UNIVERSITA' DEGLI STUDI DI VERONA
DIPARTIMENTO DI INFORMATICA

INGEGNERIA DEL SOFTWARE

Documentazione di progetto

Anno Accademico 2016/2017

Candidati:

Donatelli Nicola – VR389842

Andreoni Matteo – VR389703

Soso Simone – VR389836

Sommario

1 Introduzione 3
1.1 Specifiche 3
2 Scelte di implementazione 5
3 Assunzioni generali 6
4 Diagrammi 7
4.1 Use-Case Diagrams 7
4.2 Sequence Diagrams 11
4.2.1 Sequence Diagrams per gli Use Case 11
4.2.2 Sequence Diagrams specifici 15
4.3 Activity Diagrams 19
4.4 Class Diagram 22
5 Testing 23
6 Pattern utilizzati 24
6.1 Pattern MVC 24
6.2 Pattern Singleton 25
6.3 Pattern Factory 26
6.4 Pattern Observer 27

1 Introduzione

La seguente relazione descrive lo sviluppo ed il funzionamento del progetto d'esame di Ingegneria del Software "MusicStore", un sistema informativo per gestire un negozio online di CD e DVD.

1.1 Specifiche

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.

2 Scelte di implementazione

Per quanto riguarda gli schemi e la progettazione, abbiamo deciso di utilizzare il programma StarUML, che ci ha permesso di disegnare tutti i diagrammi richiesti dalle specifiche.

Successivamente abbiamo utilizzato il linguaggio Java per la prototipazione del sistema, insieme alle librerie AWT e Swing per creare l'interfaccia grafica. La parte di backend è stata affidata ad un database relazionale basato su PostgreSQL.

L'interfaccia grafica permette di visualizzare i dettagli di un prodotto semplicemente cliccando sopra la riga desiderata. Il login è stato implementato pensando a un form unico sia per clienti, che per impiegati: a seconda di chi effettuerà l'accesso, il sistema restituirà una visualizzazione differente.

E' stato scelto di implementare una visualizzazione grafica semplice, senza effetti elaborati: questo per rendere l'interfaccia intuitiva e non complessa da realizzare.

Le immagini usate nel programma risiedono in una cartella in locale: sono presenti le copertine degli album, alcune icone, e i diversi pulsanti, che vanno a sostituire i pulsanti standard della libreria Swing di Java.

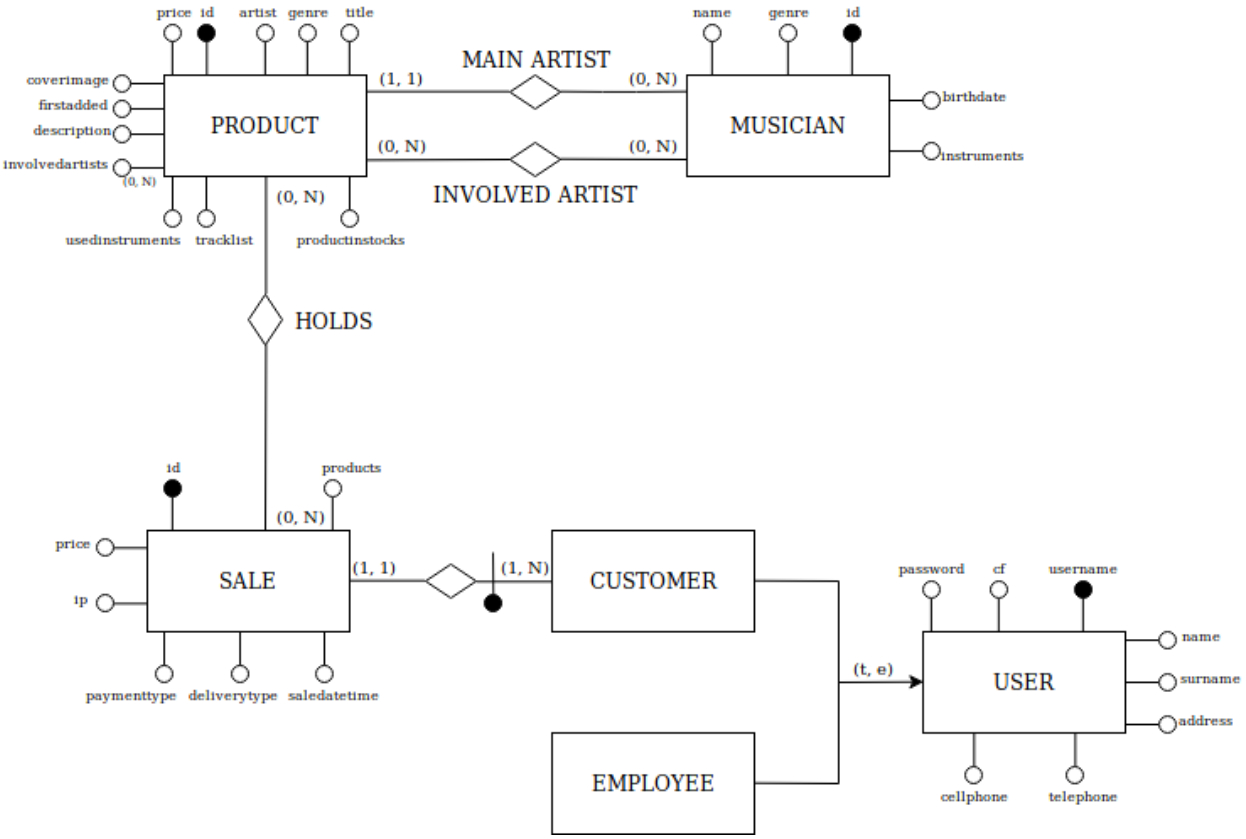
Abbiamo scelto di non implementare suggerimenti basati sugli acquisti passati effettuati dai diversi utenti.

L'inserimento di un nuovo prodotto nel database da parte dell'impiegato non tiene conto di alcune informazioni, come la scaletta dei brani, musicisti e strumenti coinvolti, in quanto non vengono visualizzate nel dettaglio del prodotto, per facilitare lo sviluppo.

La view riguardante i suggerimenti per gli utenti che hanno effettuato l'accesso è stata sviluppata non tenendo conto dei precedenti acquisti, ma si basa su un banner casuale che mostra di volta in volta i diversi prodotti del database divisi in generi musicali diversi.

Il database risiede anch'esso in locale, questo per facilitare lo sviluppo dell'applicazione, ma soprattutto il testing: non sono presenti così variabili esterne di possibile disturbo quali connessioni di rete, sistemi di autenticazione esterni, ecc.

Il seguente schema ER mostra come è rappresentato il database seguendo le direttive presenti nelle specifiche: abbiamo preferito apportare alcune modifiche però nell'implementazione del sistema, le quali ci hanno permesso uno sviluppo più agevole di particolari funzioni richieste. In particolare, le due classi *Customer* e *Employee* sono state sostituite in favore di un'unica classe *User*, che può contare sull'appoggio di due attributi in più di tipo booleano: *isEmployee*, che indica se l'account è di tipo cliente o di tipo impiegato, e *isPremium*, che indica la presenza o meno di un account premium.



3 Assunzioni generali

Durante lo svolgimento del progetto, abbiamo dovuto fare delle assunzioni, in quanto in alcuni punti i requisiti lasciavano libertà di implementazione più ampie.

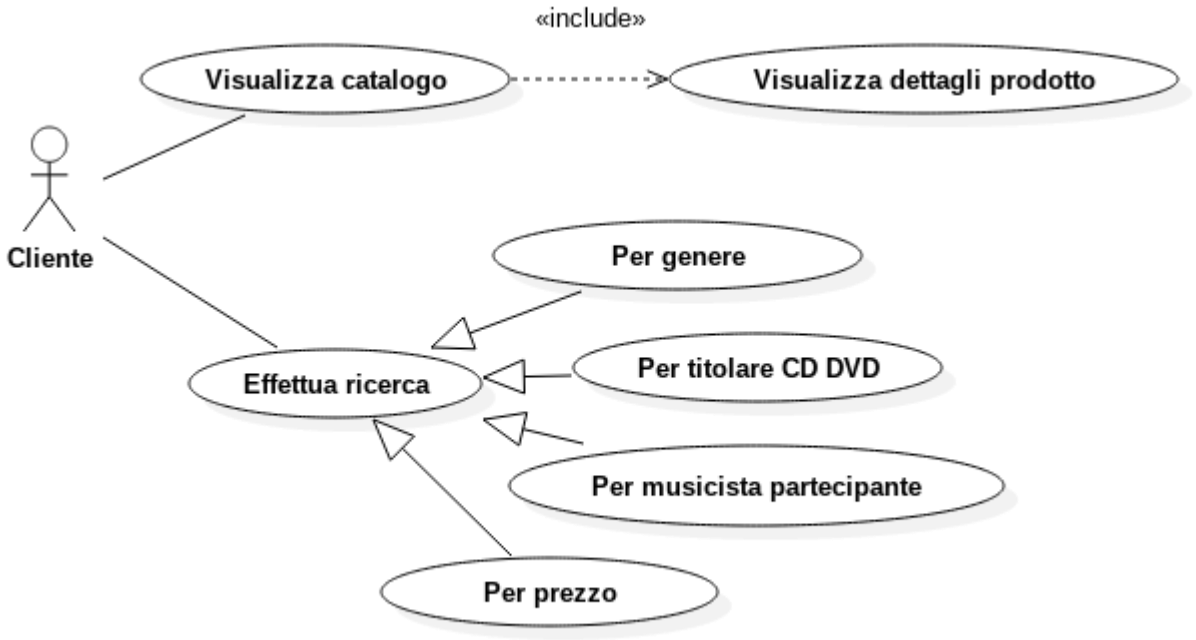
In particolare:

- Gli impiegati possono accedere al sistema con l'account di amministratore, già presente nel database. Essi non possono pertanto registrare nuovi account di tipo *Employee*.
- I clienti non possono accedere in alcuno modo alle funzioni riservate agli impiegati del sistema, in quanto bisogna inserire le credenziali d'accesso dell'account di amministratore.
- Gli impiegati non possono comportarsi da clienti quando hanno effettuato l'accesso come amministratori: di conseguenza non possono comprare prodotti e creare carrelli virtuali.

4 Diagrammi

4.1 Use-Case Diagrams

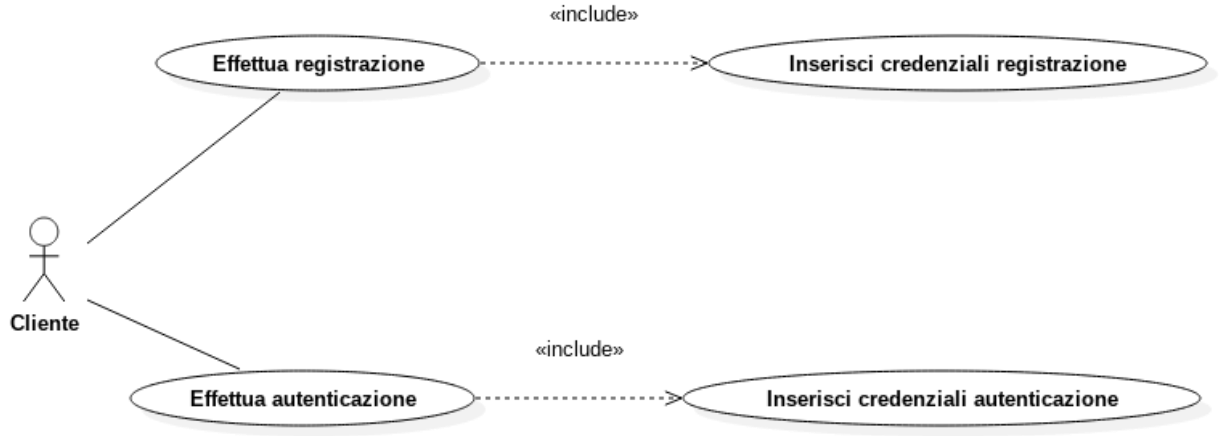
1) Caso d'uso n°1



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: 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

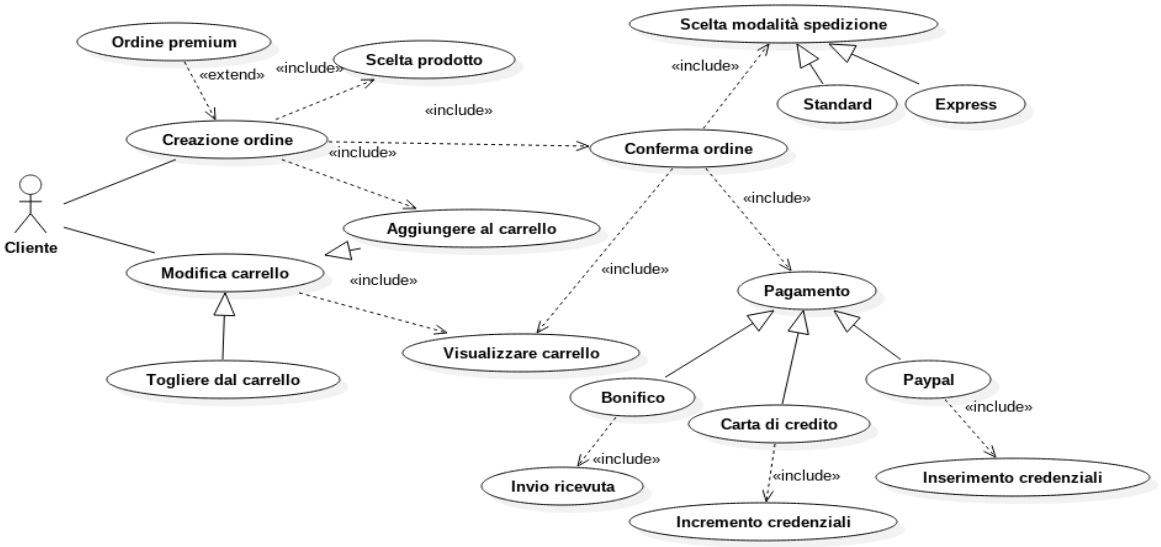
2) Caso d'uso n°2



Scheda di specifica:

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: a. Inserisce i propri dati personali richiesti dal sistema e conferma la registrazione 3) Se il Cliente vuole autenticarsi: a. Il Cliente inserisce username e password e conferma l'inserimento
Postcondizioni: Il Cliente risulta registrato, oppure autenticato al negozio

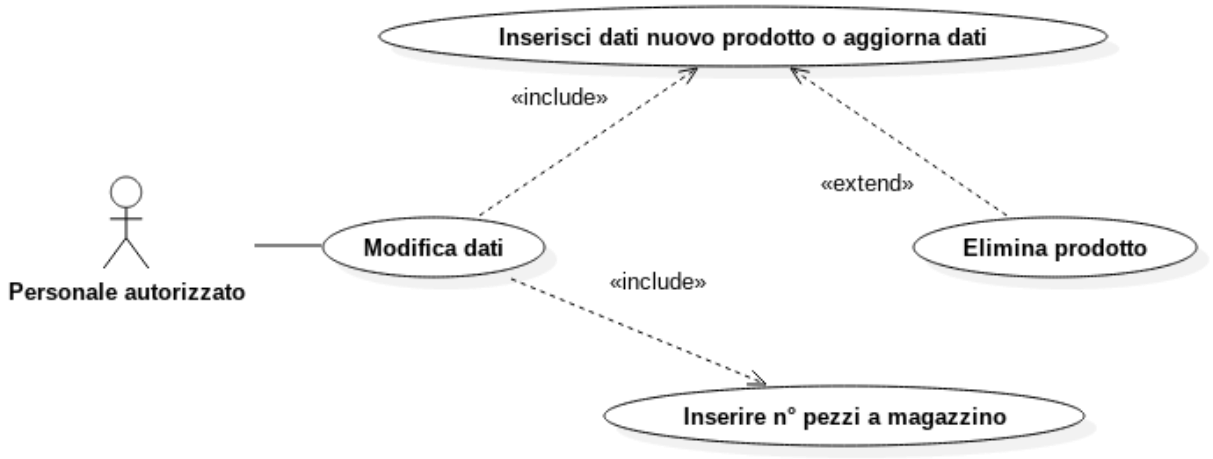
3) Caso d'uso n°3



Scheda di specifica:

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 c. Visualizza il carrello e conferma l'acquisto 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) Caso d'uso n°4



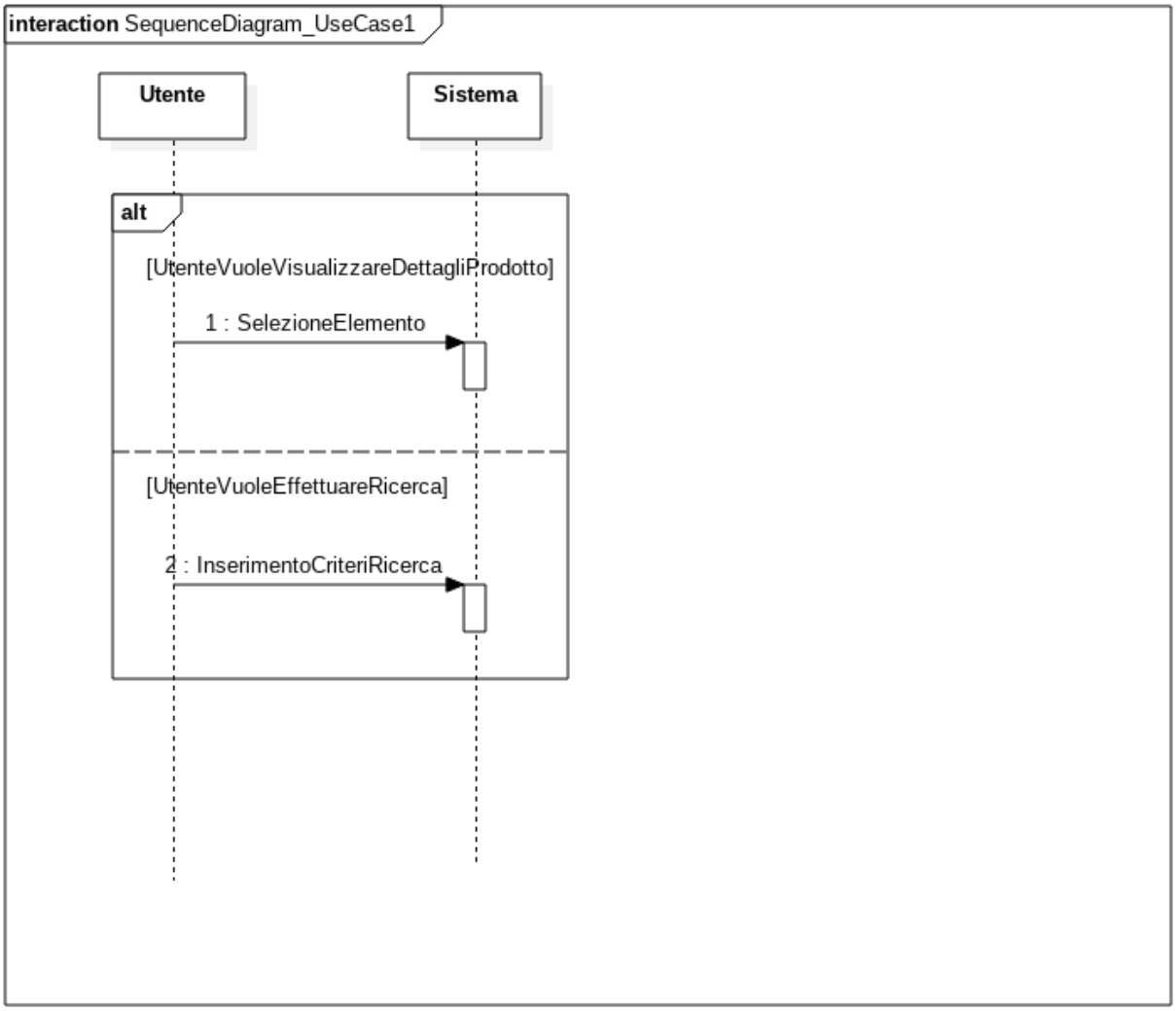
Scheda di specifica:

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 dati di CD/DVD c. Eliminare prodotti non più in vendita d. Aggiornare le disponibilità dei CD/DVD desiderati, in base alla loro presenza effettiva in magazzino
Postcondizioni: L'Impiegato ha modificato il catalogo di sistema

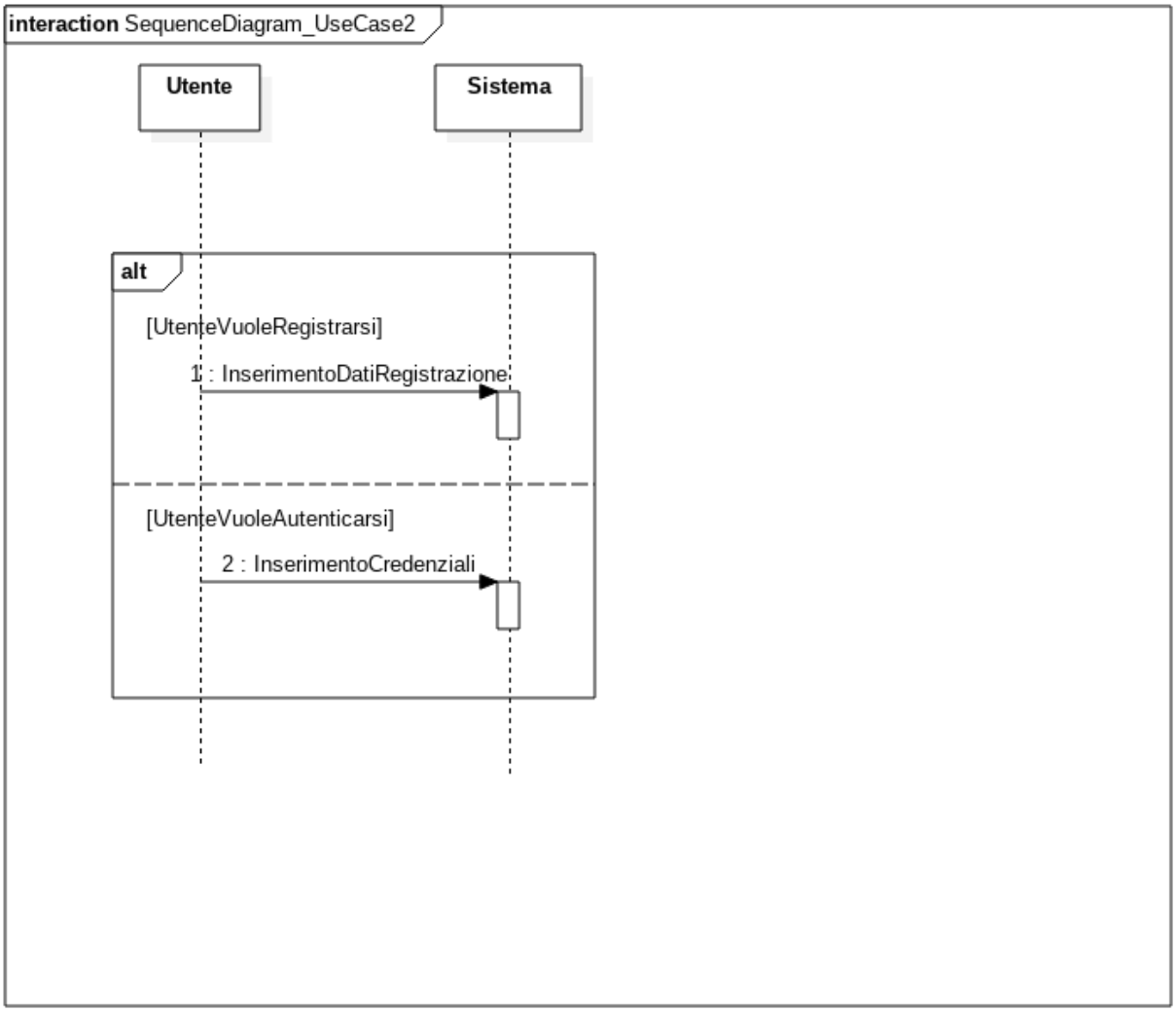
4.2 Sequence Diagrams

4.2.1 Sequence Diagrams per gli Use Case

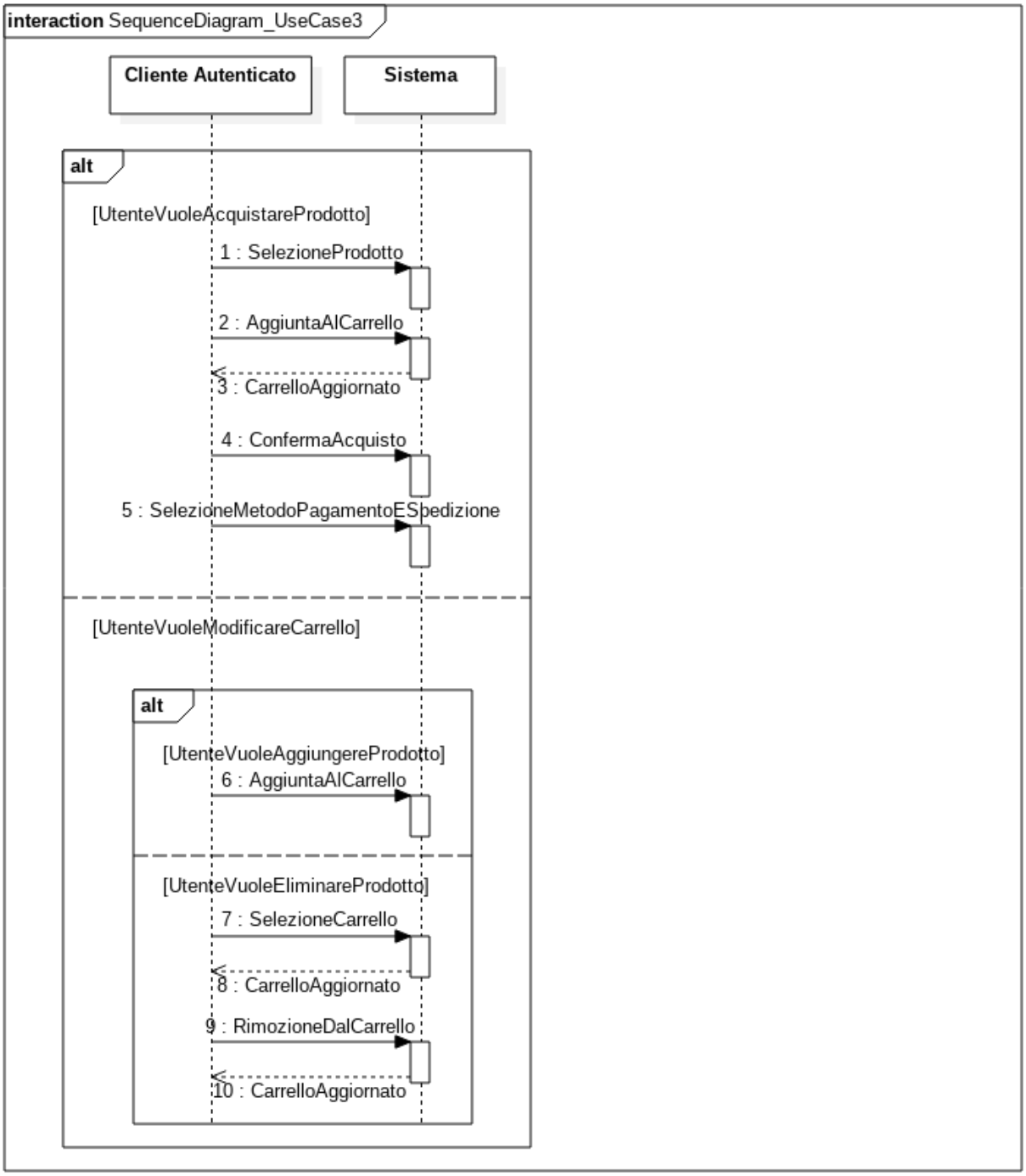
1) SeqDiagramUC1



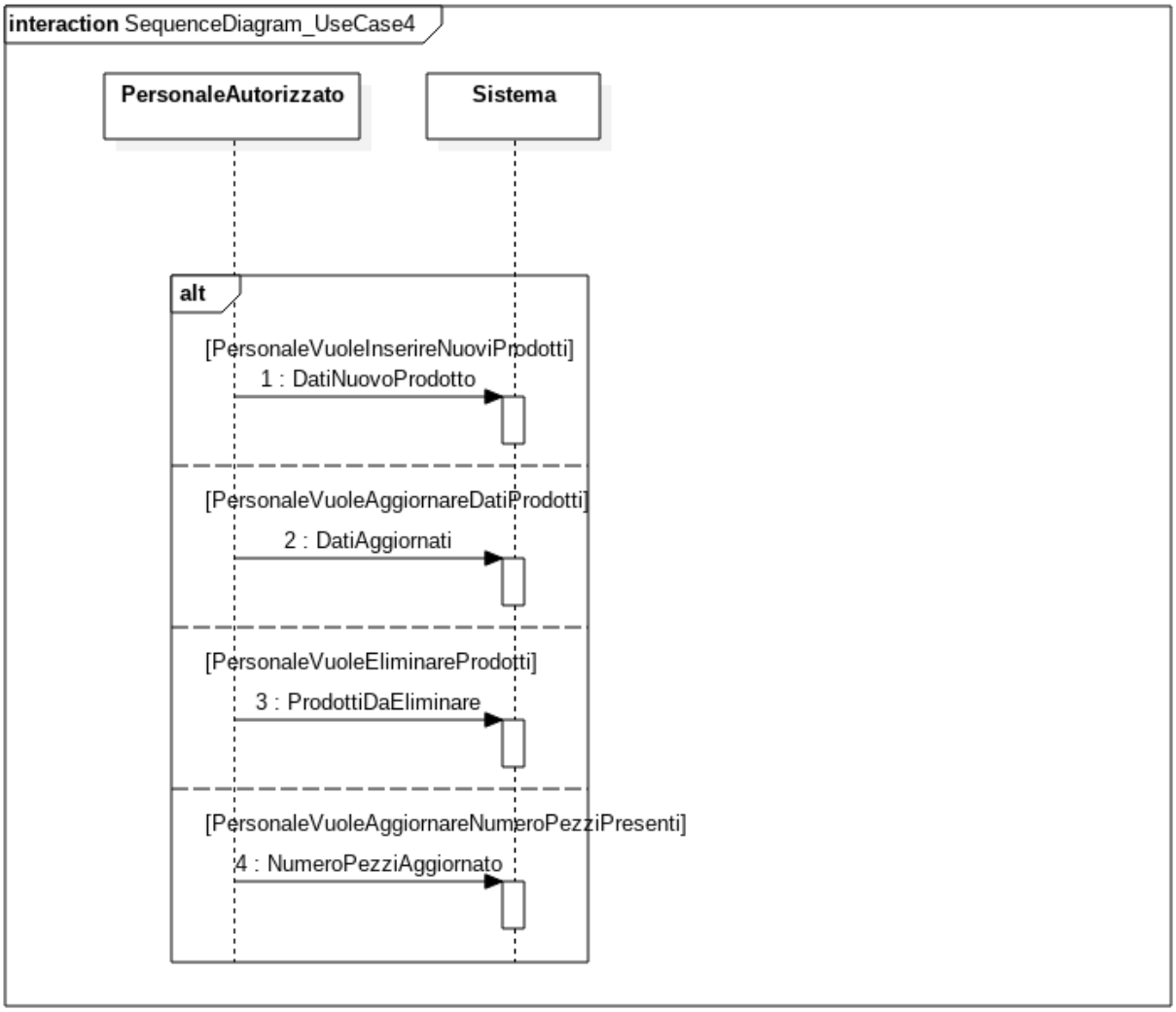
2) SeqDiagramUC2



3) SeqDiagramUC3

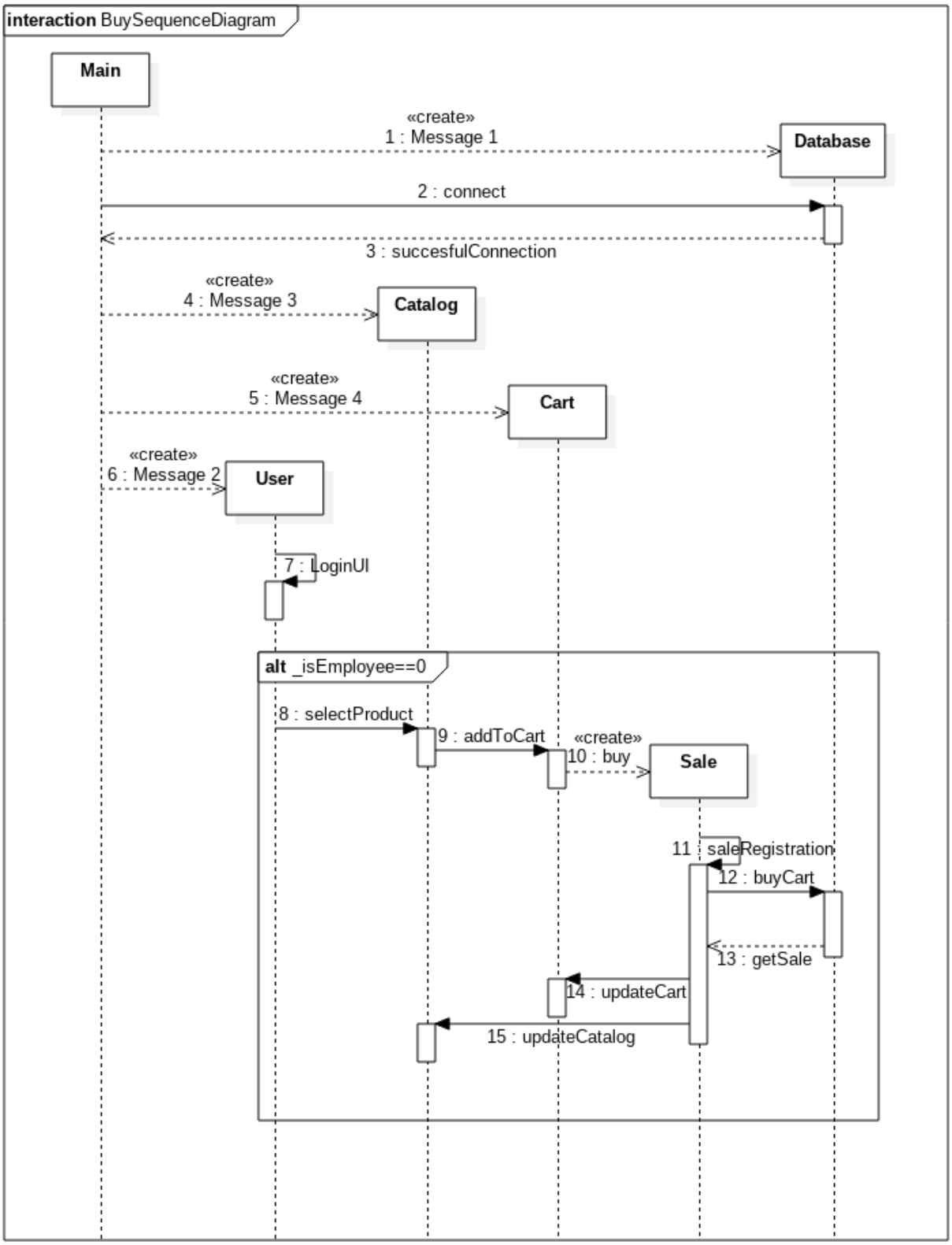


4) SeqDiagramUC4

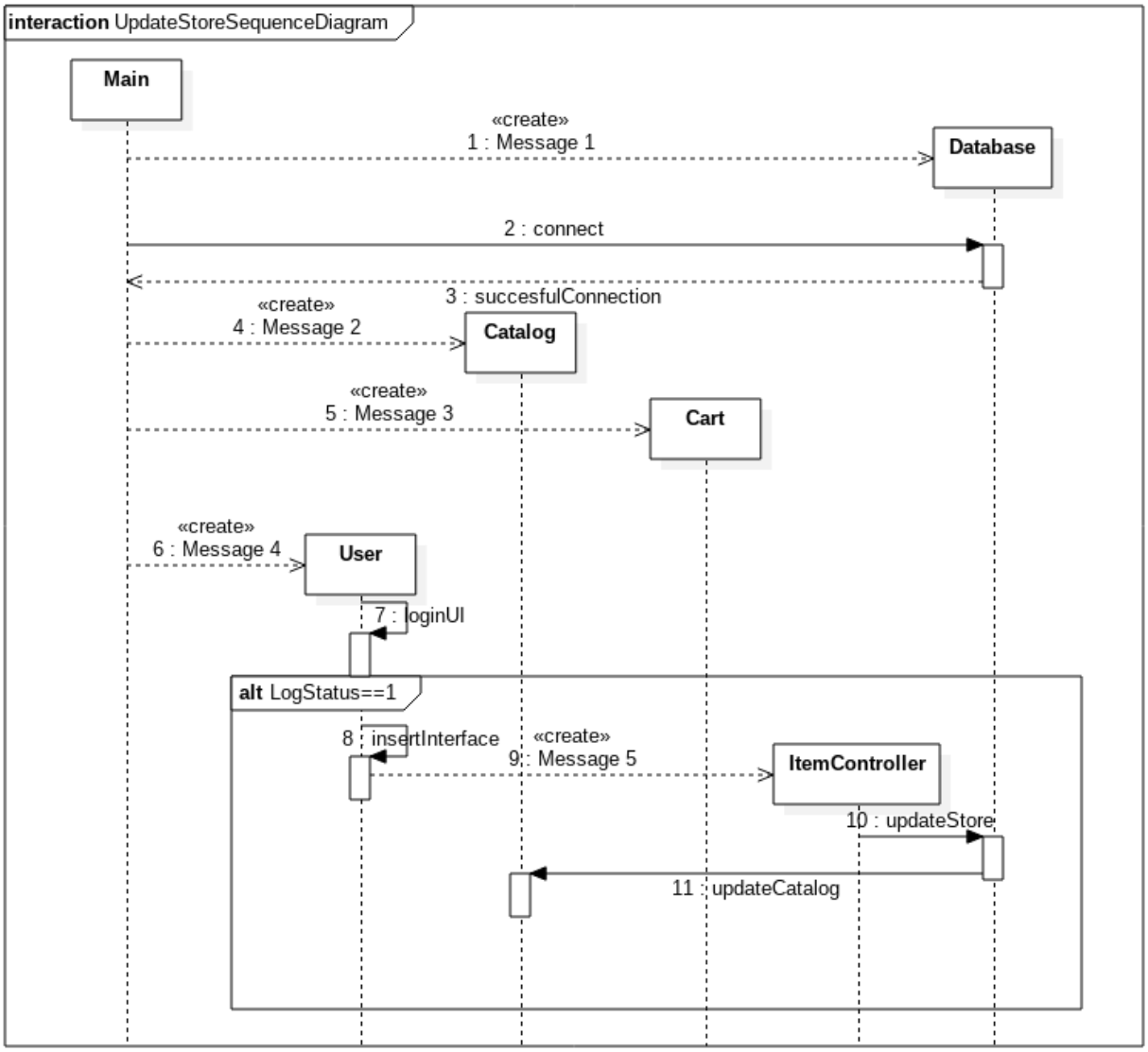


4.2.2 Sequence Diagrams specifici

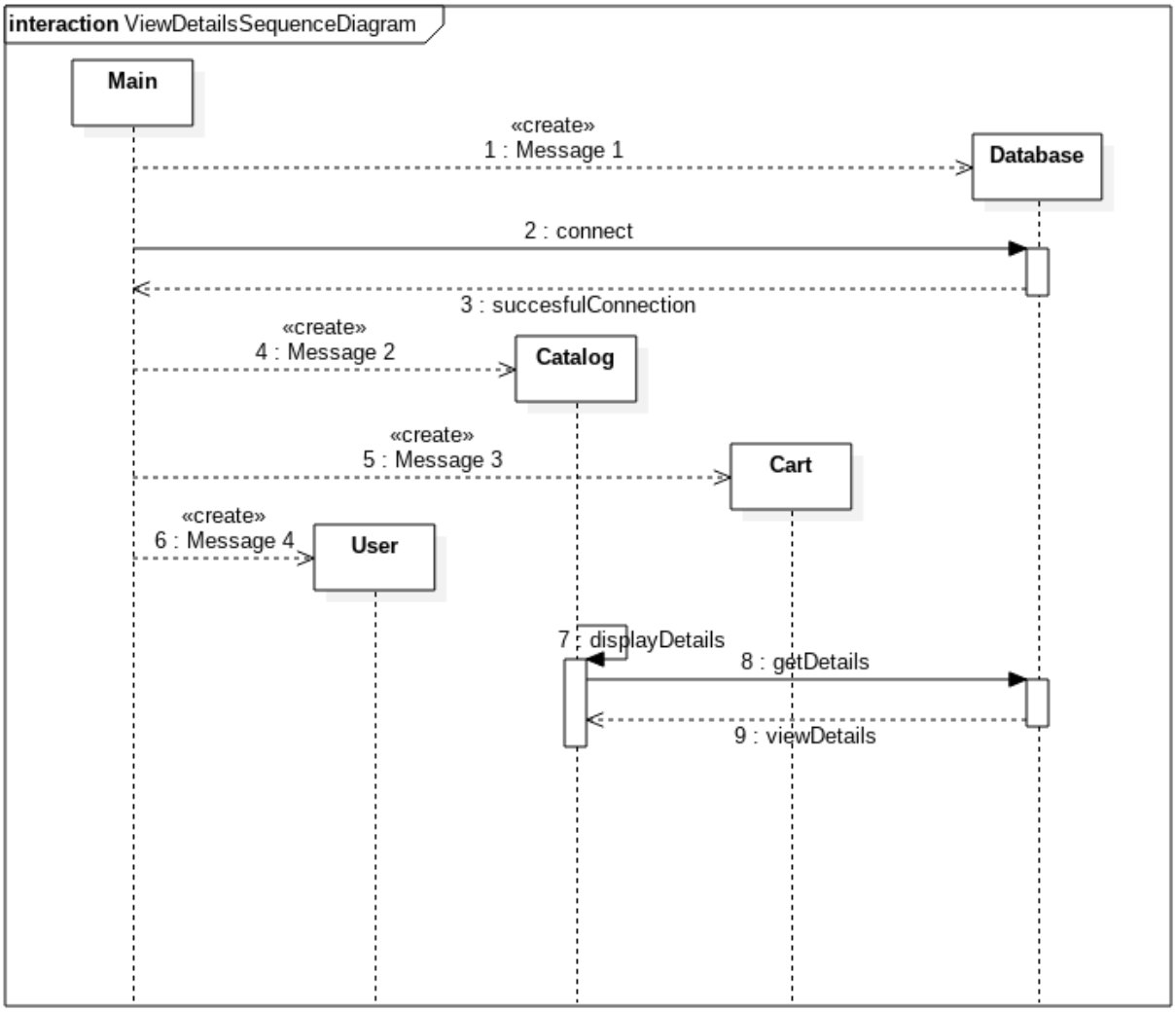
1) Buy Sequence Diagram



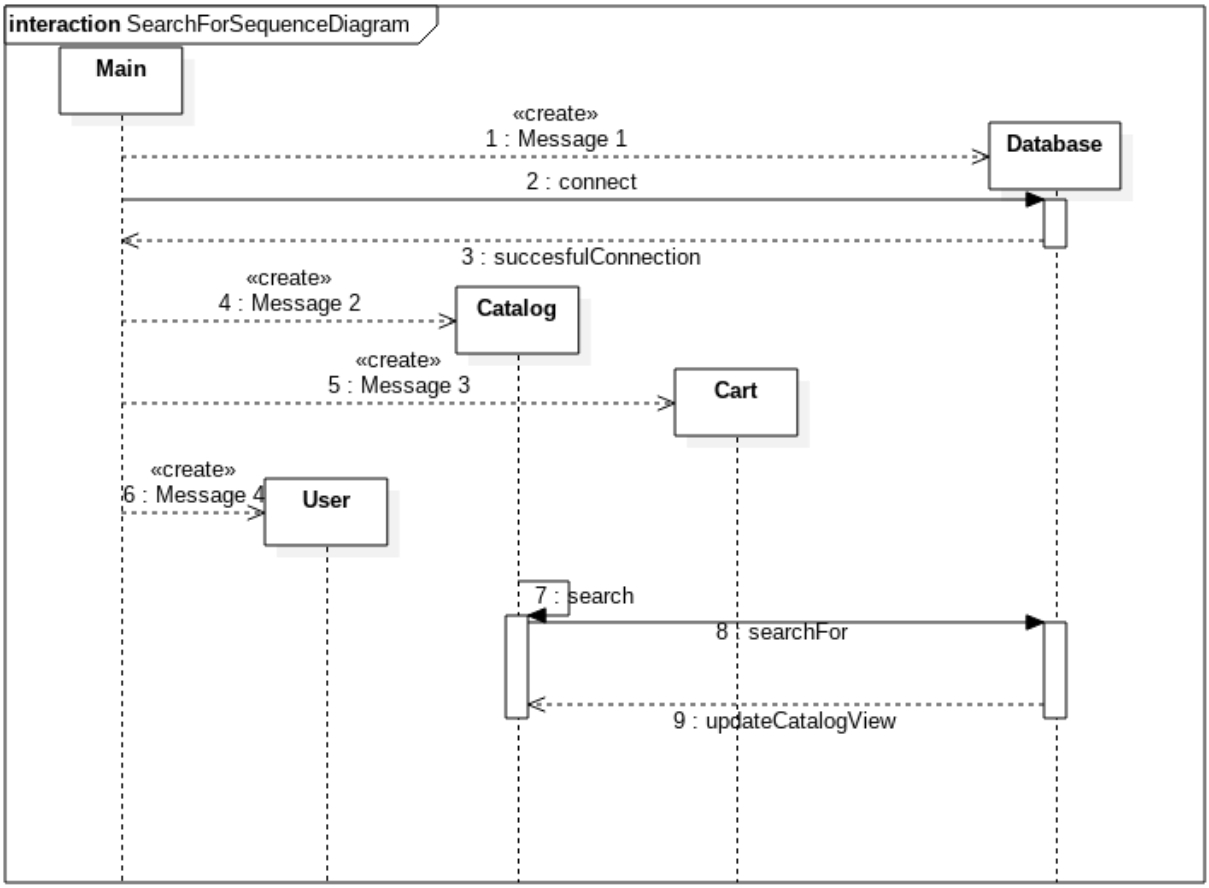
2) Update Store Sequence Diagram



3) View Details Sequence Diagram

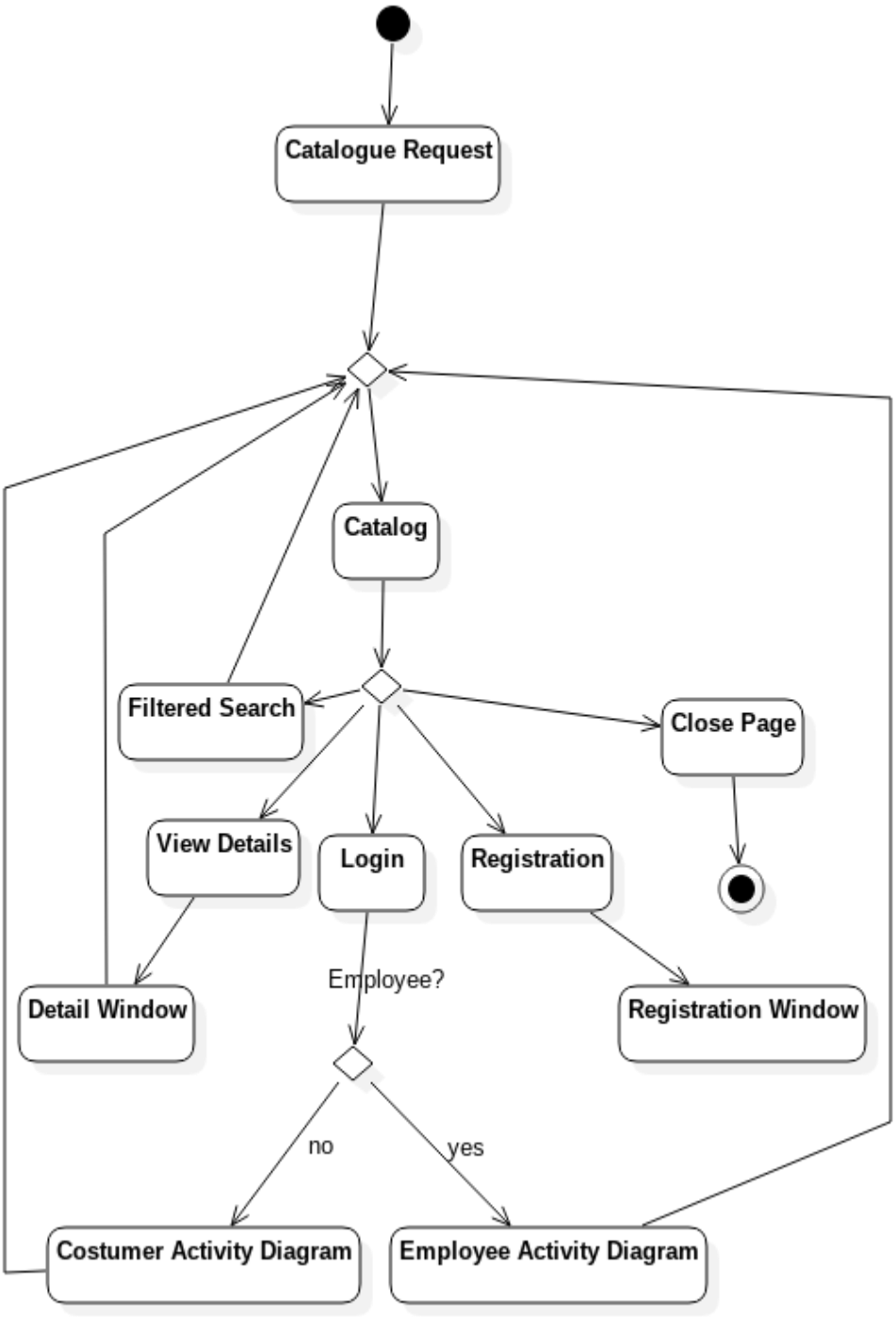


4) Search For Sequence Diagram

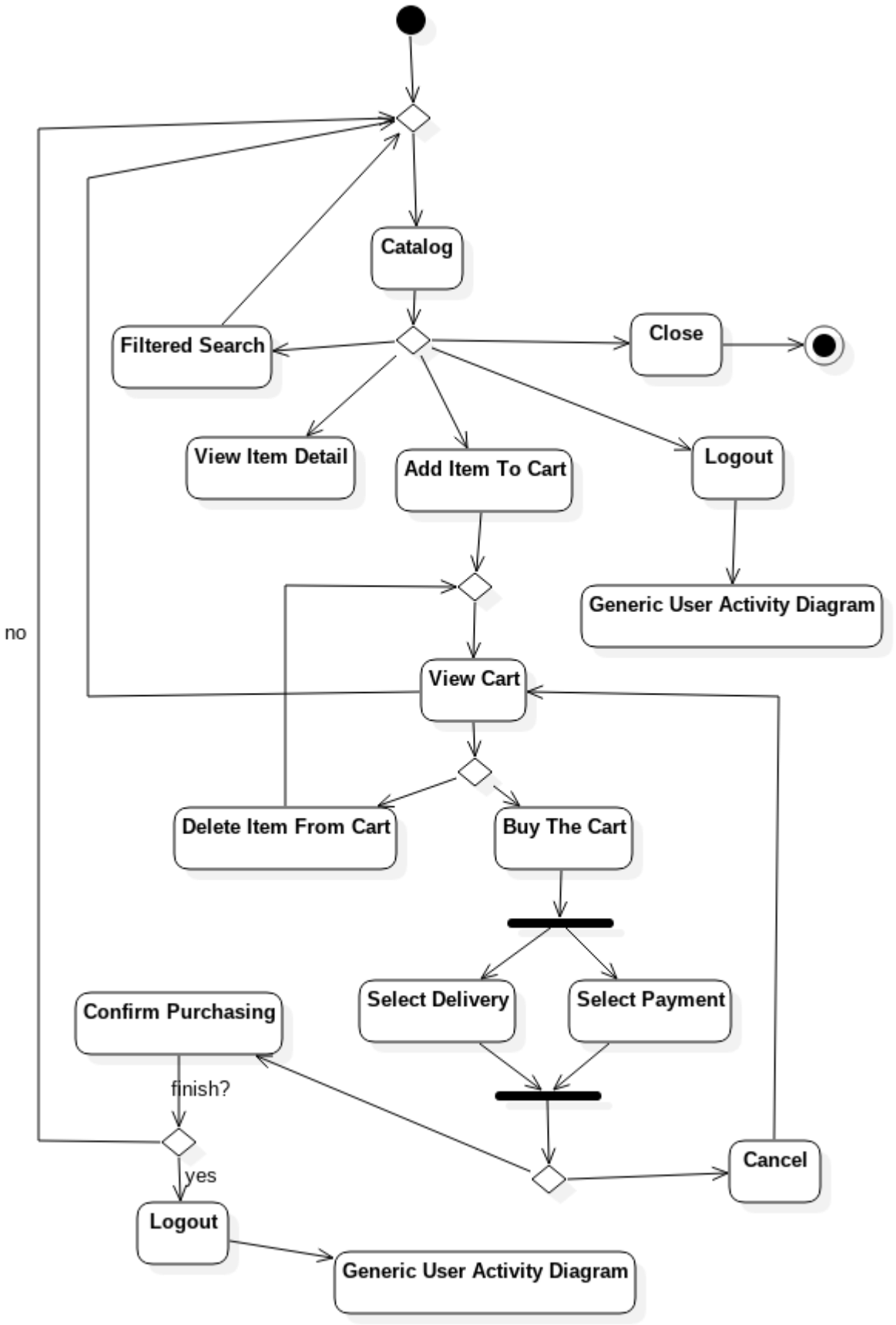


4.3 Activity Diagrams

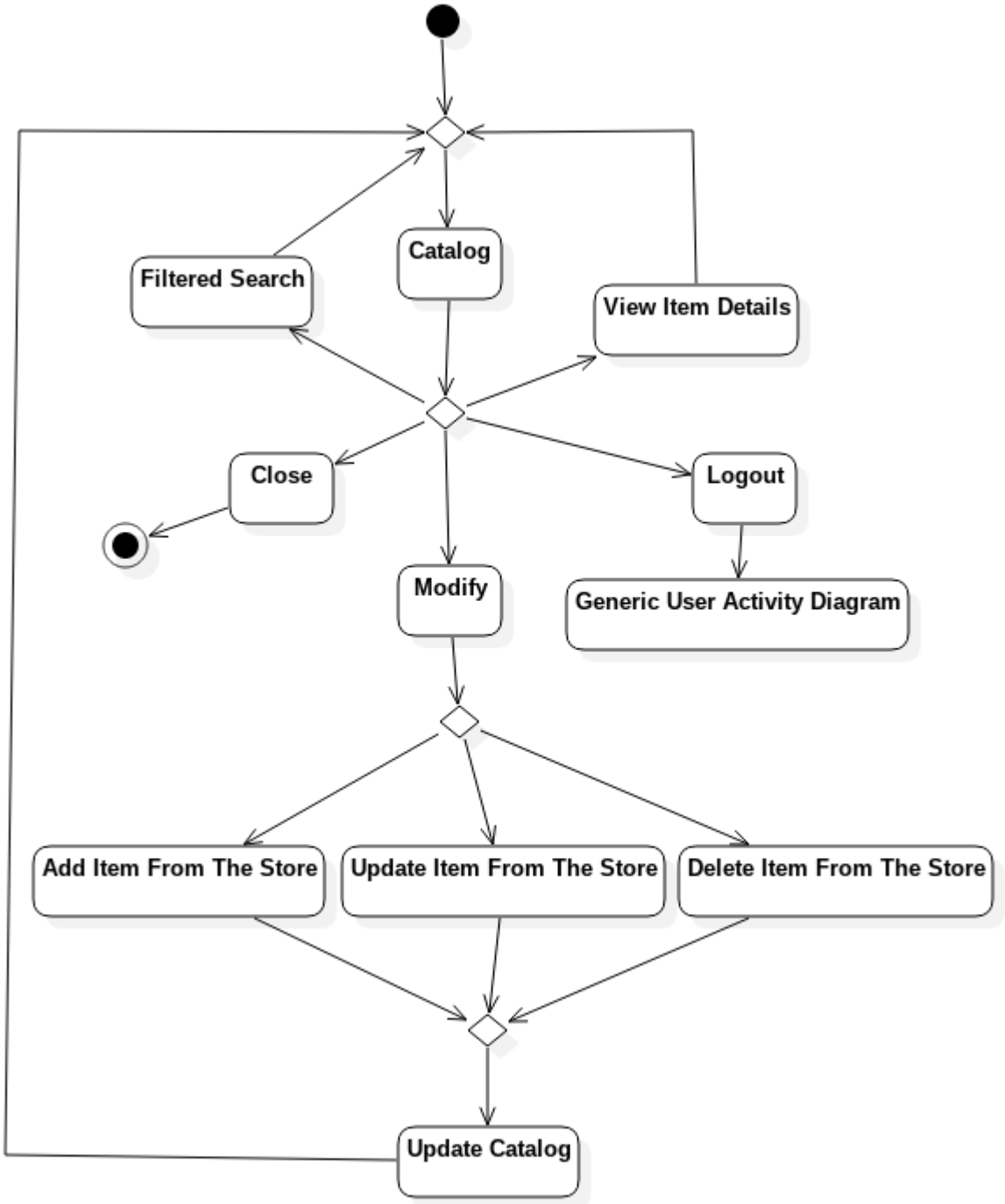
1) Generic User Activity Diagram



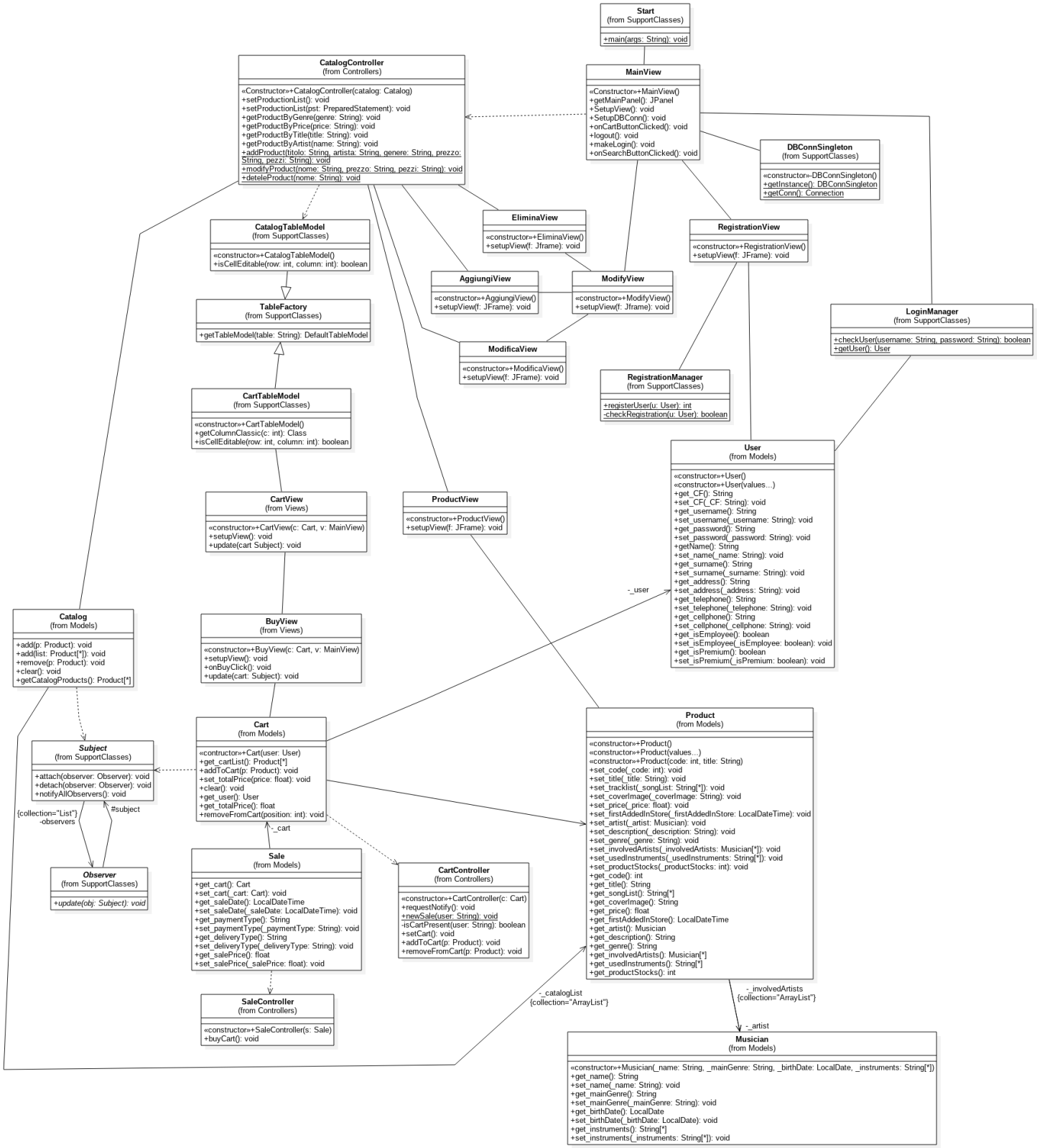
2) Customer Activity Diagram



3) Employee Activity Diagram



4.4 Class Diagram



5 Testing

Data la chiarezza dei requisiti iniziali, è stato adottato un processo di sviluppo di tipo agile: questo ha permesso di focalizzarsi sul codice da scrivere e sulle funzioni da implementare di volta in volta. Un processo agile ha anche permesso di testare subito le nuove funzioni, senza dover aspettare la fine dell'implementazione: è stato possibile quindi trovare e correggere molti bug ed errori dovuti a una scrittura non corretta di codice e metodi.

Le attività di testing finale sono poi servite a provare la completezza e la correttezza del sistema nella sua versione ultima.

Alcune di queste attività:

- Registrare diversi utenti controllando eventuali conflitti tra loro (casi di registrazioni dello stesso username più volte).
- Autenticazione e logout con i due diversi tipi di utente, per verificare la correttezza delle visualizzazioni differenti.
- Autenticazione con dati non presenti nel database, per controllare la correttezza dell'operazione di login.
- Effettuare diversi tipi di ricerca, per testare la correttezza delle query di ricerca nel database e la corretta visualizzazione delle view.
- Aggiunta ed eliminazione di prodotti nel carrello, per la verifica della corretta visualizzazione della view del carrello.
- Acquisto di prodotti per verifica delle query di update del database.
- Trasformazione di un account normale in account premium, per verificare la corretta applicazione di sconti e spese di spedizione gratuite.
- Modifica, aggiunta o eliminazione di prodotti nel database, per testare la correttezza delle query di aggiornamento del database.

6 Pattern utilizzati

6.1 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: i, *CatalogController* e *SaleController*, che si occupa di gestire la vendita dei prodotti presenti nel carrello in quel momento.

6.2 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*.

```
public class DBConnSingleton {

    private static DBConnSingleton _instance = null;
    public static Connection _conn;

    private DBConnSingleton(){
        try
        {
            _conn = DriverManager.getConnection( url: "jdbc:postgresql://localhost:5432/store",
                    user: "postgres", password: "a");
        }
        catch (SQLException e)
        {
            e.printStackTrace();
        }
    }

    public static DBConnSingleton getInstance(){

        if(_instance==null)
            _instance = new DBConnSingleton();

        return _instance;
    }

    public static Connection getConn() { return _conn; }

}
```

Classe DBConnSingleton

6.3 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.

```
public class TableFactory {

    public DefaultTableModel getTableModel(String table){

        DefaultTableModel model;

        switch(table)
        {
            case "MainView" : model = new CatalogTableModel();
                break;
            case "CartView" : model = new CartTableModel();
                break;
            default : model = new DefaultTableModel();
                break;
        }

        return model;
    }
}
```

Classe TableFactory

6.4 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.

```
public abstract class Observer extends JFrame
{
    protected Subject subject;

    public abstract void update(Subject obj);
}
```

Classe Observer

```
public void addToCart(Product p){
    _cartList.add(p);
    if(!_user.get_isPremium()) {
        _totalPrice += p.get_price();
        arrotonda(_totalPrice, nCifreDecimali: 2);
    }
    else {
        _totalPrice += p.get_price();
        _totalPrice *= 0.9;
        _totalPrice = arrotonda(_totalPrice, nCifreDecimali: 2);
        System.out.println("Prezzo totale: "+ _totalPrice);
    }
    notifyAllObservers();
}

public void removeFromCart(int position)
{
    _totalPrice -= _cartList.get(position).get_price();
    _cartList.remove(position);
    System.out.println(_cartList);
    notifyAllObservers();
}
```

Classe Cart

```java
public void add(Product p)
{
    _catalogList.add(p);
    notifyAllObservers();
}

public void add(ArrayList<Product> list)
{
    _catalogList.addAll(list);
    notifyAllObservers();
}


public void remove(Product p)
{
    _catalogList.remove(p);
    notifyAllObservers();
}

public void clear()
{
    _catalogList.clear();
    notifyAllObservers();
}
```

Classe Catalog

```java
public void set_code(int _code) {
    this._code = _code;
    notifyAllObservers();
}

public void set_title(String _title) {
    this._title = _title;
    notifyAllObservers();
}

public void set_trackList(ArrayList<String> _songList) {
    this._trackList = _songList;
    notifyAllObservers();
}

public void set_coverImage(String _coverImage) {
    this._coverImage = _coverImage;
    notifyAllObservers();
}

public void set_price(float _price) {
    this._price = _price;
    notifyAllObservers();
}

public void set_firstAddedInStore(LocalDateTime _firstAddedInStore) {
    this._firstAddedInStore = _firstAddedInStore;
    notifyAllObservers();
}
```

Classe Product (continua)