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 assuzioni 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.

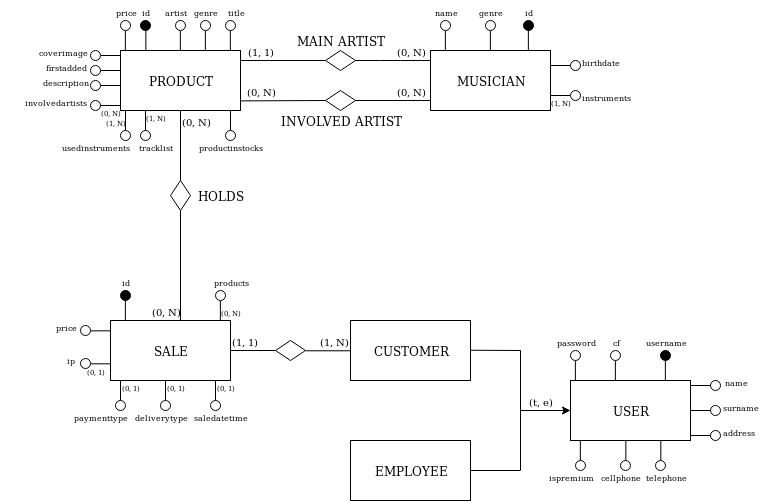
Se il cliente ha fatto già 3 acquisti superiori ai 250 euro l’uno entro l’anno, il sistema gli propone sconti e consegna senza spese di spedizione.

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*.

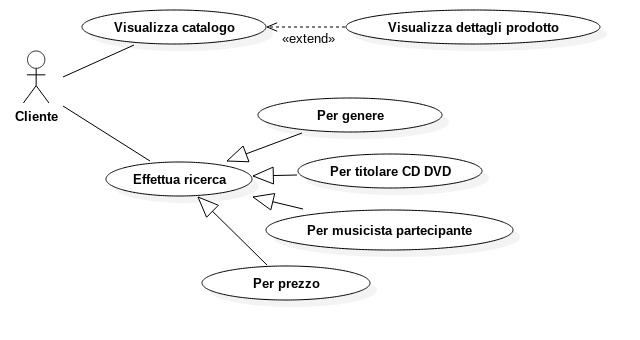


3.2 Diagrammi dei casi d’uso e relative schede di specifica

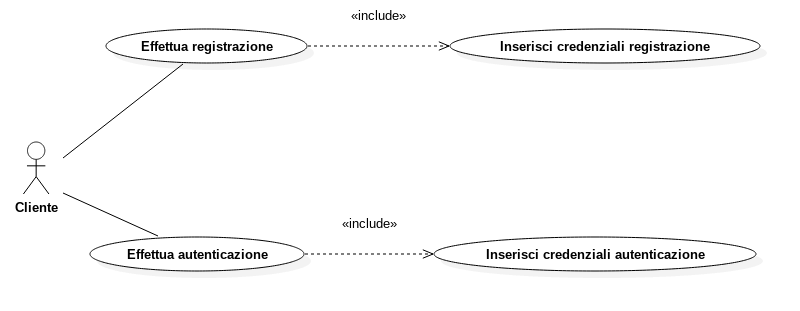
Abbiamo creato quattro diagrammi dei casi d’uso per rappresentare i modi in cui il sistema può essere utilizzato e le funzionalità che mette a disposizione dei suoi utilizzatori, basandoci sui requisiti funzionali delle specifiche del progetto. Per ogni diagramma riportiamo la relativa scheda di specifica.

|  |
| --- |
| **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:    1. Il sistema mostra una finestra con i dettagli del prodotto 3. Se il cliente vuole effettuare una ricerca:    1. Il Cliente inserisce i criteri di ricerca |
| **Postcondizioni:**  Il Cliente ha visualizzato i dettagli del prodotto o effettuato una ricerca |

1. Visualizzazione prodotti

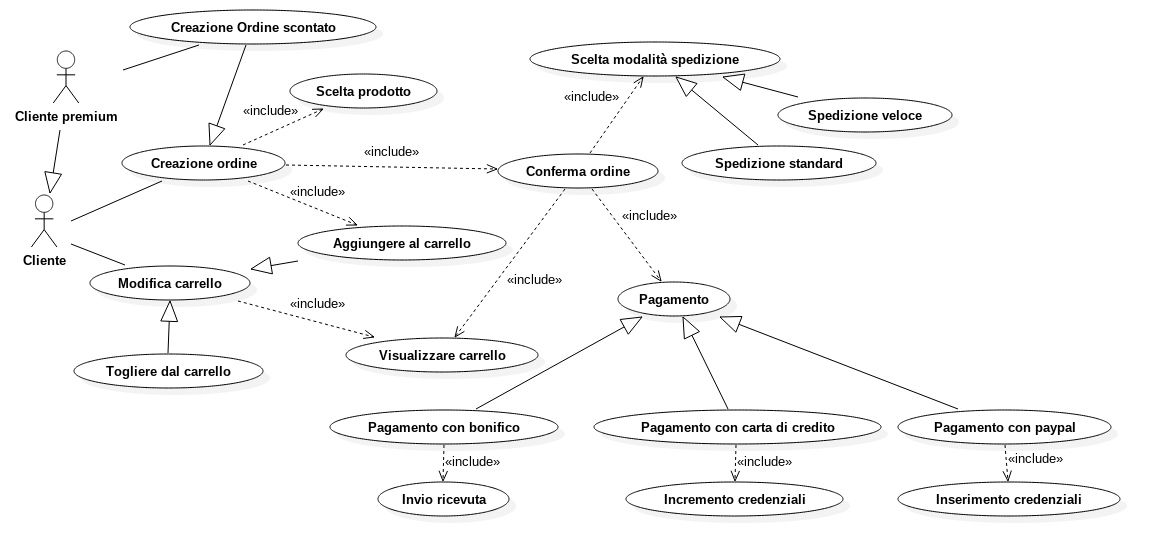


1. Accesso al sistema



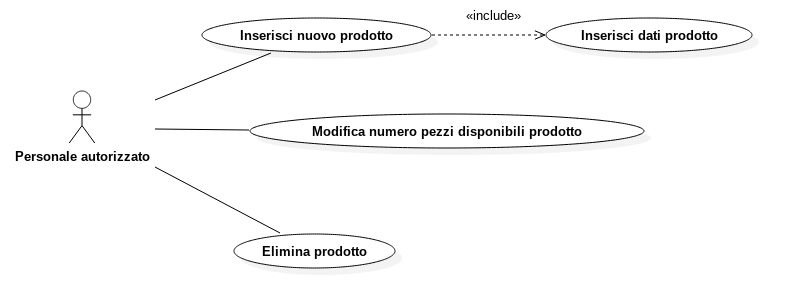
|  |
| --- |
| **Caso d’uso: AccessoAlSistema** |
| **ID:** UC2 |
| **Attori:** Cliente |
| **Sequenza degli eventi:**   1. Il caso d’uso inizia quando il Cliente vuole registrarsi oppure autenticarsi 2. Se il Cliente vuole registrarsi:    1. Inserisce i propri dati personali richiesti dal sistema e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

1. 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:    1. Seleziona il prodotto desiderato dal catalogo    2. Aggiunge il prodotto al carrello    3. Visualizza il carrello e conferma l’acquisto    4. Seleziona i metodi di pagamento e spedizione 3. Se il Cliente vuole modificare il carrello:    1. Inserisce un nuovo prodotto dal catalogo   **Oppure**   * 1. Visualizza il carrello   2. Seleziona gli elementi da eliminare   3. Conferma |
| **Postcondizioni:**  Il Cliente ha completato un acquisto oppure ha modificato il contenuto del carrello |

1. Aggiornamento dati magazzino

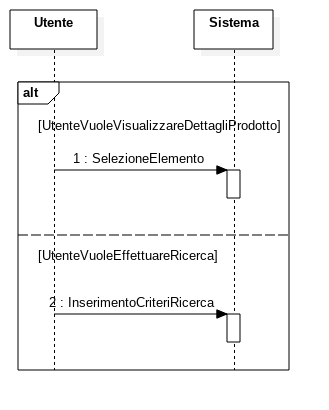


|  |
| --- |
| **Caso d’uso: AggiornamentoDatiMagazzino** |
| **ID:** UC4 |
| **Attori:** Impiegato |
| **Precondizioni:** I dati dell’Impiegato sono già presenti nel sistema; l’Impiegato ha effettuato l’autenticazione come personale autorizzato |
| **Sequenza degli eventi:**   1. Il caso d’uso inizia quando l’Impiegato è autenticato 2. L’Impiegato seleziona il pulsante di modifica del catalogo 3. L’impiegato può:    1. Inserire nuovi CD/DVD e dati associati    2. Aggiornare dati di CD/DVD    3. Eliminare prodotti non più in vendita    4. Aggiornare le disponibilità dei CD/DVD desiderati, in base alla loro presenza effettiva in magazzino |
| **Postcondizioni:**  L’Impiegato ha modificato il catalogo di sistema |

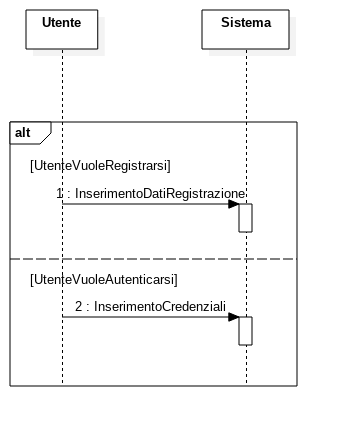
3.3 Diagrammi di sequenza dei casi d’uso

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

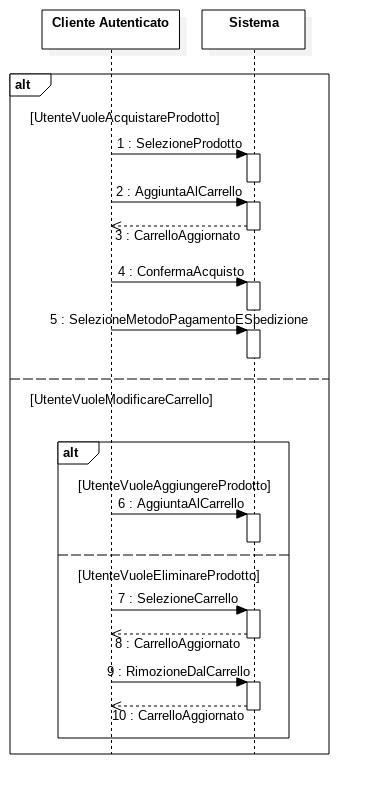
1. Diagramma di sequenza del 1° caso d’uso: visualizzazione prodotti



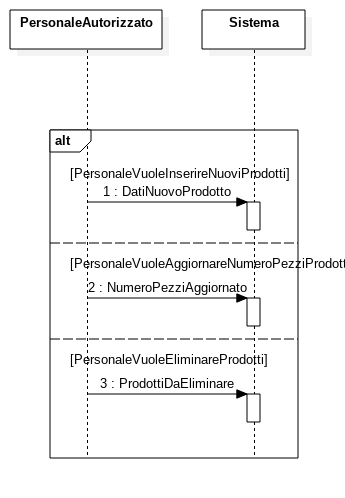
1. Diagramma di sequenza del 2° caso d’uso: accesso al sistema



1. Diagramma di sequenza del 3° caso d’uso: acquisto prodotti



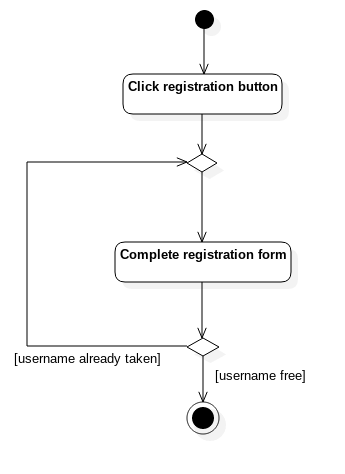
1. Diagramma di sequenza del 4° caso d’uso: aggiornamento dati magazzino



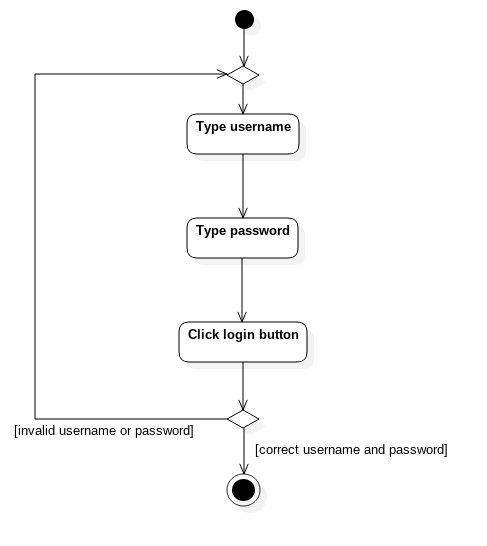
3.4 Diagrammi delle attività

Abbiamo creato sette diagrammi delle attività per descrivono i comportamenti delle tipologie di utenti che utilizzano il sistema. I tre diagrammi riguardano: utente generico, cliente e impiegato. Abbiamo quindi modellato i comportamenti di ciascuno con i relativi stati interni, stato iniziale e stato finale, costruendo il flusso di azioni che permette di raggiungere lo stato finale.

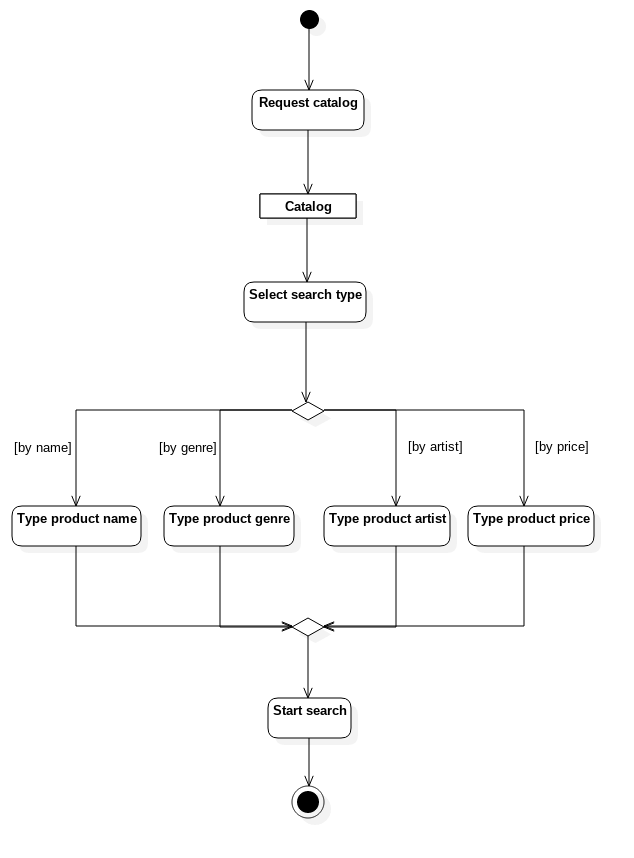
1. Registrazione



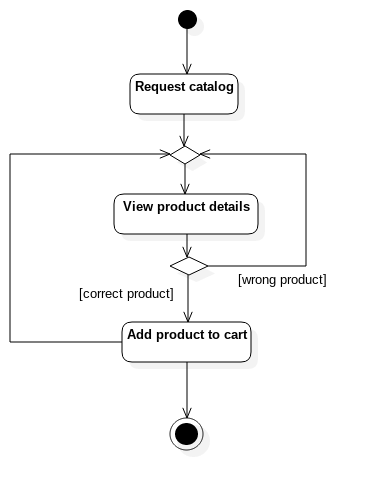
1. Login



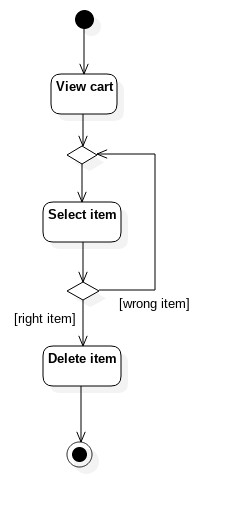
1. Ricerca



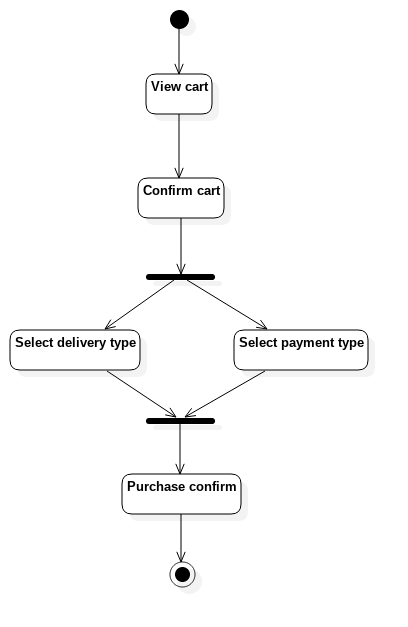
1. Aggiunta al carrello



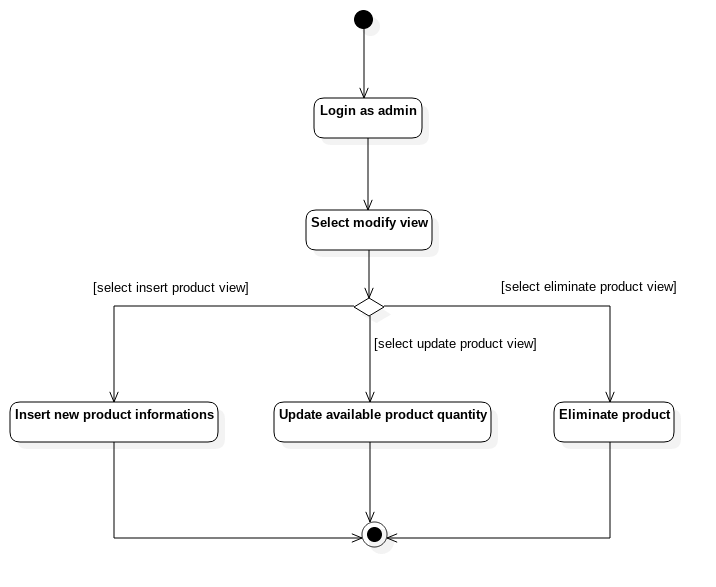
1. Rimozione dal carrello



1. Acquisto del carrello

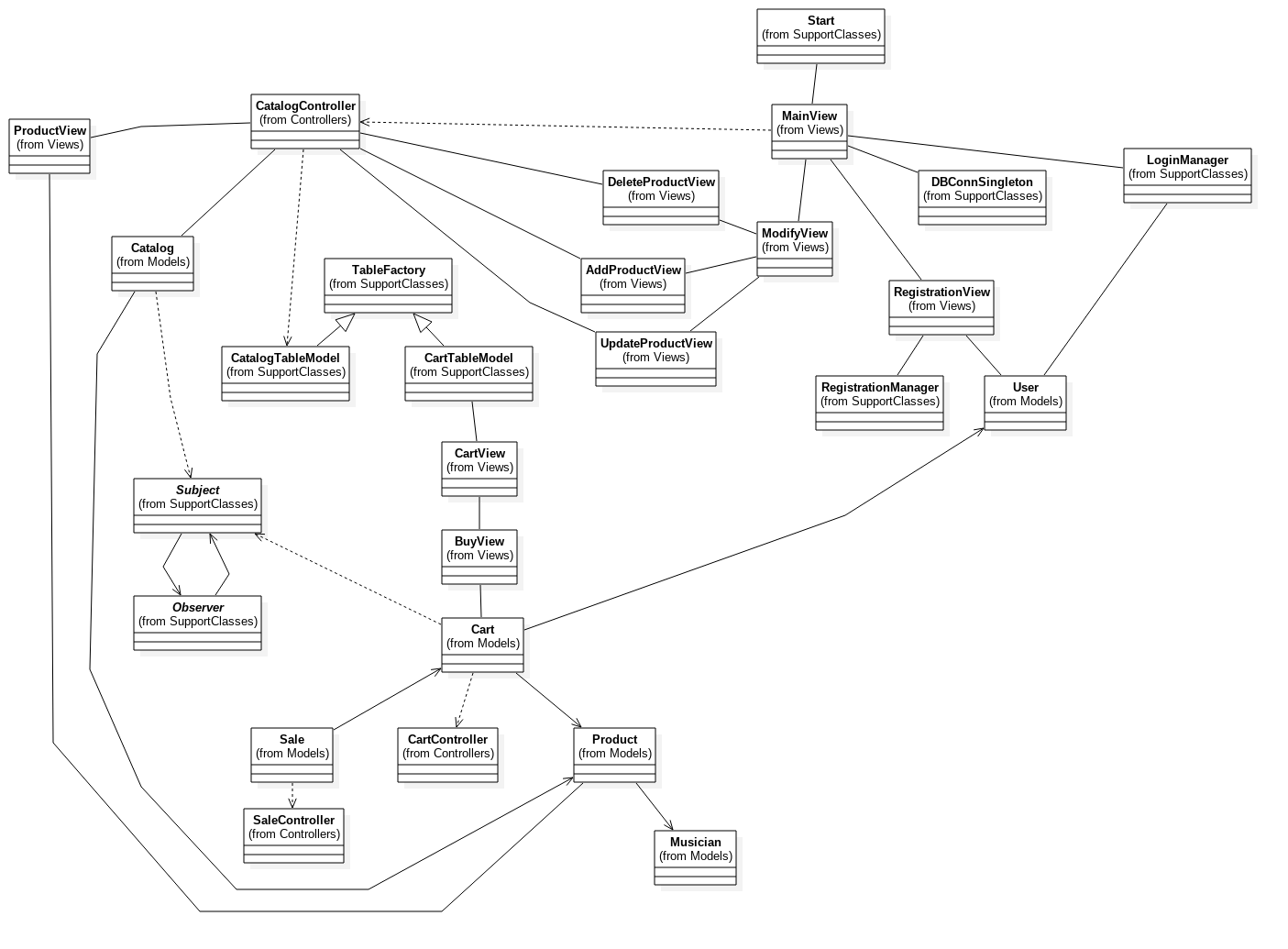


1. Aggiornamento database



3.5 Diagramma delle classi

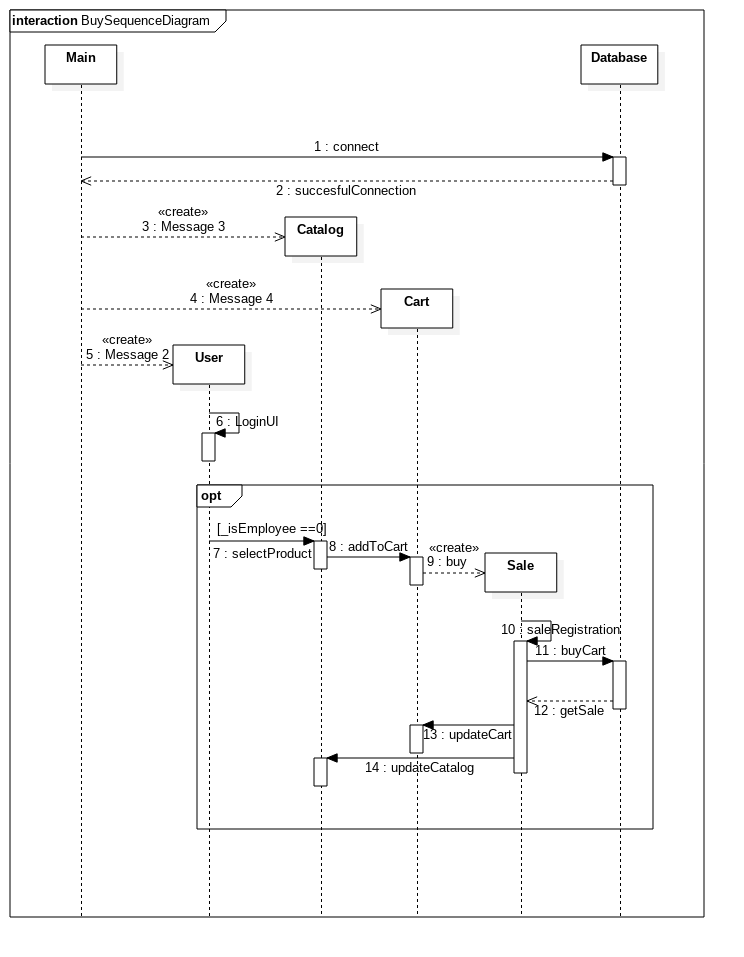
Abbiamo costruito il diagramma delle classi che contiene le classi più importanti, specificando solamente il nome delle classi.



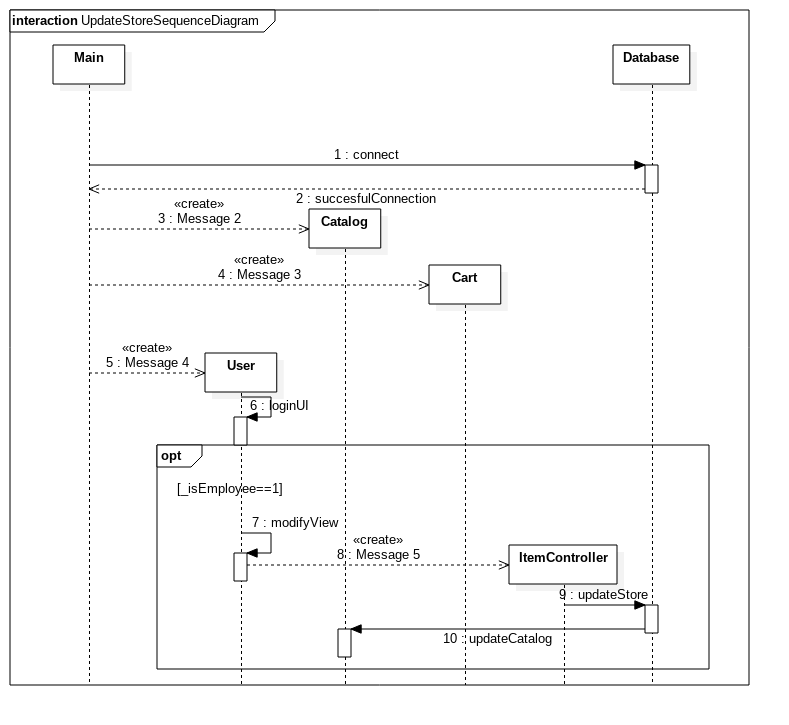
3.6 Diagrammi di sequenza del diagramma delle classi

Abbiamo creato quattro diagrammi di sequenza relativi all’interazione tra front-end (main) e back-end (database), modellandoli sulla base del diagramma delle classi.

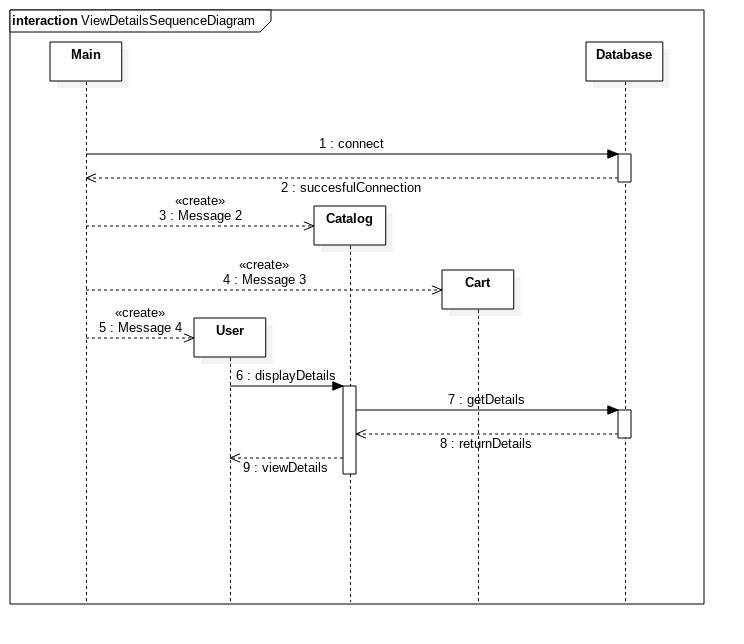
1. Diagramma di sequenza: acquisto prodotto



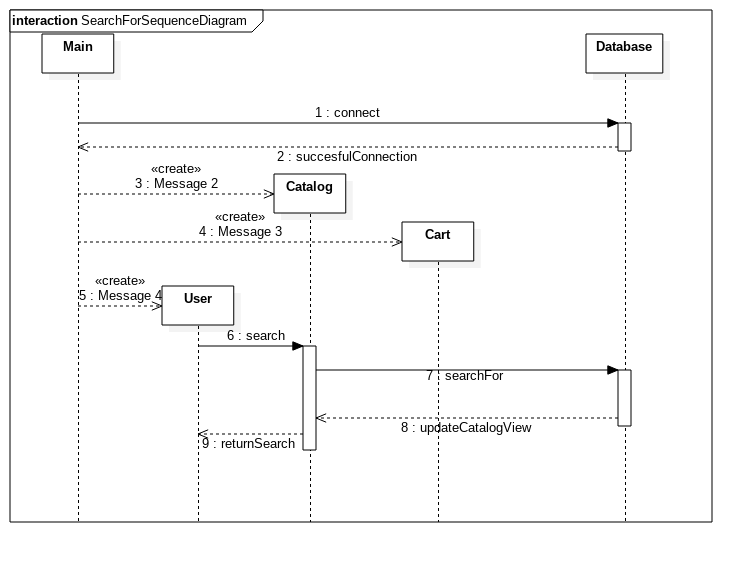
1. Diagramma di sequenza: aggiornamento catalogo



1. Diagramma di sequenza: visualizzazione dettagli prodotti



1. Diagramma di sequenza: ricerca nel catalogo



**4 Scelte di implementazione e assunzioni**

**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 Facotry e pattern comportamentali come Observer e Iterator.

5.1 Pattern MVC

Il pattern MVC è il principale pattern architetturale utilizzato per questo progetto. Esso è composto da una serie di tre elementi: Modelli, Views e Controller.

Il componente centrale del MVC, il modello, cattura il comportamento dell’applicazione in termini di dominio del problema, indipendentemente dall’interfaccia utente. Il modello gestisce direttamente i dati, la logica e le regole dell’applicazione.

Una vista può essere una qualsiasi rappresentazione in output di informazioni, come un grafico o un diagramma. Sono possibili viste multiple delle stesse informazioni, come ad esempio un grafico a barre per la gestione e la vista tabellare per l’amministrazione.

La terza parte, il controller, accetta l’input e lo converte in comandi per il modello e/o vista.

In relazione a questa applicazione, le view si riferiscono a ogni schermata che l’utente può visualizzare: le principali sono quelle relative al catalogo (*MainView*), al carrello (*CartView*) e alla schermata d’acquisto (*BuyView*). I modelli principali sono il carrello (*Cart*), il catalogo (*Catalog*) e il prodotto (Product): questi modelli sono quindi controllati dai relativi controller: *CartController*, *CatalogController* e *SaleController*, che si occupa di gestire la vendita dei prodotti presenti nel carrello in quel momento.

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*.



*Classe DBConnSingleton*

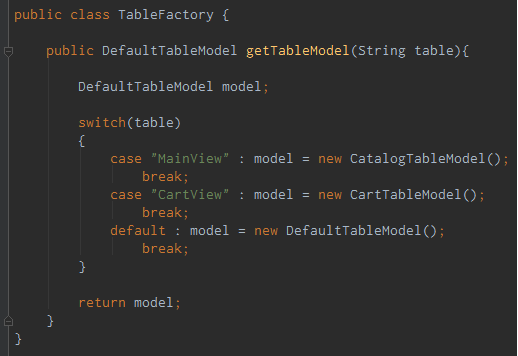
5.3 Pattern Factory

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.



*Classe TableFactory*

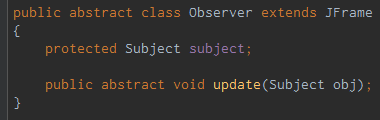
5.4 Pattern Observer

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.

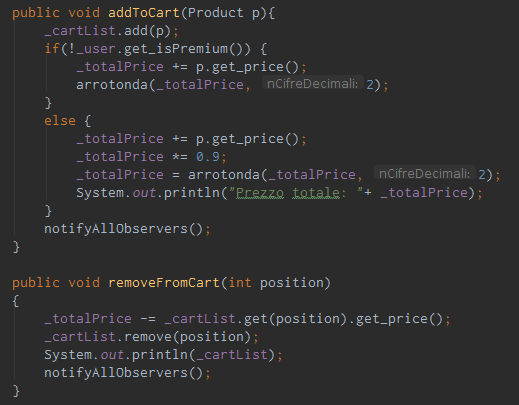
In questo progetto i Subject vengono identificati dagli oggetti *Cart*, *Catalog* e *Product*, che rappresentano rispettivamente un carrello di prodotti pronti ad essere comprati, il catalogo dei prodotti e il modello di un prodotto.

I rispettivi Observer invece sono rappresentati dalle classi *CartView*, *MainView*, e *ProductView*.

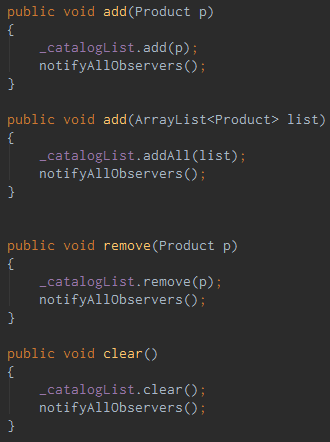
Ogni volta che i Subject si modificano, le tre View (gli Observer) aggiornano i propri dati e li mostrano a video aggiornati.



*Classe Observer*



*Classe Cart*

**

*Classe Catalog*

**

*Classe Product (continua)*

5.5 Pattern Iterator

l **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](http://www.datrevo.com/tag/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.

La libreria standard Java mette a disposizione un’**interfaccia Iterator**, che contiene il metodo *next* che restituisce il prossimo elemento del contenitore e contemporaneamente fa avanzare l’indice di una posizione e il metodo hasNext che restituisce vero se c’è un almeno un altro elemento del contenitore che deve ancora essere visitato.

**6 Testing**