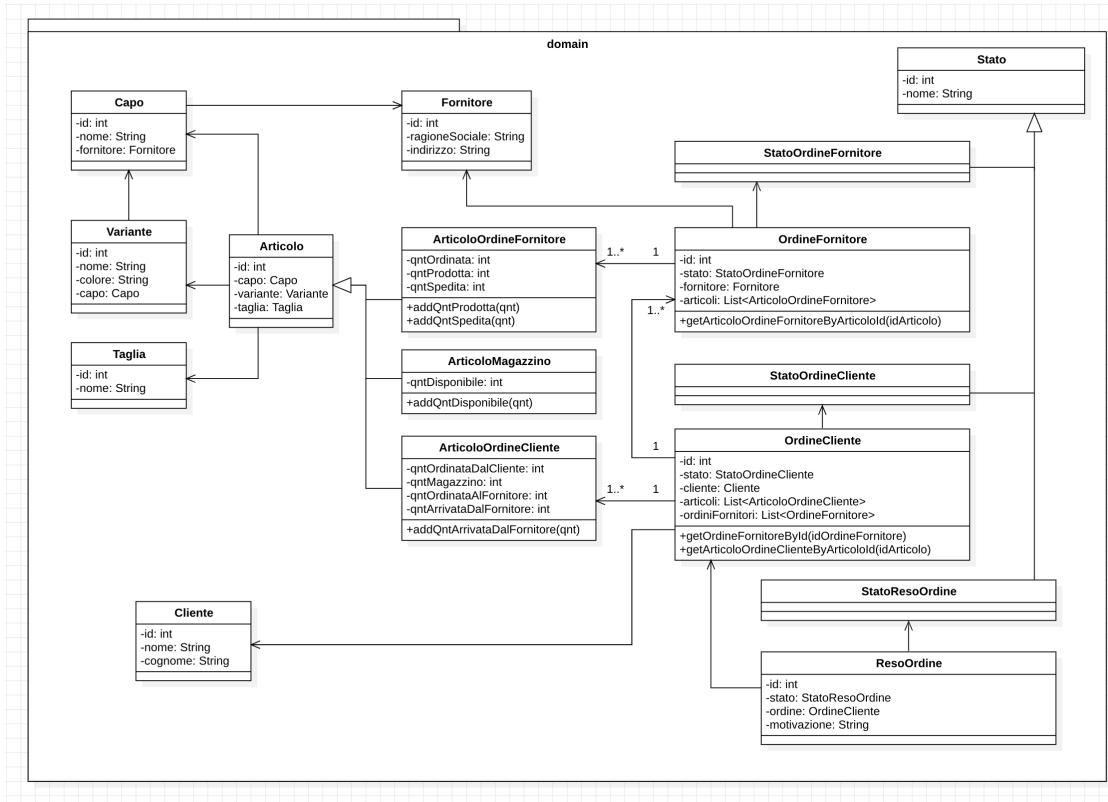


Figura 21: Diagramma delle classi del Domain Model

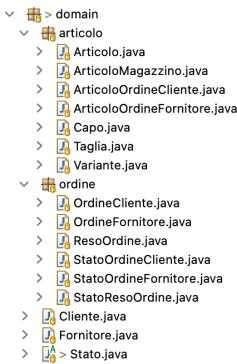


Figura 22: Package domain

2.3.1 Cliente

Rappresenta il cliente che effettua un ordine verso il reparto commerciale e presenta solamente gli attributi id, nome e cognome.

2.3.2 Fornitore

Identifica un qualsiasi fornitore del B2B, il quale produce i capi ordinati dal reparto commerciale per poi spedirli al magazzino, ed è descritto dagli attributi id, ragioneSociale ed indirizzo.

2.3.3 Articolo - Capo, Variante e Taglia

Capo, Variante e Taglia sono le tre classi che insieme vanno a determinare l'articolo che il cliente può ordinare:

- La classe Capo identifica uno dei capi prodotti da un certo fornitore, il quale può avere una o più varianti, ed è descritta dall'id, il nome del capo ed il fornitore che lo produce
- La classe Variante identifica la variante di un certo capo ed è descritta dall'id, il nome della variante, il colore di questa ed il capo a cui fa riferimento
- La classe Taglia identifica semplicemente la taglia di un determinato prodotto ed è descritta dall'id ed il nome della taglia

La classe Articolo non è altro che la ternaria di queste tre classi, ed identifica il vero e proprio articolo che il cliente può inserire nel carrello ed acquistare; viene quindi descritta dall'id ed i riferimenti alle tre precedenti classi.

2.3.4 ArticoloMagazzino

Rappresenta gli articoli depositati nel magazzino e va quindi ad estendere la classe Articolo con l'attributo qntDisponibile che identifica il numero di unità dell'articolo in questione disponibili ad essere usate per comporre nuovi ordini.

2.3.5 ArticoloOrdineCliente

Rappresenta gli articoli che vengono ordinati dal cliente ed estende la classe Articolo con i seguenti attributi:

- qntOrdinataDalCliente, il numero di unità dell'articolo ordinate dal cliente
- qntMagazzino, il numero di unità prese dal magazzino per soddisfare la quantità ordinata
- qntOrdinataAlFornitore, ovvero il numero di unità da dover ordinare al fornitore, che non è altro che la differenza tra la quantità ordinata dal cliente e quella già presente in magazzino
- qntArrivataDalFornitore, il numero di unità arrivate al magazzino a seguito di un ordine al fornitore dell'articolo in questione

2.3.6 ArticoloOrdineFornitore

Rappresenta gli articoli che vengono ordinati al fornitore per soddisfare l'ordine di un cliente ed anche questa estende la classe Articolo tramite:

- qntOrdinata, il numero di unità dell'articolo ordinate al fornitore
- qntProdotta, il numero di unità prodotte dal fornitore
- qntSpedita, il numero di unità spedite al magazzino dal fornitore

2.3.7 Stati

Stato è la classe astratta che ci definisce come vengono descritti gli stati nell'applicazione, ovvero tramite id e nome, e viene estesa dalle seguenti classi:

- StatoOrdineCliente, che rappresenta lo stato in cui si trova l'ordine di un cliente
- StatoOrdineFornitore, che rappresenta lo stato in cui si trova l'ordine verso un fornitore
- StatoResoOrdine, che rappresenta lo stato in cui si trova il reso dell'ordine di un cliente

2.3.8 OrdineCliente

Rappresenta l'ordine effettuato dal cliente ed è identificato dall'id, lo stato in cui si trova, il cliente a cui appartiene, gli articoli ordinati (istanze di ArticoloOrdineCliente) e gli eventuali ordini effettuati verso i fornitori per soddisfare le quantità ordinate.

2.3.9 OrdineFornitore

Rappresenta l'ordine effettuato verso il fornitore ed è identificato dall'id, lo stato in cui si trova, il fornitore verso cui è stato fatto e gli articoli che sono stati ordinati (istanze di ArticoloOrdineFornitore).

2.3.10 ResoOrdine

Rappresenta il reso avviato per un certo ordine di un cliente ed è identificato dall'id, lo stato in cui si trova, l'ordine a cui fa riferimento e la motivazione per cui è stato richiesto.

2.4 Database e Object-Relational Mapping

2.4.1 Database

Per gestire la persistenza dei dati ho optato per un database PostgreSQL, utilizzando Render come piattaforma di hosting, in modo da poter operare sul medesimo database da terminali diversi. Seguono lo schema logico e lo schema ER del database:

```
clienti(PK(id), nome, cognome)

fornitori(PK(id), ragione_sociale, indirizzo)

taglie(PK(id), nome)

capi(PK(id), nome, id_fornitore)
    , FK(id_fornitore) REF fornitori

varianti(PK(id), nome, colore, id_capo)
    , FK(id_capo) REF capi

articoli(PK(id), id_capo, id_variante, id_taglia)
    , FK(id_capo) REF capi
    , FK(id_variante) REF varianti
    , FK(id_taglia) REF taglie

stati_ordine_cliente(PK(id), nome)

ordini_clienti(PK(id), id_stato_ordine_cliente, id_cliente)
    , FK(id_stato_ordine_cliente) REF stati_ordine_cliente
    , FK(id_cliente) REF clienti
```

```

articolisti_ordine_cliente(PK(id), id_ordine_cliente, id_articolo,
    qnt_ordinata_dal_cliente, qnt_magazzino, qnt_ordinata_al_fornitore,
    qnt_arrivata_dal_fornitore)
, FK(id_ordine_cliente) REF ordini_cliente
, FK(id_articolo) REF articoli

stati_ordine_fornitore(PK(id), nome)

ordini_fornitori(PK(id), id_stato_ordine_fornitore, id_fornitore,
    id_ordine_cliente)
, FK(id_stato_ordine_fornitore) REF stati_ordine_fornitore
, FK(id_fornitore) REF fornitori
, FK(id_ordine_cliente) REF ordini_cliente

articolisti_ordine_fornitore(PK(id), id_ordine_fornitore, id_articolo, qnt_ordinata
    , qnt_prodotta, qnt_spedita)
, FK(id_ordine_fornitore) REF ordini_fornitori
, FK(id_articolo) REF articoli

stati_reso_ordine(PK(id), nome)

resi_ordini(PK(id), id_stato_reso_ordine, id_ordine_cliente, motivazione)
, FK(id_stato_reso_ordine) REF stati_reso_ordine
, FK(id_ordine_cliente) REF ordini_cliente

articolisti_magazzino(PK(id), id_articolo, qnt_disponibile)
, FK(id_articolo) REF articoli

```

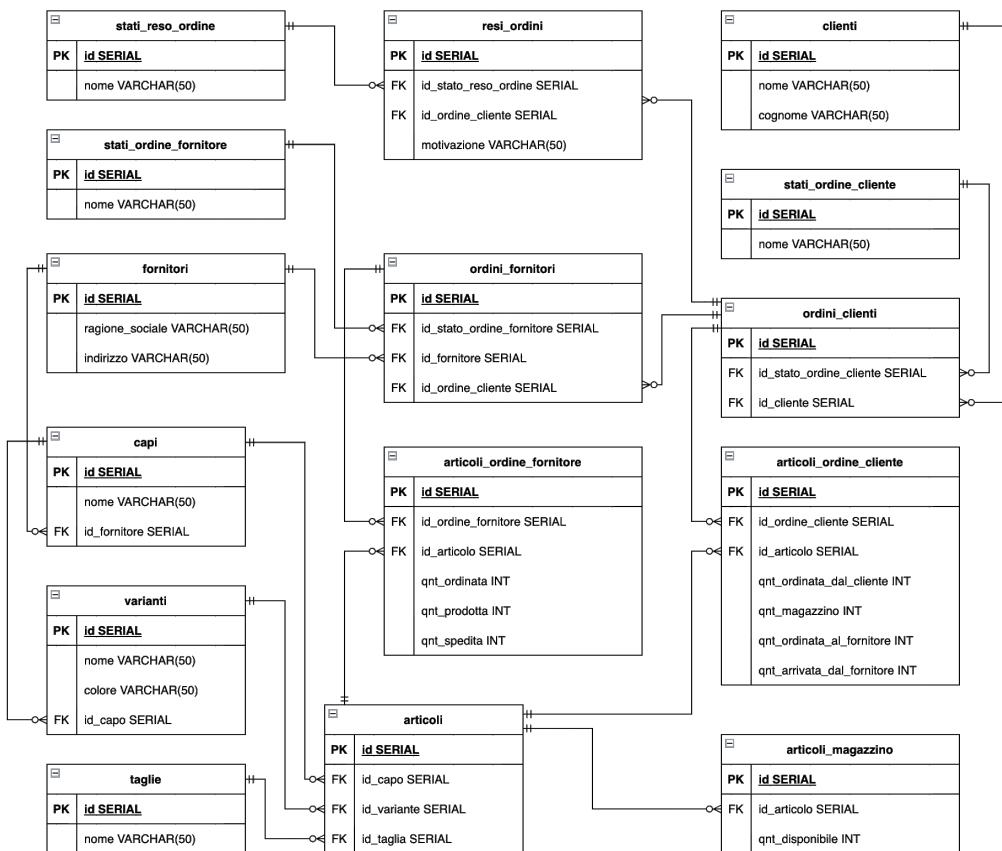


Figura 23: Schema ER del database

2.4.2 Object-Relational Mapping

L'Object-Relational Mapping è definito nel package `orm`, il quale contiene:

- la classe `ConnectionManager` che gestisce la connessione al database
- la classe `Database` che gestisce le operazioni effettuate sul database
- il subpackage `dao`, il quale contiene le corrispettive classi DAO di tutte le classi presenti nel Domain Model

Seguono il diagramma delle classi e la struttura del pacchetto:

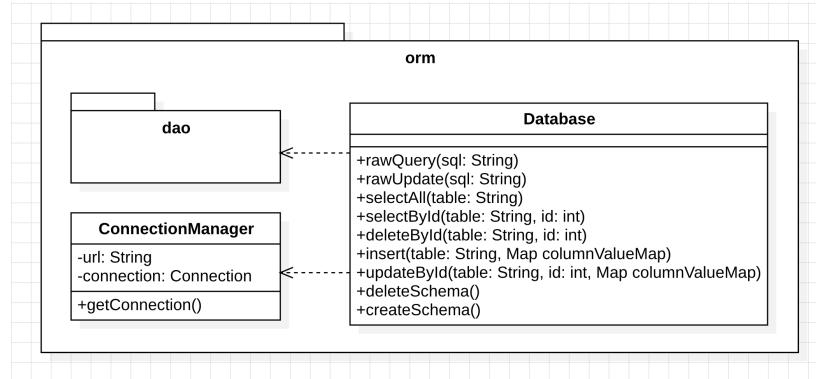


Figura 24: Diagramma delle classi dell'Object-Relational Mapping



Figura 25: Package orm

Tutte le classi DAO presentano i seguenti metodi:

- Insert - inserisce l'istanza dell'entità nella relativa tabella del db
- Update - aggiorna la riga del db corrispondente all'istanza dell'entità
- Delete - rimuove l'istanza dell'entità dalla relativa tabella del db
- Get - restituisce l'istanza dell'entità ottenuta dalla corrispettiva riga del db
- GetAll - restituisce un array con tutte le istanze dell'entità presenti nella relativa tabella del db

Alcune classi DAO presentano metodi aggiuntivi utili ai Controller della Business Logic; seguono alcuni esempi:

```

public static List<ArticoloOrdineCliente> getArticoliOrdineClinteByOrdineClienteId(
    int idOrdineCliente
) throws ClassNotFoundException, SQLException {
    List<ArticoloOrdineCliente> articoliOrdineCliente = new ArrayList<>();
    StringBuilder sqlBuilder = new StringBuilder("SELECT * FROM ")
        .append(Database.Table.articoli_ordine_cliente)
        .append(" WHERE id_ordine_cliente = ")
        .append(idOrdineCliente);
    ResultSet rs = Database.rawQuery(sqlBuilder.toString());
    while (rs.next()) {
        Articolo articolo = ArticoloDAO.getArticolo(rs.getInt("id_articolo"));
        articoliOrdineCliente.add(new ArticoloOrdineCliente(
            articolo,
            rs.getInt("qnt_ordinata_dal_cliente"),
            rs.getInt("qnt_magazzino"),
            rs.getInt("qnt_ordinata_al_fornitore"),
            rs.getInt("qnt_arrivata_dal_fornitore")
        ));
    }
    return articoliOrdineCliente;
}

public static ArticoloOrdineCliente getArticoloOrdineClinteByOrdineClienteIdAndArticoloID(
    int idOrdineCliente,
    int idArticolo
) throws ClassNotFoundException, SQLException {
    StringBuilder sqlBuilder = new StringBuilder("SELECT * FROM ")
        .append(Database.Table.articoli_ordine_cliente)
        .append(" WHERE id_ordine_cliente = ")
        .append(idOrdineCliente)
        .append(" AND id_articolo = ")
        .append(idArticolo);
    ResultSet rs = Database.rawQuery(sqlBuilder.toString());
    if (rs.next()) {
        Articolo articolo = ArticoloDAO.getArticolo(rs.getInt("id_articolo"));
        return new ArticoloOrdineCliente(
            articolo,
            rs.getInt("qnt_ordinata_dal_cliente"),
            rs.getInt("qnt_magazzino"),
            rs.getInt("qnt_ordinata_al_fornitore"),
            rs.getInt("qnt_arrivata_dal_fornitore")
        );
    }
    return null;
}

```

Figura 26: Metodi aggiuntivi ArticoloOrdineClienteDAO

```

public static List<ArticoloOrdineFornitore> getArticoliOrdineFornitoreByOrdineFornitoreId(
    int idOrdineFornitore
) throws ClassNotFoundException, SQLException {
    List<ArticoloOrdineFornitore> articoliOrdineFornitore = new ArrayList<>();
    StringBuilder sqlBuilder = new StringBuilder("SELECT * FROM ")
        .append(Database.Table.articoli_ordine_fornitore)
        .append(" WHERE id_ordine_fornitore = ")
        .append(idOrdineFornitore);
    ResultSet rs = Database.rawQuery(sqlBuilder.toString());
    while (rs.next()) {
        Articolo articolo = ArticoloDAO.getArticolo(rs.getInt("id_articolo"));
        articoliOrdineFornitore.add(new ArticoloOrdineFornitore(
            articolo,
            rs.getInt("qnt_ordinata"),
            rs.getInt("qnt_prodotta"),
            rs.getInt("qnt_spedita")
        ));
    }
    return articoliOrdineFornitore;
}

```

Figura 27: Metodi aggiuntivi ArticoloOrdineFornitoreDAO

2.5 Business Logic

La Business Logic è definita nel package logic, il quale contiene le classi controller di ciascun attore, che implementano gli use case sfruttando le classi del Domain Model ed i DAO dell'Object-Relational Mapping. Seguono il diagramma delle classi e la struttura del pacchetto:

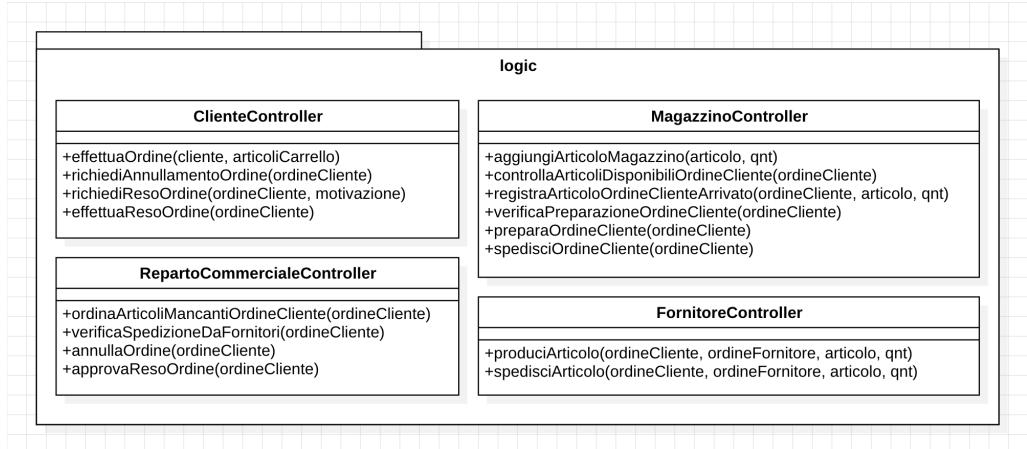


Figura 28: Diagramma delle classi della Business Logic



Figura 29: Package logic

2.5.1 ClienteController

La classe ClienteController implementa i casi d'uso individuati per il cliente:

- effettuaOrdine(cliente, articoliCarrello) permette di generare l'istanza di un OrdineCliente dati il cliente ed una Map di Articolo-Integer che rappresenta gli articoli presenti nel carrello.
- richiediAnnullamentoOrdine(ordineCliente) sposta l'OrdineCliente passato nello stato "Richiesto Annullamento".
- richiediResoOrdine(ordineCliente, motivazione) sposta l'OrdineCliente passato nello stato "Richiesto reso", controllando prima che sia stato effettivamente spedito al cliente (quindi nello stato "Spedito"), e genera l'istanza di ResoOrdine associata ad esso, includendo la motivazione del reso.
- effettuaResoOrdine(ordineCliente) sposta il ResoOrdine associato all'OrdineCliente passato nello stato "Reso effettuato", controllando prima che sia stato effettivamente approvato dal reparto commerciale (quindi nello stato "Reso Approvato").

2.5.2 RepartoCommercialeController

La classe RepartoCommerciale implementa i casi d'uso individuati per il reparto commerciale:

- ordinaArticoliMancantiOrdineCliente(OrdineCliente) dopo aver verificato che l'ordine sia nello stato "Controllato da magazzino", va a generare tutti gli OrdiniFornitori per gli articoli mancanti dell'OrdineCliente, e sposta quest'ultimo nello stato "Ordinato ai fornitori"; se non trova articoli mancanti da dover ordinare, sposta direttamente l'OrdineCliente nello stato "Da preparare".
- verificaSpedizioneDaFornitori(OrdineCliente) controlla se almeno uno dei fornitori degli OrdiniFornitori associati all'OrdineCliente passato ha iniziato a spedire gli articoli ordinati.
- annullaOrdine(OrdineCliente) sposta l'OrdineCliente passato nello stato "Annullato" a seguito della verificaSpedizioneDaFornitori e del controllo che l'ordine sia nello stato "Richiesto Annulloamento".
- approvaResoOrdine(OrdineCliente) sposta il ResoOrdine associato all'OrdineCliente passato nello stato "Reso approvato"

2.5.3 MagazzinoController

La classe Magazzino implementa i casi d'uso individuati per il magazzino:

- aggiungiArticoloMagazzino(Articolo, qnt) aggiunge l'Articolo passato come ArticoloMagazzino con la quantità indicata.
- controllaArticoliDisponibiliOrdineCliente(OrdineCliente) controlla quali articoli dell'OrdineCliente passato sono già presenti in magazzino e setta le quantità disponibili, inoltre sposta l'ordine nello stato "Controllato da magazzino".
- registraArticoloOrdineClienteArrivato(OrdineCliente, Articolo, qnt) aggiorna la quantità arrivata dal fornitore dell'Articolo dell'OrdineCliente passato
- verificaPreparazioneOrdineCliente(OrdineCliente) verifica di avere tutti gli articoli necessari alla preparazione dell'OrdineCliente passato, se sì va ad impostare l'ordine nello stato "Da preparare"
- preparaOrdineCliente(OrdineCliente) sposta l'OrdineCliente passato nello stato "Preparato", controllando prima che sia nello stato "Da preparare"
- spedisceOrdineCliente(OrdineCliente) sposta l'OrdineCliente passato nello stato "Spedito", controllando prima che sia nello stato "Preparato"

2.5.4 FornitoreController

La classe FornitoreController implementa i casi d'uso individuati per il fornitore:

- produciArticolo(OrdineCliente, OrdineFornitore, Articolo, qnt) aggiorna la quantità prodotta dell'Articolo appartenente all'OrdineFornitore associato all'OrdineCliente passato, ed imposta l'OrdineFornitore nello stato "Produzione avviata" se è il primo articolo dell'ordine che il fornitore produce.
- spedisceArticolo(OrdineCliente, OrdineFornitore, Articolo, qnt) aggiorna la quantità spedita dell'Articolo appartenente all'OrdineFornitore associato all'OrdineCliente passato, ed imposta l'OrdineFornitore nello stato "Completemente evaso" se il fornitore ha spedito tutti gli articoli dell'ordine.

2.6 Testing

Per il testing del codice ho utilizzato il framework JUnit, con cui sono andato ad implementare i test per i controllers della Business Logic. Il package test contiene quindi il subpackage controllers, con le corrispettive classi Test dei controller, e la classe TestUtils, la quale espone delle funzioni utili allo svolgimento dei test.

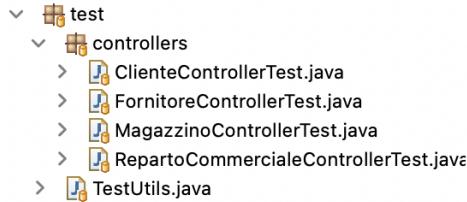


Figura 30: Package test

Le classi Test dei controllers vanno a testare ogni metodo che il controller della Business Logic espone. Questo avviene andando a "costruire" lo scenario in cui verrà utilizzato il metodo, per poi verificare che questo abbia avuto l'effetto desiderato.

Prendiamo come esempio il test del metodo effettuaOrdine del ClienteController (vedi Figura 32):

1. Prima di invocare il metodo ci genereremo un cliente ed un array di articoli presenti nel carrello (Map di Articolo-Integer)
2. Invochiamo il metodo effettuaOrdine il quale ci restituisce un'istanza di Ordine-Cliente
3. Verifichiamo gli attributi dell'istanza di OrdineCliente restituita (stato, cliente, articoli) e controlliamo che l'istanza sia presente su database ed allineata a quella ottenuta dal metodo

```

    @Test
    public void effettuaOrdine() {
        System.out.println("Cliente Controller Test > Effettua Ordine");
        try {
            Cliente cliente = TestUtils.getRandomCliente();
            OrdineCliente ordine = ClienteController.effettuaOrdine(cliente, TestUtils.generateArticoliCarrello(5, 10));
            OrdineCliente ordineDB = OrdineClienteDAO.getOrdineCliente(ordine.getId());
            assertEquals(ordine.getId(), ordineDB.getId());
            assertEquals(ordine.getStato().getId(), 1);
            assertEquals(ordine.getStato().getId(), ordineDB.getStato().getId());
            assertEquals(ordine.getCliente().getId(), cliente.getId());
            assertEquals(ordine.getCliente().getId(), ordineDB.getCliente().getId());
            assertEquals(ordine.getArticoli().size(), ordineDB.getArticoli().size());
            for(ArticoloOrdineCliente articolo : ordine.getArticoli()) {
                ArticoloOrdineCliente articoloDB = ordineDB.getArticoloOrdineClienteByArticoloId(articolo.getId());
                assertEquals(articolo.getQntOrdinataDalCliente(), articoloDB.getQntOrdinataDalCliente());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
  
```

Figura 31: Test del metodo effettuaOrdine del ClienteController

Tramite questi test riusciamo ad avere un coverage su tutti i casi d'uso individuati per l'applicativo; segue l'elenco di tutti i test.

```
ClientControllerTest [Runner: JUnit 5] (10,987 s)
  ✓ effettuaOrdine (2,993 s)
  ✓ richiediAnnullamentoOrdine (3,124 s)
  ✓ richiediResoOrdine (1,934 s)
  ✓ effettuaResoOrdine (2,935 s)

RepartoCommercialeControllerTest [Runner: JUnit 5] (11,695 s)
  ✓ ordinaArticoliMancanti (5,921 s)
  ✓ verificaSpedizioneArticoliOrdineClienteDaFornitori (1,434 s)
  ✓ annullaOrdine (1,677 s)
  ✓ approvaResoOrdine (2,660 s)

MagazzinoTest [Runner: JUnit 5] (11,689 s)
  ✓ preparaOrdineCliente (2,073 s)
  ✓ registraArticoloOrdineClienteArrivato (0,935 s)
  ✓ aggiungiArticoloMagazzino (1,319 s)
  ✓ verificaPreparazioneOrdineCliente (1,984 s)
  ✓ controllaArticoliDisponibiliOrdineCliente (3,495 s)
  ✓ spedisciOrdineCliente (1,881 s)

FornitoreControllerTest [Runner: JUnit 5] (3,003 s)
  ✓ spedisciArticolo (1,553 s)
  ✓ produciArticolo (1,449 s)
```

Figura 32: Test del metodo effettuaOrdine del ClienteController