UNIVERSITÀ DEGLI STUDI DI VERONA
DIPARTIMENTO DI INFORMATICA

Documentazione del progetto di

INGEGNERIA DEL SOFTWARE

Anno Accademico 2016/2017

Autori:

Donatelli Nicola – VR389842

Andreoni Matteo – VR389703

Sommario

1 Introduzione

Abbiamo scelto di implementare il progetto di Ingegneria del Software relativo al Music Store. Nella seguente relazione abbiamo inserito in ordine di lettura: le specifiche del progetto, i diagrammi richiesti, le assunzioni fatte, le scelte di implementazione fatte, la descrizione della fase di testing e i pattern utilizzati nel progetto.

2 Specifiche del progetto

Si vuole progettare un sistema informativo per gestire le informazioni relative alla gestione di un negozio virtuale di CD e DVD musicali (vende solo via web).

Il negozio mette in vendita CD di diversi generi: jazz, rock, classica, latin, folk, world-music, e così via.

Per ogni CD o DVD il sistema memorizza: un codice univoco, il titolo, i titoli di tutti i pezzi contenuti, eventuali fotografie della copertina, il prezzo, la data dalla quale è presente sul sito web del negozio, il musicista/band titolare, una descrizione, il genere del CD o DVD, i musicisti che vi suonano, con il dettaglio degli strumenti musicali usati. Per ogni musicista il sistema registra il nome d'arte, il genere principale, l'anno di nascita, se noto, gli strumenti che suona.

Sul sito web del negozio è illustrato il catalogo dei prodotti in vendita.

Cliccando sul nome del prodotto, appare una finestra con i dettagli del prodotto stesso.

I clienti possono acquistare on-line selezionando gli oggetti da mettere in un "carrello della spesa" virtuale.

Deve essere possibile visualizzare il contenuto del carrello, modificare il contenuto del carrello, togliendo alcuni articoli.

Al termine dell'acquisto va gestito il pagamento, che può avvenire con diverse modalità.

Il sistema supporta differenti ricerche: per genere, per titolare del CD o DVD, per musicista partecipante, per prezzo. Coerentemente, differenti modalità di visualizzazione, sono altresì supportate.

Ogni vendita viene registrata indicando il cliente che ha acquistato, i prodotti acquistati, il prezzo complessivo, la data di acquisto, l'ora, l'indirizzo IP del PC da cui è stato effettuato l'acquisto, la modalità di pagamento (bonifico, carta di credito, paypal) e la modalità di consegna (corriere, posta, …).

Per ogni cliente il sistema registra: il suo codice fiscale, il nome utente (univoco) con cui si è registrato, la sua password, il nome, il cognome, la città di residenza, il numero di telefono ed eventualmente il numero di cellulare.

Per i clienti autenticati, il sistema propone pagine specializzate che mostrano suggerimenti basati sul genere dei precedenti prodotti acquistati.

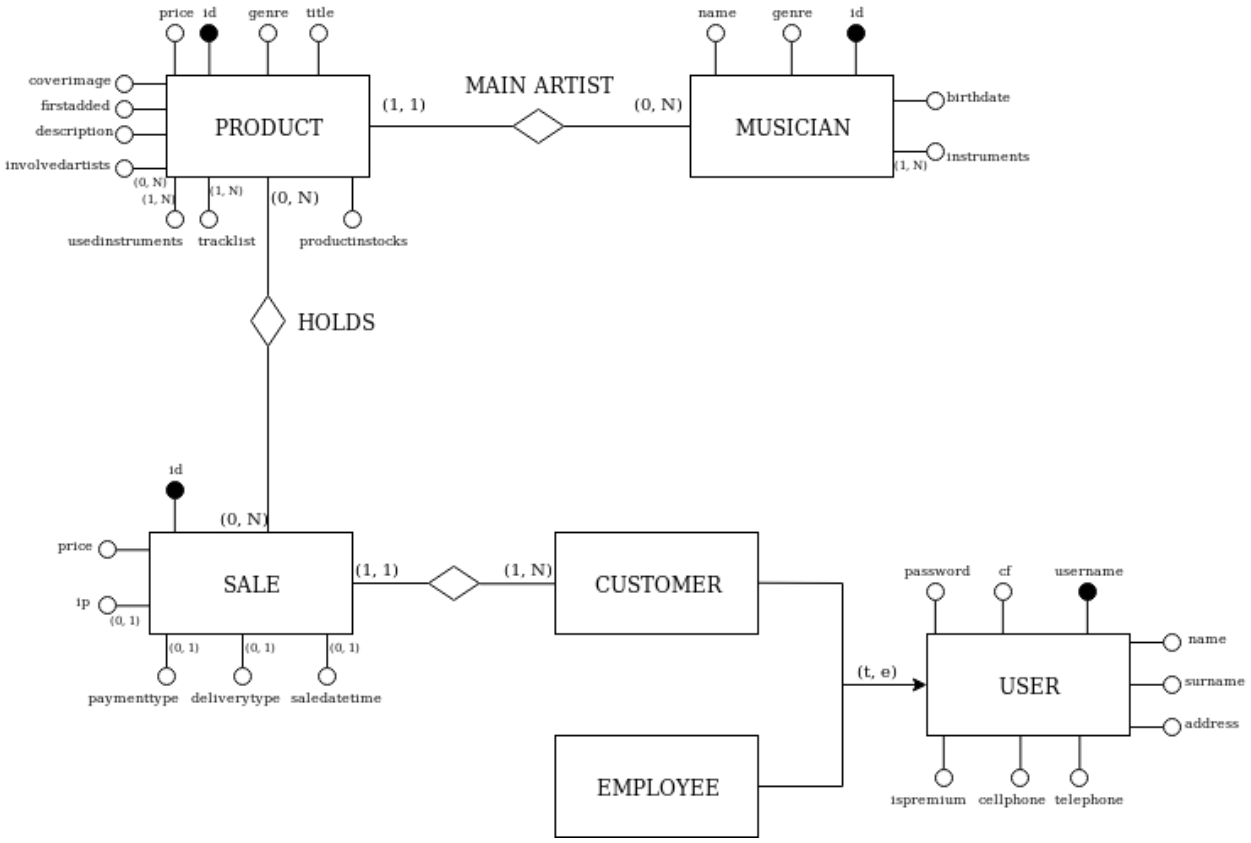
Se il cliente ha fatto già 3 acquisti superiori ai 250 euro l'uno entro l'anno, il sistema gli propone sconti e consegna senza spese di spedizione.

Il personale autorizzato del negozio può inserire tutti i dati dei CD e DVD in vendita. Il personale inserisce anche il numero di pezzi a magazzino. Il sistema tiene aggiornato il numero dei pezzi a magazzino durante la vendita e avvisa il personale del negozio quando un articolo (CD o DVD) scende sotto i 2 pezzi presenti in magazzino.

3 Diagrammi

3.1 Diagramma ER

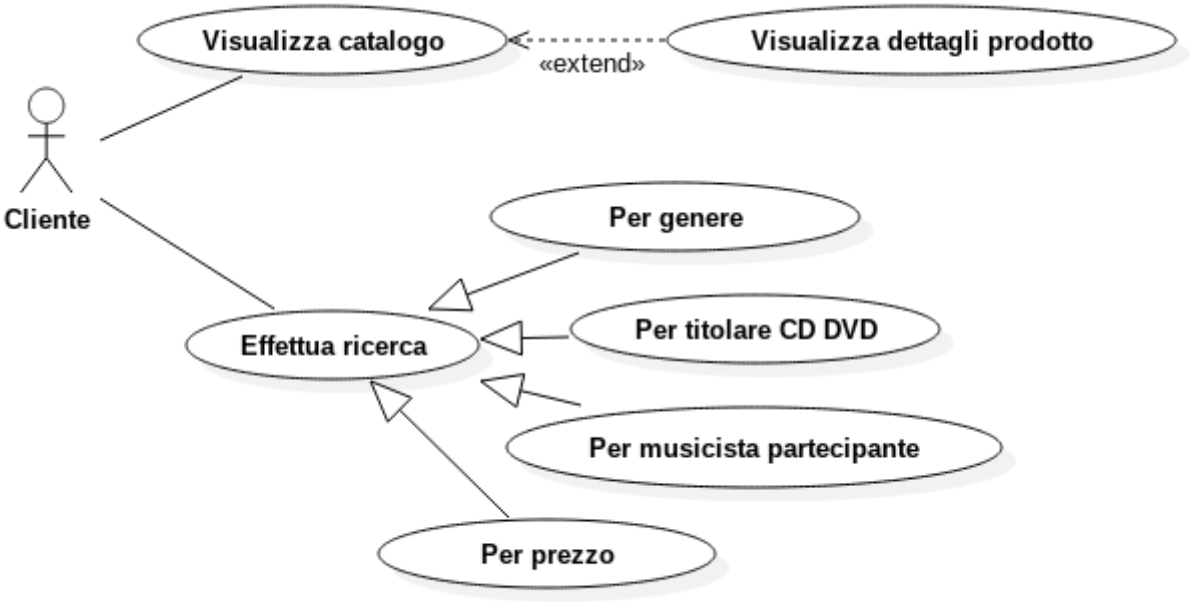
Abbiamo progettato il database relazionale seguendo le specifiche richieste, andando però ad apportare modifiche nella fase di implementazione del sistema: nel sistema implementato, l'entità *user* non ha generalizzazione come raffigurato nello scherma ER poiché abbiamo deciso di implementare solo l'entità *user* alla quale abbiamo aggiunto un attributo booleano (*isEmployee*) che specifica se l'account è di tipo cliente o di tipo impiegato. Per ogni utente, l'entità *SALE* associata rappresenta univocamente il carrello; in seguito ad un effettivo acquisto, *SALE* rappresenta in modo univoco l'acquisto appena effettuato. Dopo aver acquistato dei prodotti viene creato un nuovo carrello con un nuovo *id*. Abbiamo scelto di memorizzare solo gli artisti principali dei prodotti, mentre gli artisti coinvolti sono memorizzati in un campo apposito nella tabella *PRODUCT*, questo per mantenere più semplice la gestione degli artisti.



3.2 Diagrammi dei casi d'uso e relative schede di specifica

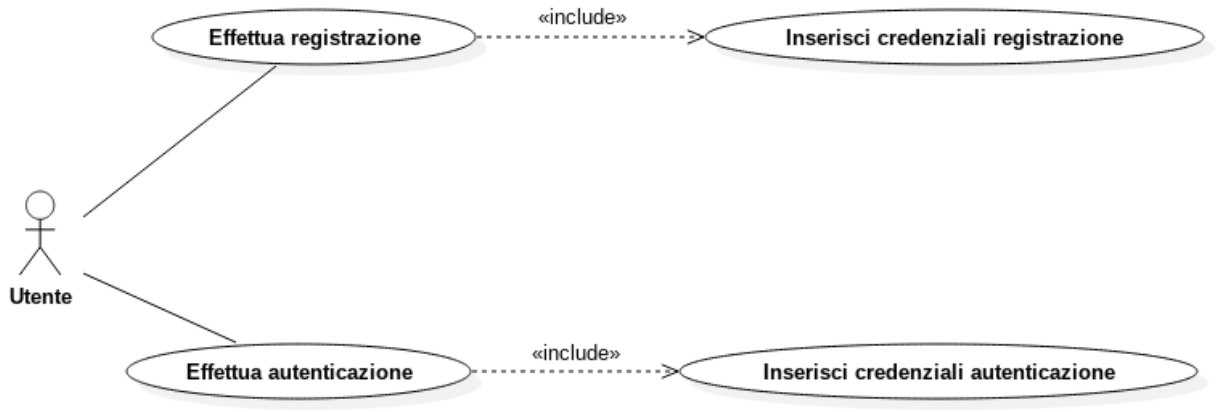
Abbiamo creato quattro diagrammi dei casi d'uso per rappresentare i modi in cui il sistema può essere utilizzato e le funzionalità che mette a disposizione dei suoi utilizzatori, basandoci sui requisiti funzionali delle specifiche del progetto. Per ogni diagramma riportiamo la relativa scheda di specifica.

1) Visualizzazione prodotti



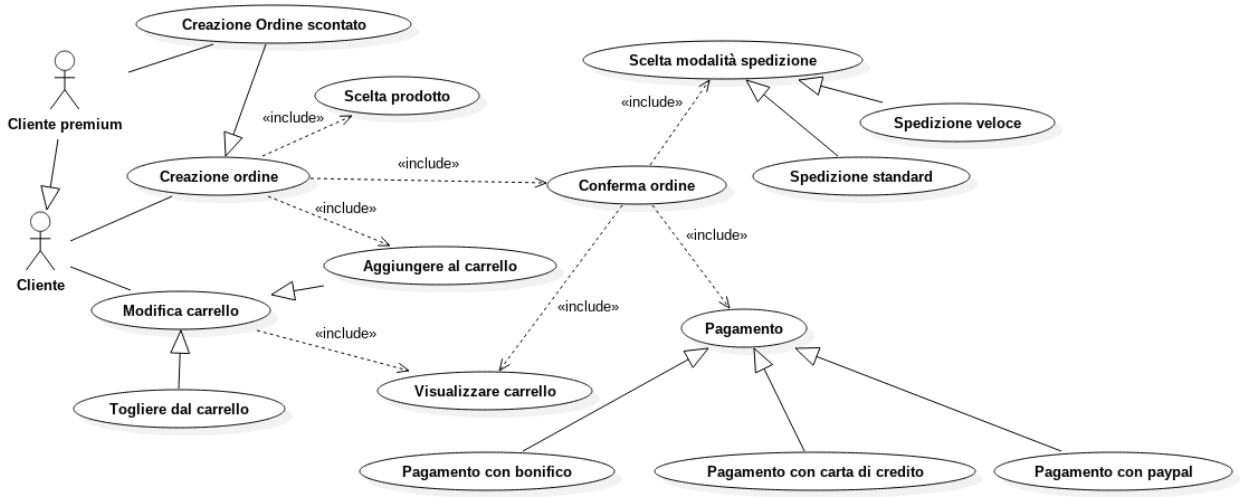
Caso d'uso: VisualizzazioneProdotti
ID: UC1
Attori: Cliente
Sequenza degli eventi: 1) Il caso d'uso inizia quando il Cliente visualizza il catalogo 2) Se il Cliente vuole visualizzare i dettagli di un prodotto: a. Il sistema mostra una finestra con i dettagli del prodotto 3) Se il cliente vuole effettuare una ricerca: a. Il Cliente inserisce i criteri di ricerca
Postcondizioni: Il Cliente ha visualizzato i dettagli del prodotto o effettuato una ricerca

2) Accesso al sistema



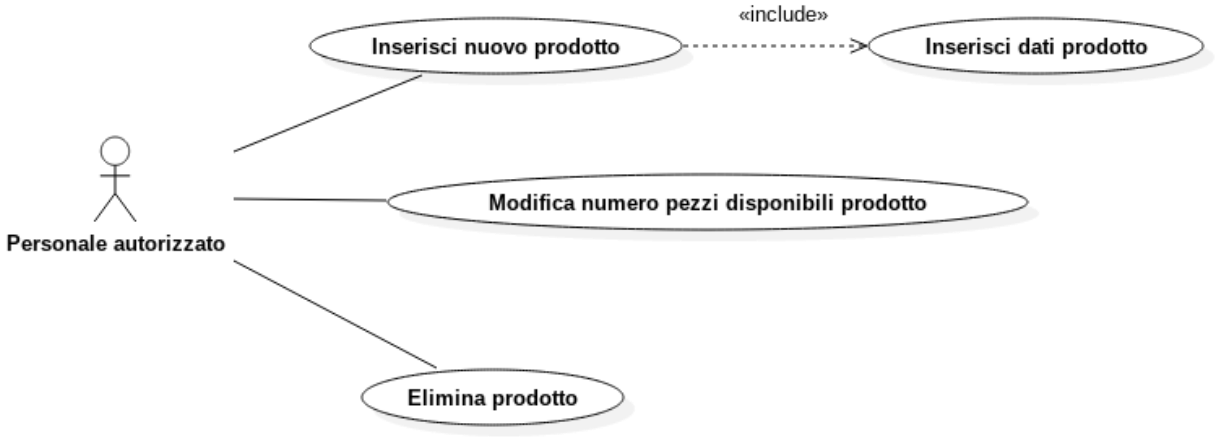
Caso d'uso: AccessoAlSistema
ID: UC2
Attori: Utente
Sequenza degli eventi: 1) Il caso d'uso inizia quando l'Utente vuole registrarsi oppure autenticarsi 2) Se l'Utente vuole registrarsi: a. Inserisce i propri dati personali richiesti dal sistema e conferma la registrazione 3) Se l'Utente vuole autenticarsi: a. L'Utente inserisce username e password e conferma l'inserimento
Postcondizioni: Il Cliente risulta registrato, oppure autenticato al negozio

3) Acquisto prodotti



Caso d'uso: AcquistoProdotti
ID: UC3
Attori: Cliente autenticato
Precondizioni: Il Cliente deve aver effettuato la registrazione al sistema e successivamente, l'autenticazione
Sequenza degli eventi: 1) Il caso d'uso inizia quando il Cliente è autenticato al sistema 2) Se il Cliente vuole acquistare un prodotto: a. Seleziona il prodotto desiderato dal catalogo b. Aggiunge il prodotto al carrello nella quantità desiderata c. Visualizza il carrello e conferma la scelta d. Seleziona i metodi di pagamento e spedizione 3) Se il Cliente vuole modificare il carrello: a. Inserisce un nuovo prodotto dal catalogo **Oppure** b. Visualizza il carrello c. Seleziona gli elementi da eliminare d. Conferma
Postcondizioni: Il Cliente ha completato un acquisto oppure ha modificato il contenuto del carrello

4) Aggiornamento dati magazzino

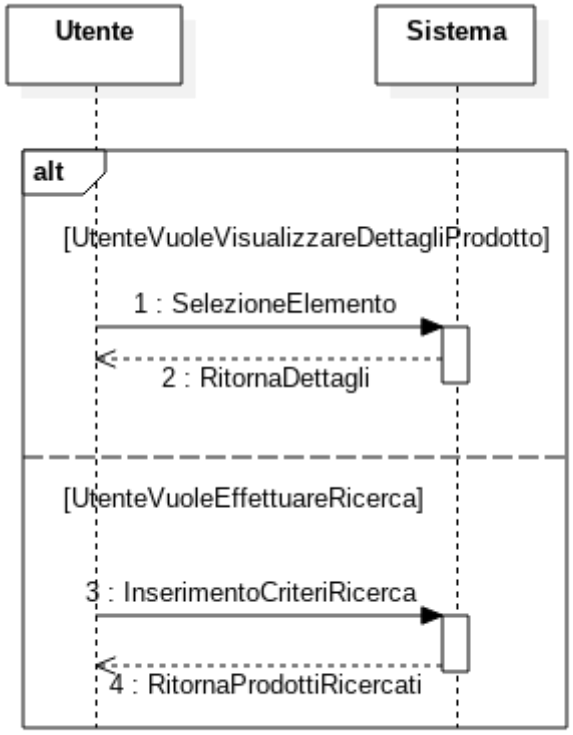


Caso d'uso: AggiornamentoDatiMagazzino
ID: UC4
Attori: Impiegato
Precondizioni: I dati dell'Impiegato sono già presenti nel sistema; l'Impiegato ha effettuato l'autenticazione come personale autorizzato
Sequenza degli eventi: 1) Il caso d'uso inizia quando l'Impiegato è autenticato 2) L'Impiegato seleziona il pulsante di modifica del catalogo 3) L'impiegato può: a. Inserire nuovi CD/DVD e dati associati b. Aggiornare le disponibilità dei CD/DVD desiderati, in base alla loro presenza effettiva in magazzino c. Eliminare prodotti non più in vendita
Postcondizioni: L'Impiegato ha modificato il catalogo di sistema

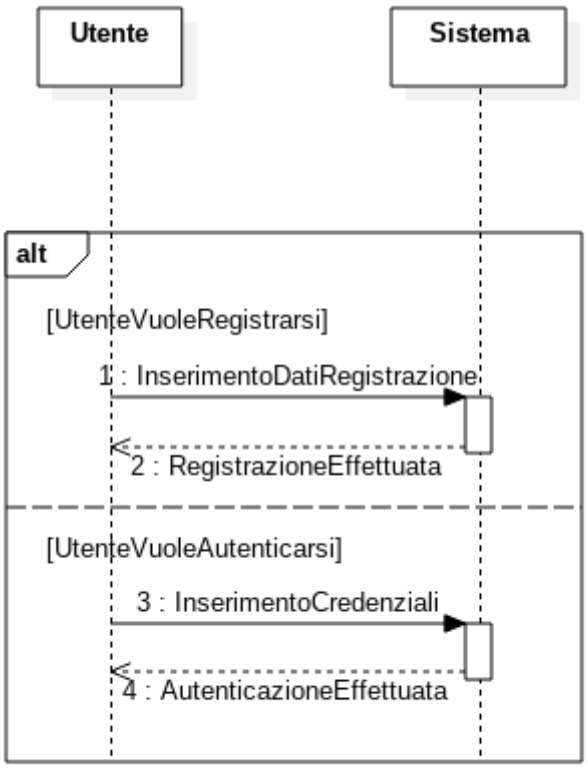
3.3 Diagrammi di sequenza dei casi d'uso

Per ogni diagramma dei casi d'uso abbiamo costruito il corrispondente diagramma di sequenza.

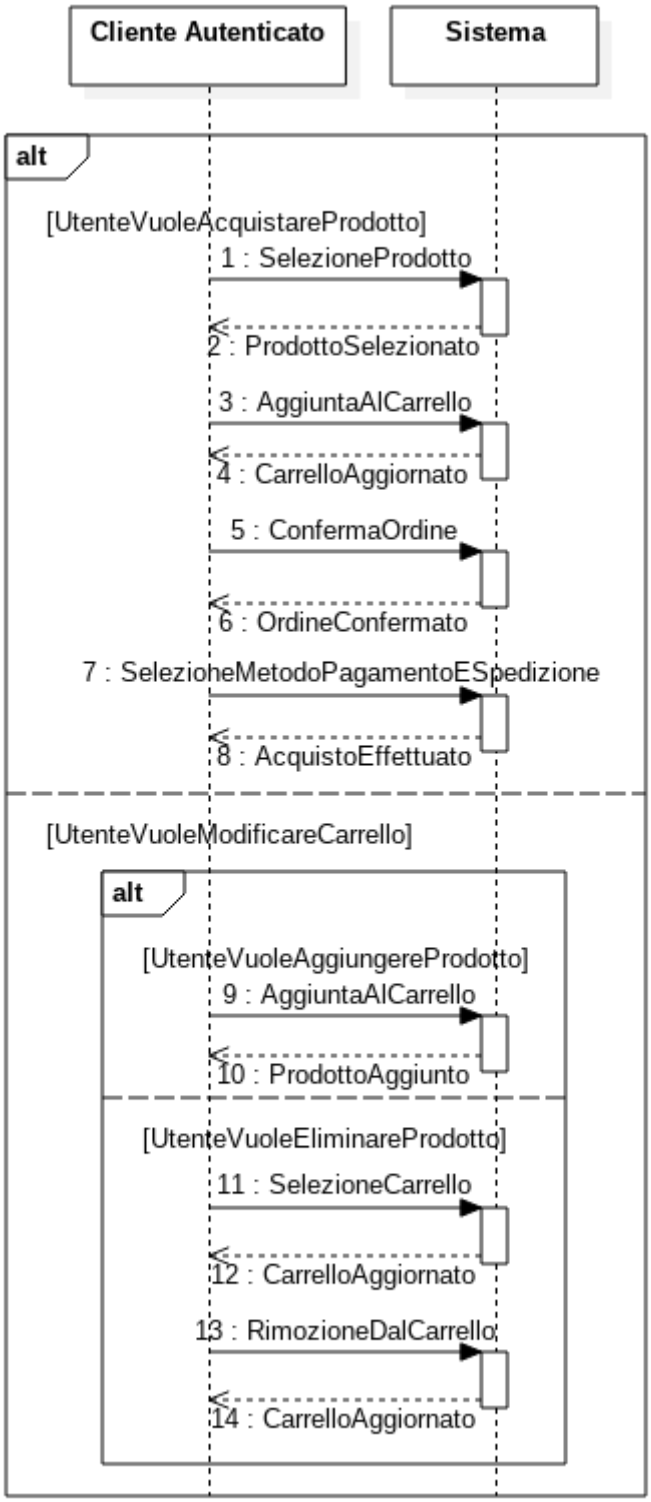
1) Diagramma di sequenza del 1° caso d'uso: visualizzazione prodotti



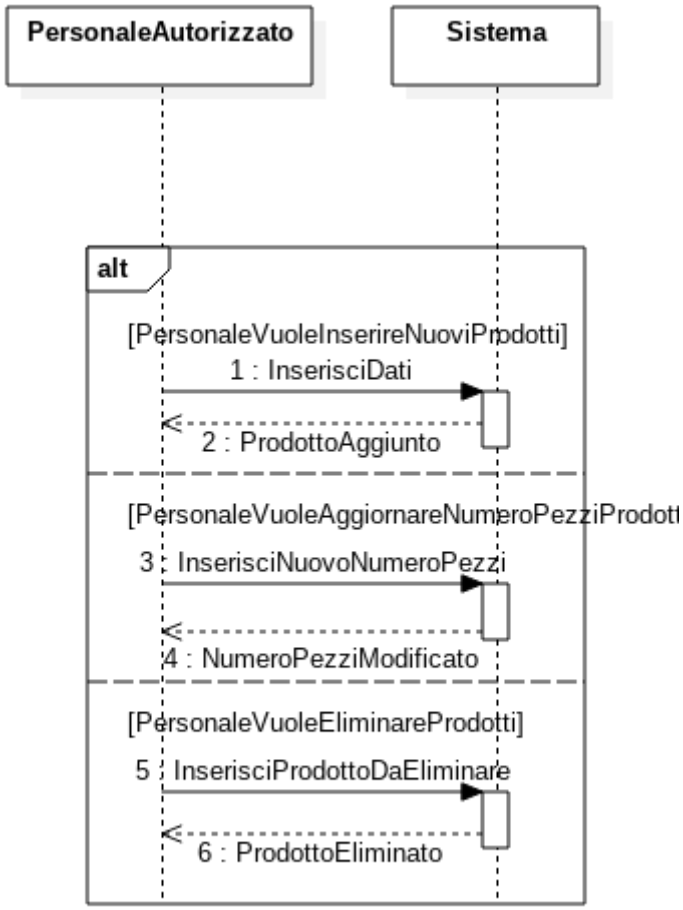
2) Diagramma di sequenza del 2° caso d'uso: accesso al sistema



3) Diagramma di sequenza del 3° caso d'uso: acquisto prodotti



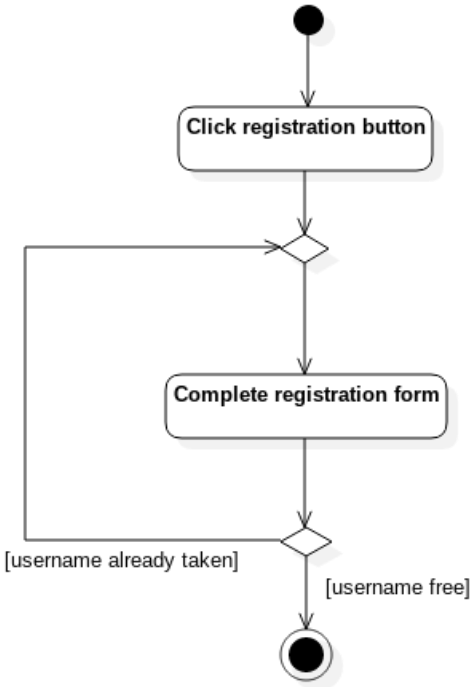
4) Diagramma di sequenza del 4° caso d'uso: aggiornamento dati magazzino



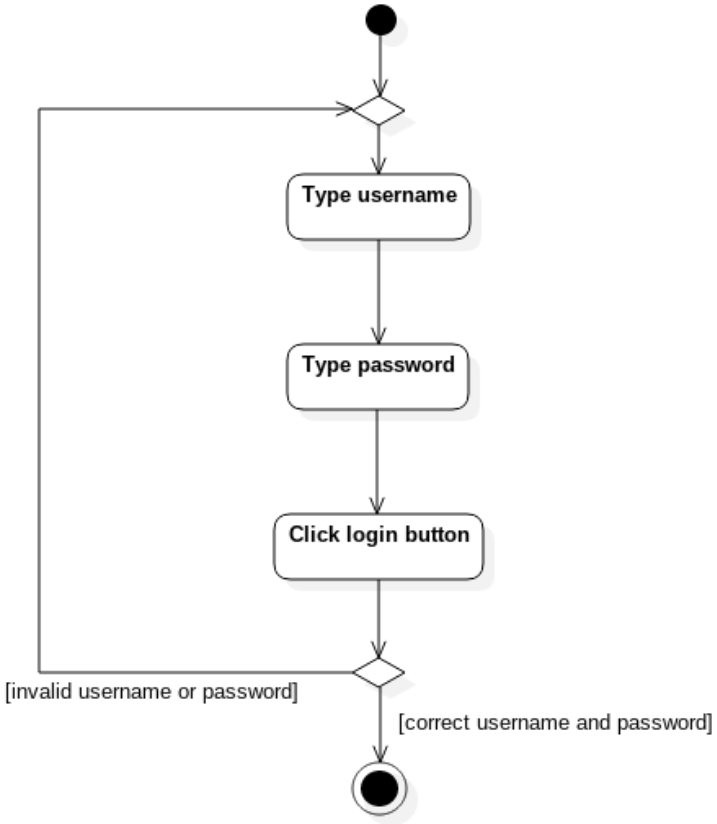
3.4 Diagrammi delle attività

Successivamente abbiamo creato i diagrammi delle attività per descrivere le attività principali che possono compiere gli utenti in relazione alle specifiche del progetto. Abbiamo preferito suddividere queste attività in molte piccole attività distinte, questo per preservare la leggibilità e la compattezza dei diagrammi. Abbiamo quindi modellato i comportamenti di ciascuna con i relativi stati interni, stato iniziale e stato finale, costruendo il flusso di azioni che permette di raggiungere lo stato finale.

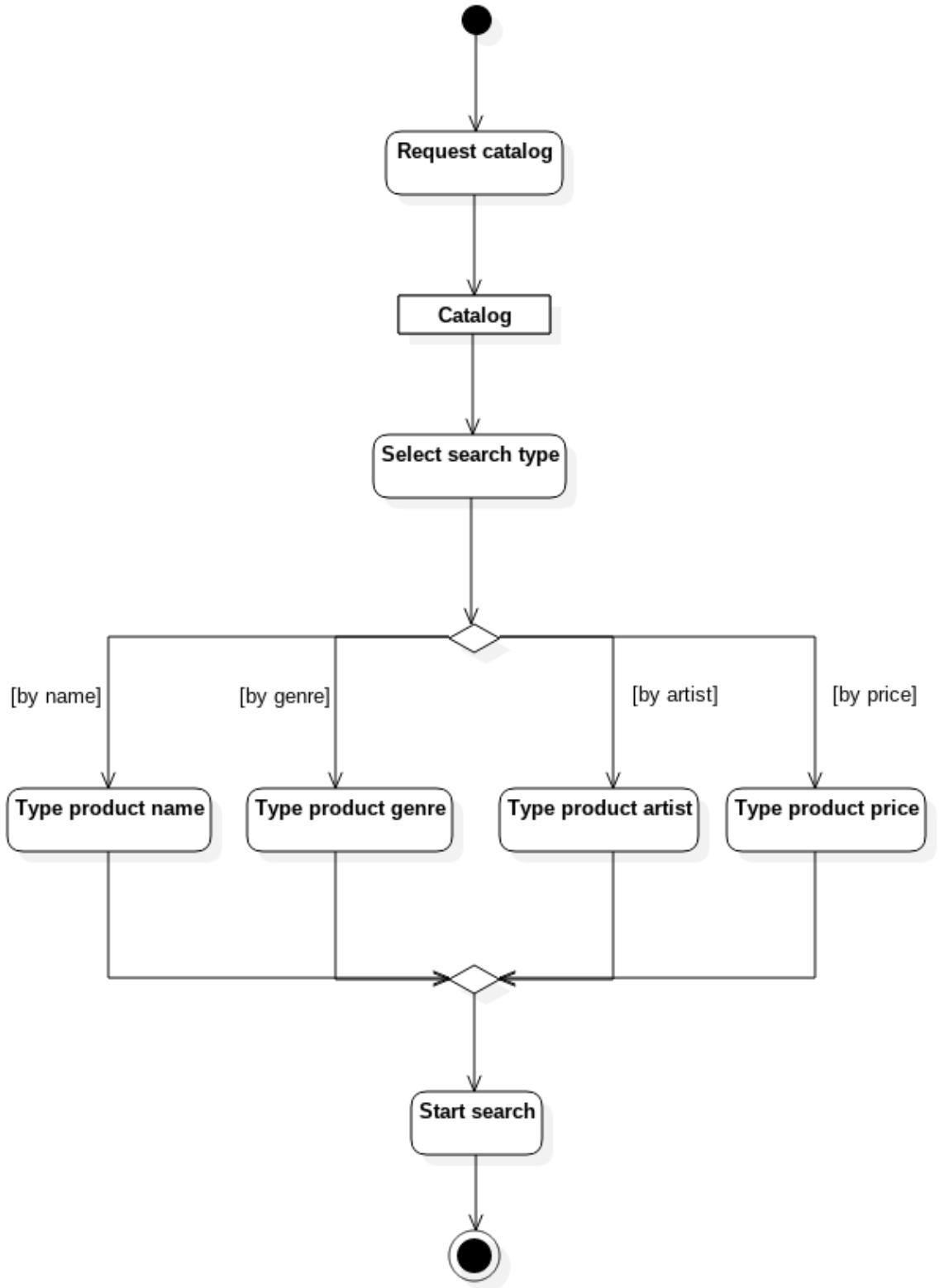
1) Registrazione



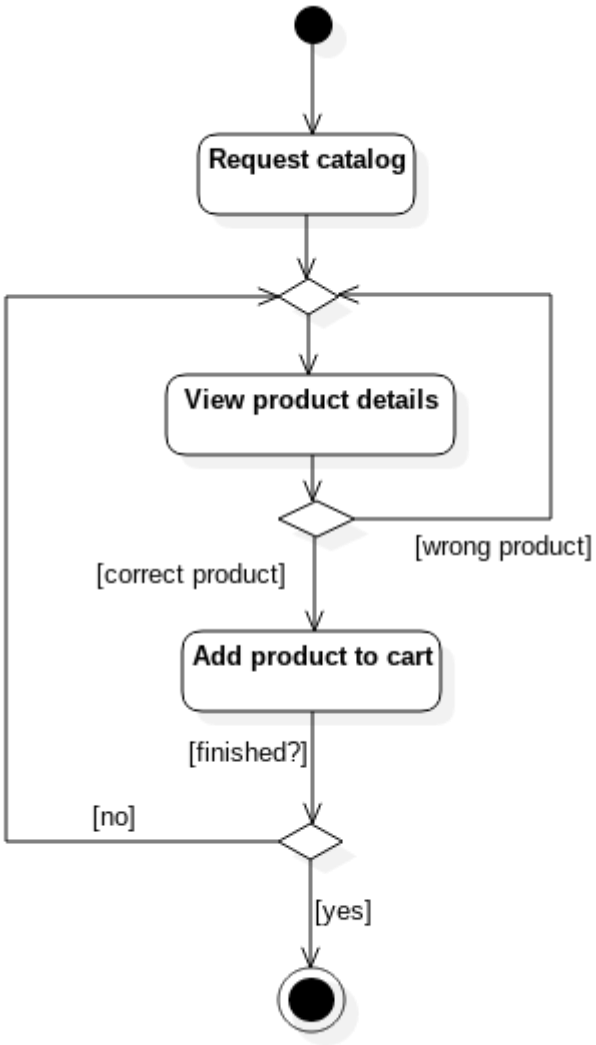
2) Login



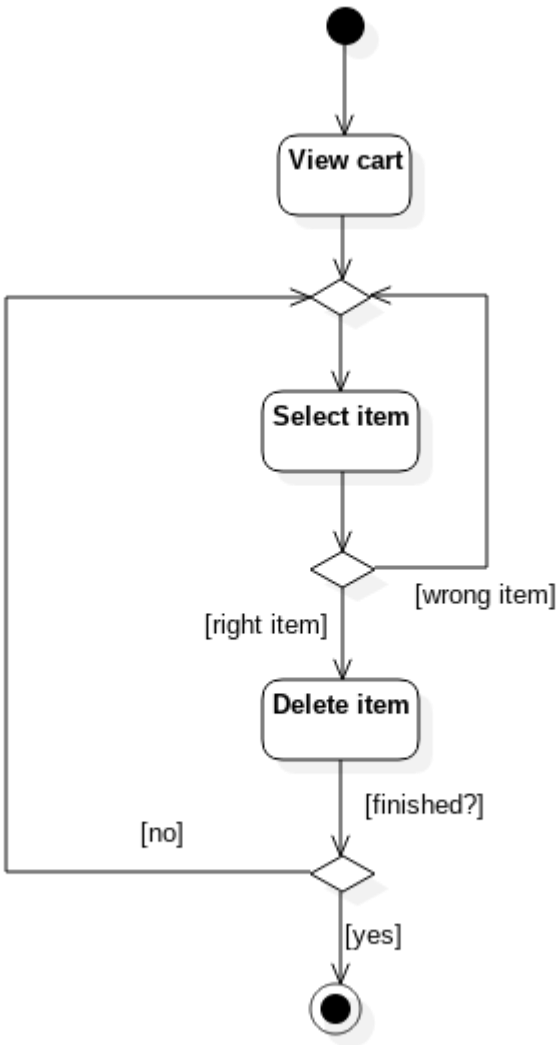
3) Ricerca prodotti



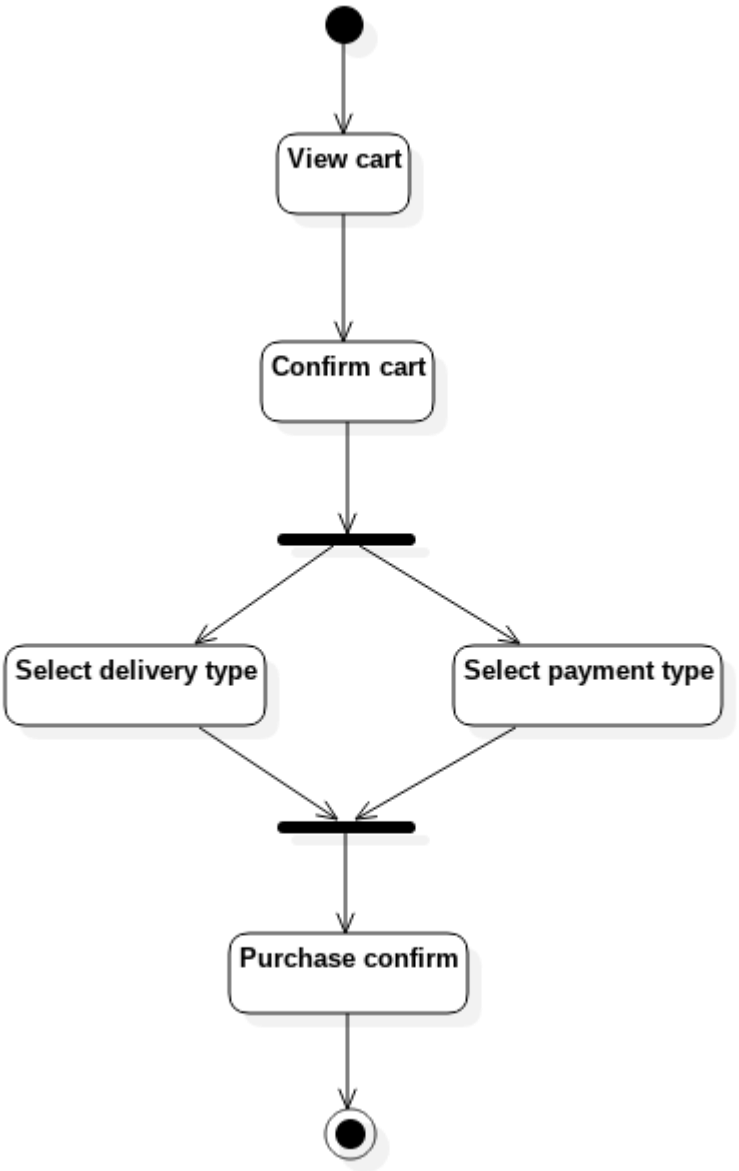
4) Aggiunta al carrello



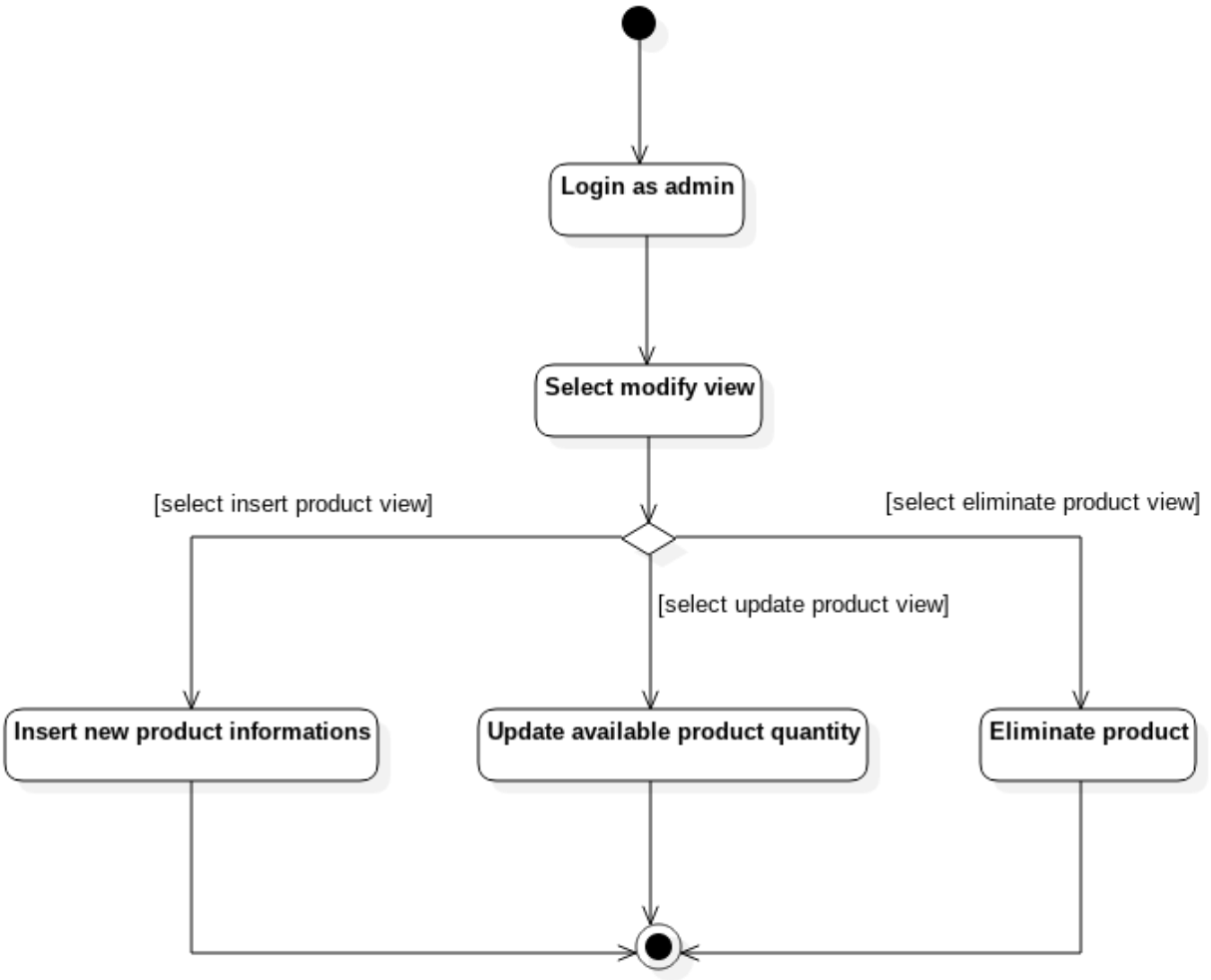
5) Rimozione dal carrello



6) Acquisto del carrello

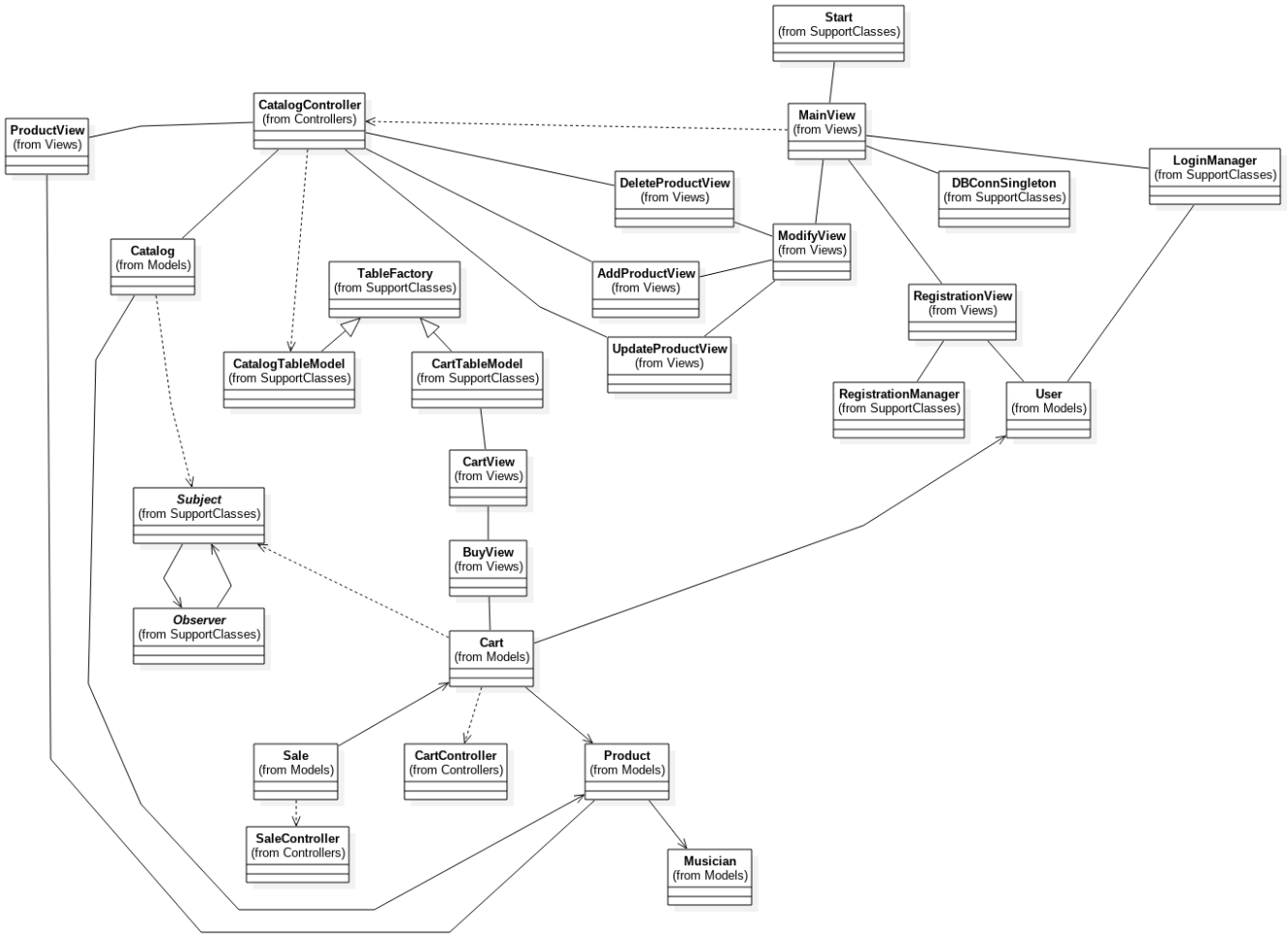


7) Aggiornamento database



3.5 Diagramma delle classi

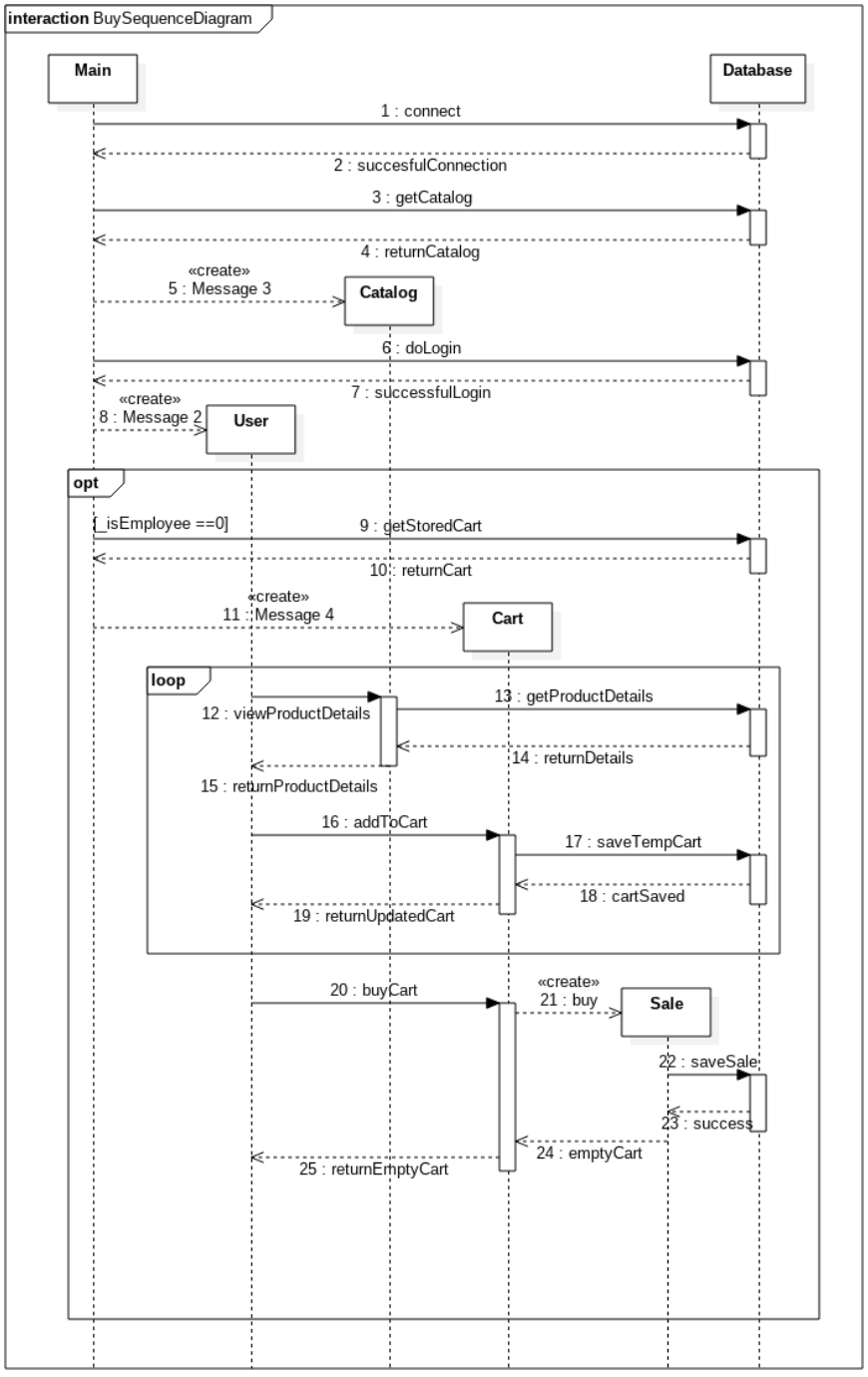
Dopo aver identificato le attività principali che possono svolgere gli utenti, abbiamo costruito il diagramma delle classi che contiene tutte le classi che saranno sviluppate successivamente, con le relative associazioni, specificando solamente il loro nome e il package di provenienza.



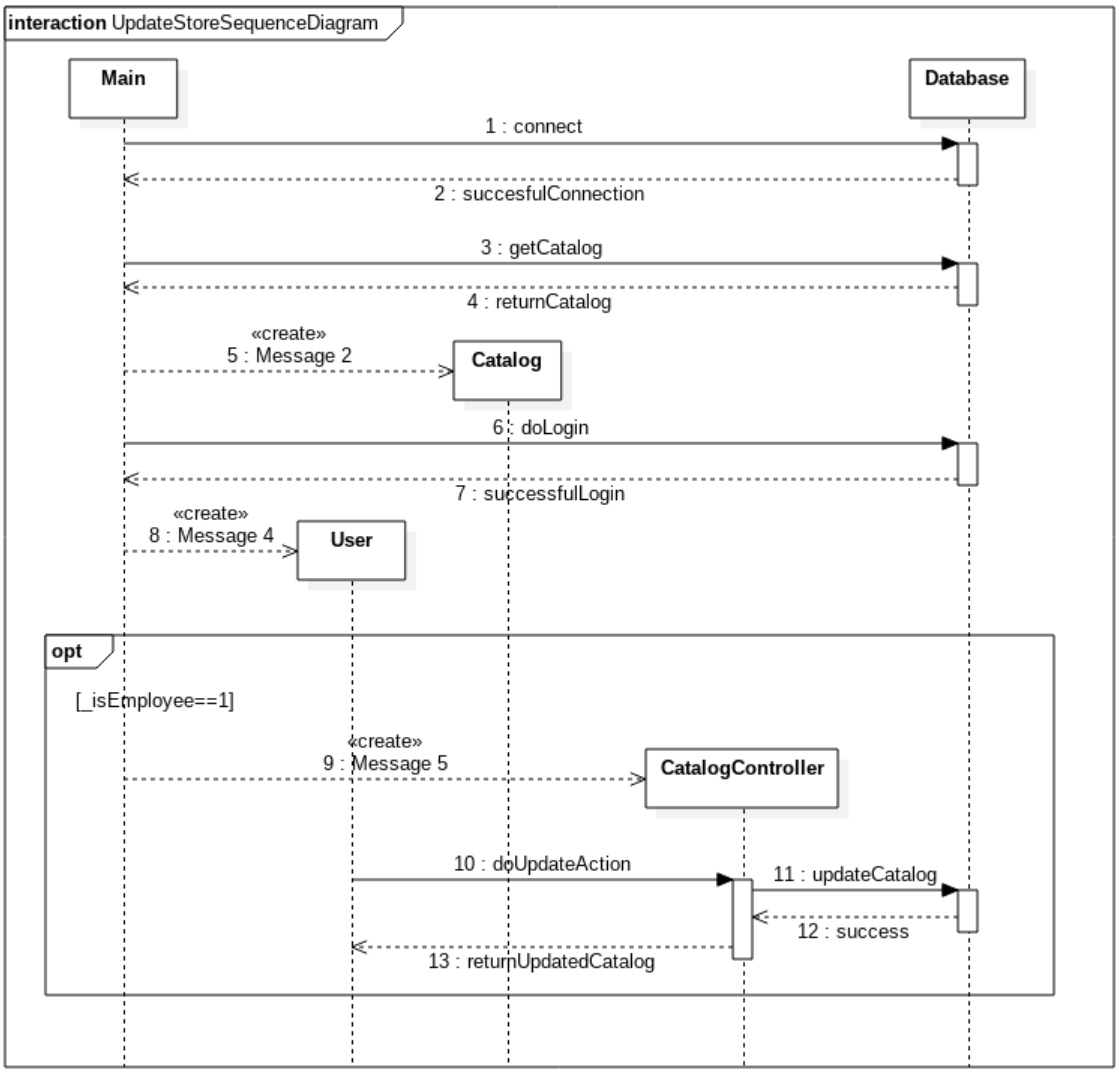
3.6 Diagrammi di sequenza del diagramma delle classi

Infine abbiamo creato due diagrammi di sequenza relativi all'interazione tra front-end (main del programma) e back-end (database), modellandoli sulla base del diagramma delle classi. Le sequenze scelte sono quelle principali, ovvero l'acquisto di prodotti da parte di un utente, e la modifica del database da parte di un impiegato.

1) Diagramma di sequenza: acquisto prodotto



2) Diagramma di sequenza: aggiornamento catalogo



4 Scelte di implementazione e assunzioni

Svolta la parte di progettazione del sistema, abbiamo deciso di implementare il software con il linguaggio di programmazione Java, un linguaggio orientato agli oggetti: questa scelta ci ha permesso di focalizzarci meglio sulle entità da rappresentare e sui dati da elaborare. La parte di visualizzazione grafica è affidata alle librerie Swing e AWT di Java, le quali si occupano di fornire un'interfaccia semplice ed intuitiva con finestre, bottoni, tabelle.

Per quanto riguarda l'implementazione del database, ci siamo affidati ad un database di tipo relazionale basato su PostgreSQL, il quale è stato implementato in locale: questo per evitare problemi provenienti da fattori esterni, come connessioni di rete inefficienti o insicure. Il database è stato poi riempito con una serie di prodotti, musicisti e utenti sufficienti ad eseguire la fase di testing. Per eseguire interrogazioni e modifiche sul database infine, è stato utilizzato il driver JDBC.

Le immagini di copertina degli album, le immagini di bottoni e altre immagini estetiche sono state inserite anch'esse in una cartella in locale. L'attributo *coverimage* della tabella *PRODUCT* contiene il percorso per risalire all'immagine nella cartella in locale.

Passando alle scelte di implementazione dei vari requisiti, abbiamo deciso di non implementare suggerimenti mirati agli utenti basati sugli acquisti precedenti. L'inserimento di un prodotto da parte di un impiegato non tiene conto di alcune informazioni del prodotto, come la tracklist e gli artisti e strumenti presenti, in quanto non visibili all'utente nella vista di dettaglio del prodotto. Non abbiamo inoltre implementato form aggiuntivi di inserimenti dei dati privati per i vari metodi di pagamento (bonifico, paypal, carta di credito).

Requisiti abbastanza liberi ci hanno poi permesso di fare delle assunzioni in modo che lo sviluppo del codice non risultasse troppo complicato. In particolare:

- Gli impiegati accedono al sistema con un account di amministratore già presente nel database, e non possono creare nuovi account di qualsiasi tipo.
- I clienti non possono accedere con account di amministratore e, di conseguenza, non possono compiere azioni riservate agli impiegati.
- Gli impiegati e amministratori non possono comportarsi da clienti quando hanno effettuato l'accesso con l'account di admin: è quindi vietato visualizzare, riempire carrelli ed effettuare acquisti;
- CD e DVD sono trattati allo stesso modo, entrambi sono oggetti di tipo prodotto con gli stessi attributi e la loro visualizzazione non cambia.

5 Pattern utilizzati

Di seguito elenchiamo tutti i pattern che abbiamo utilizzato nella progettazione: il primo, il pattern MVC è un pattern architetturale, mentre i successivi pattern elencati sono pattern creazionali come Singleton e Factory, e pattern comportamentali come Observer e Iterator.

5.1 Pattern MVC

Il pattern MVC è il principale pattern di architettura che è stato scelto per questo progetto. Esso si basa su tre elementi principali: modelli, viste e controller, ognuno con le proprie caratteristiche. In particolare:

- Modello: componente centrale, che cattura il comportamento dell'applicazione in termini del dominio del problema, indipendentemente dall'interfaccia utente;
- Vista: rappresentazione in output di informazioni (grafico, diagramma…);
- Controller: accetta l'input e lo converte in comandi per il modello e/o vista.

Nell'ottica di questo progetto, i modelli scelti sono derivati dalla base di dati progettata precedentemente: oggetti come utenti, musicisti, prodotti e vendite riprendono gli attributi principali delle tabelle della base di dati. Sono stati aggiunti poi altri due modelli fondamentali, basati sugli oggetti appena definiti, che arricchiscono e completano il comportamento del programma:

- Catalogo: insieme (*ArrayList*) di prodotti;
- Carrello: insieme (*ArrayList*) di prodotti, utente e un attributo che indica il prezzo totale

Questi modelli dispongono di costruttori e metodi *set* e *get*, che permettono la loro costruzione, modifica e visualizzazione.

Le viste implementate costituiscono le varie schermate che può trovarsi davanti l'utente al momento dell'esecuzione del programma: sono presenti una *MainView* da cui è possibile vedere il catalogo, effettuare ricerche, visualizzare i dettagli del prodotto, effettuare login o registrazione e vedere il carrello; una *CartView* che illustra i prodotti presenti nel carrello virtuale; una *BuyView* che permette di acquistare il carrello scelto, e altre schermate secondarie. Queste viste si occupano dell'interazione principale con l'utente: egli andrà ad interagire con bottoni, campi di testo, tabelle. Le viste passeranno questi input a determinati metodi dei controller, che provvederanno a processarli.

I controller infine sono i responsabili delle modifiche alle strutture dati: i metodi a disposizione servono infatti ad accettare gli input delle viste, processarli e modificare modelli e database. Abbiamo individuato tre controller principali:

- CartController: modifica oggetti di tipo Carrello;
- CatalogController: modifica oggetti di tipo Catalogo;
- SaleController: modifica oggetti di tipo Vendita.

Tutti e tre si occupano poi di eseguire interrogazioni o aggiornare il database, con metodi che fanno uso del driver JDBC.

5.2 Pattern Singleton

Questo pattern assicura che una classe abbia solo un'istanza e provvede un punto di accesso globale a tale istanza. Viene utilizzato per realizzare la classe *DBconnSingleton*, il cui scopo è quello di creare una connessione con il database in locale, e di inizializzare l'attributo *connection* (che rappresenta tale connessione).

All'interno del programma viene creato una sola volta, nella classe *MainView*, classe chiamata direttamente dal *Main* del programma.

L'implementazione di questo pattern prevede che la classe interessata abbia quindi un unico costruttore privato, in modo da impedire l'istanziazione diretta della classe. La classe fornisce inoltre un metodo get statico (*getConn*) che restituisce l'istanza della classe (l'attributo *connection* inizializzato) creata alla prima chiamata del metodo *getInstance*.

5.3 Pattern Factory

Il pattern factory definisce un'interfaccia per creare diversi oggetti, ma lascia decidere alle sottoclassi quale oggetto istanziare.Viene utilizzato nella classe *TableFactory* per definire le diverse tipologie di interfaccia con cui creare le tabelle all'interno del programma.

All'interno di questa classe vengono create due tipologie di oggetti di *DefaultTableModel* diversi, a seconda del tipo di View che si vuole utilizzare (*MainView* o *CartView*).

Il metodo *getTableModel* prende quindi come input una stringa che indica il tipo di view in utilizzo, chiama i metodi opportuni per costruire la tabella specifica e restituisce tale tabella alla classe chiamante.

5.4 Pattern Observer

Il pattern Observer permette di definire una dipendenza uno a molti in modo tale che quando un oggetto (Subject) cambia stato, tutti quelli che ne dipendono (Observer) vengono automaticamente notificati del fatto ed aggiornati di conseguenza. In questo progetto i Subject vengono identificati dagli oggetti *Cart*, *Catalog* e *Product*, che rappresentano rispettivamente un carrello di prodotti pronti ad essere comprati, il catalogo dei prodotti e il modello di un prodotto. I rispettivi Observer invece sono rappresentati dalle classi *CartView*, *MainView*, e *ProductView*. Ogni volta che i Subject si modificano, le tre View (gli Observer) aggiornano i propri dati e li mostrano a video aggiornati.

5.5 Pattern Iterator

Il pattern Iterator viene utilizzando quando si ha la necessità di voler accedere a tutti gli elementi di una generica collezione senza esporne la struttura interna. Inoltre, gli utilizzatori devono poter accedere agli elementi del contenitore contemporaneamente.

L'idea alla base del pattern Iterator è quella di avere un determinato oggetto, detto *contenitore*, che contiene altri oggetti al suo interno. A questo contenitore viene associato un nuovo oggetto, chiamato iteratore, che rappresenta un indice per scorrere l'interno del contenitore.

Nel nostro programma, il pattern iterator ha trovato utilizzo nel metodo *next()* utilizzato per ciclare sugli oggetti di tipo *ResultSet*, cioè i risultati delle interrogazioni fatte al database. Il metodo permette in questo caso di accedere ad ogni riga dei risultati, non curandosi della complessità di questi ultimi (più o meno colonne, tipi di campo diversi…). Non è invece presente un metodo *hasNext()*, in quanto, in questo caso, basta usare il metodo *next()* all'interno di un ciclo per verificare di arrivare alla fine di quel *ResultSet*.

6 Fase di test

Per il test del sistema, ci siamo affidati a una tipologia di testing dinamico, ossia provare con diversi tipi di input le diverse funzioni e metodi. In particolare abbiamo provato a inserire input di tipo corretto e input sbagliati, per vedere come reagiva il programma e per vedere di conseguenza il suo comportamento. Questo è stato fatto soprattutto durante la fase di implementazione di codice: ad ogni nuova funzione o metodo implementato, sono poi seguiti dei piccoli test, che ci hanno permesso di agire anzitempo ed evitare di dover ristrutturare più avanti parti di codice più grandi.

Successivamente abbiamo testato insieme diversi componenti del programma per provare a soddisfare i vari casi d'uso, come la fase di login, l'acquisto di un carrello o la modifica del catalogo. Sono stati utilizzati diversi scenari, ad esempio:

- Utente che si registra, sfoglia il catalogo, aggiunge prodotti al carrello e lo acquista;
- Utente che acquista tre ordini superiori ai 250 euro l'uno, e diventa premium;
- Impiegato che accede tramite l'account di amministratore, e aggiorna il database.

Abbiamo deciso, data la difficoltà non troppo elevata, e la relativamente bassa quantità di codice, di non usare strumenti per l'automatizzazione della fase di testing: tutti i vari test sono stati eseguiti manualmente da noi.