Documentazione progetto Ingegneria del Software

Elisa Zanella e Lorenzo Mioso

July 14, 2020

Contents

1 Requisiti ed interazioni utente-sistema

1.1 Specifiche casi d'uso

Il sistema è diviso in due parti principali. Una dedicata all'utente e l'altra al responsabile del reparto. Per **utente** si intende la persona che utilizza questo servizio per fare la spesa e per **responsabile reparto** l'impiegato che gestisce il reparto a lui assegnato. L'utente non necessita di autenticarsi per consultare il catalogo dei prodotti, ma per acquistarli deve registrarsi nel sistema, invece il responsabile deve autenticarsi con delle credenziali pre-fornite dagli amministratori del sistema.

1.1.1 Casi d'uso relativi all'utente

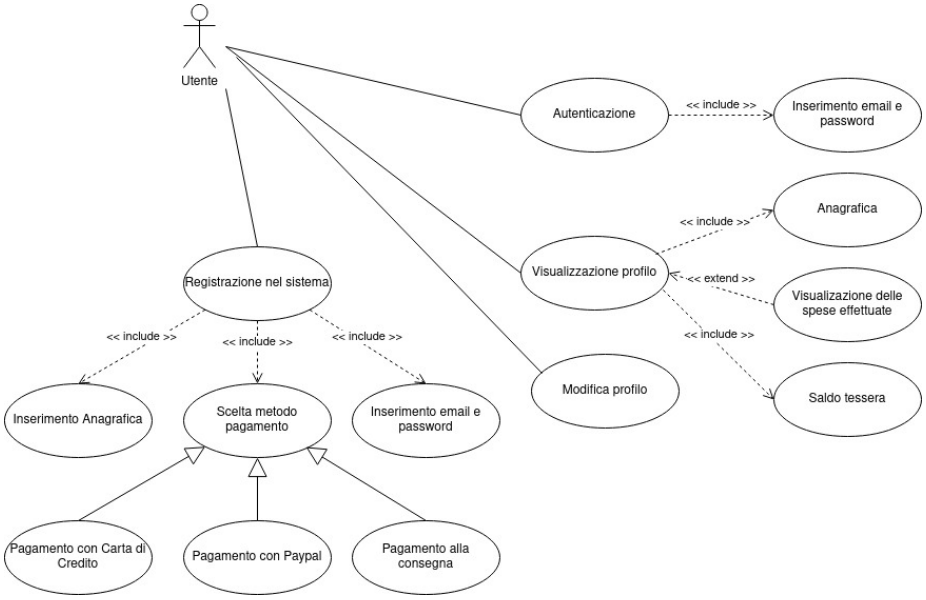

Figure 1: Use case di utente per la gestione del profilo e la registrazione

Registrazione nel sistema:

Attori :
Utente
Descrizione :
Procedura di inserimento dei dati dell'utente per registrarsi nel sistema
Sequenza di eventi :
L'utente inserisce i suoi dati relativi all'anagrafica, scelta di metodo di pagamento preferito, email e password.
Post condizioni:
Il sistema registra l'utente con i dati inseriti
Sequenza Alternativa :
Se i dati inseriti non sono validi il sistema non effettua la registrazione
Se l'email è già stata inserita nel sistema, il sistema ne richiede una diversa

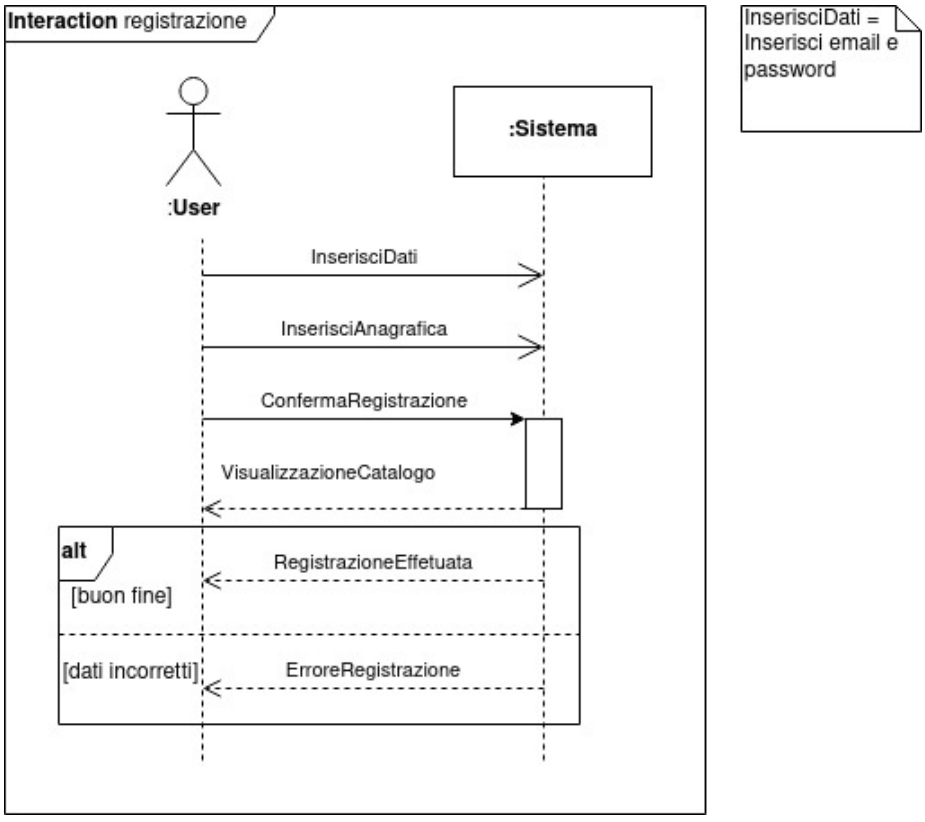

Figure 2: Sequence Diagram della registrazione

Autenticazione:

Attori :
Utente
Descrizione :
Procedura di autenticazione dell'utente
Sequenza di eventi :
L'utente inserisce i dati necessari per l'autenticazione
Pre-condizioni:
L'utente deve essere registrato nel sistema
Sequenza Alternativa :
Se i dati inseriti non sono validi il sistema non effettua l'autenticazione

Visualizzazione catalogo:

Attori :
Utente
Descrizione :
Il sistema permette di visualizzare tutti i prodotti disponibili, e ordinarli:

1. in modo crescente e decrescente per prezzo
2. in ordine alfabetico per marca (dalla A alla Z e dalla Z alla A)

Sequenza di eventi :
L'utente accede all'area dedicata

Gestione profilo:

Attori:
Utente
Scopo e Descrizione sintetica:
Il sistema permette all'utente di gestire il proprio profilo. L'utente può visualizzare e modificare il proprio profilo.
Sequenza di eventi :
Questo caso d'uso viene attivato quando l'utente vuole modificare o visualizzare il proprio profilo.

1. L'utente sceglie la funzione richiesta
2. Uno dei seguenti casi d'uso viene utilizzato:
   (a) Modifica profilo: comprende la modifica dell'anagrafica.
   (b) Visualizzazione profilo: comprende la visualizzazione dell'anagrafica, delle spese effettuate e l'attuale saldo punti della tessera.

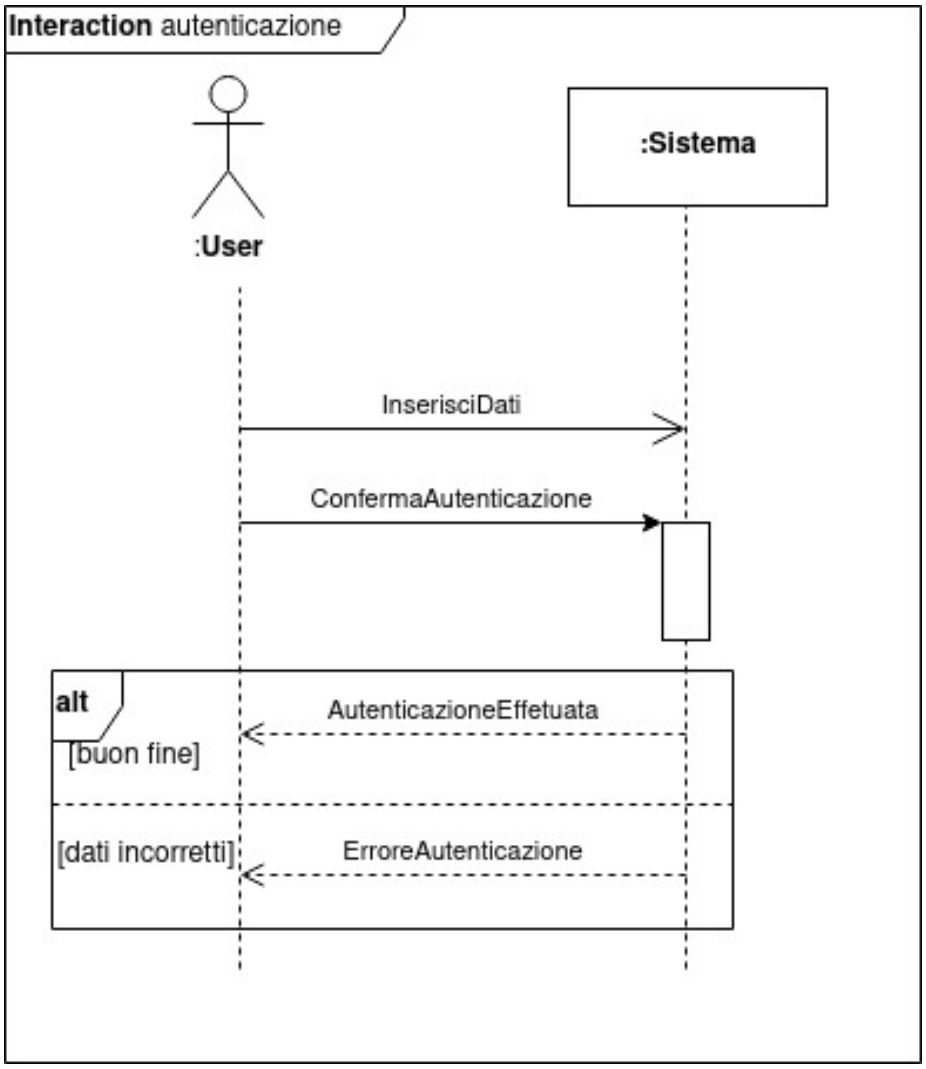

Figure 3: Sequence Diagram dell'autenticazione

Pre condizioni:
L'utente deve essere registrato ed aver fatto il login nel sistema
Post-condizioni:
Se le operazioni di modifica vanno a buon fine lo stato del profilo viene modificato, altrimenti il profilo rimane invariato.
Sequenza Alternativa:
Se i dati inseriti non sono validi il sistema non effettua modifiche.

1.1.2 Gestione carrello:

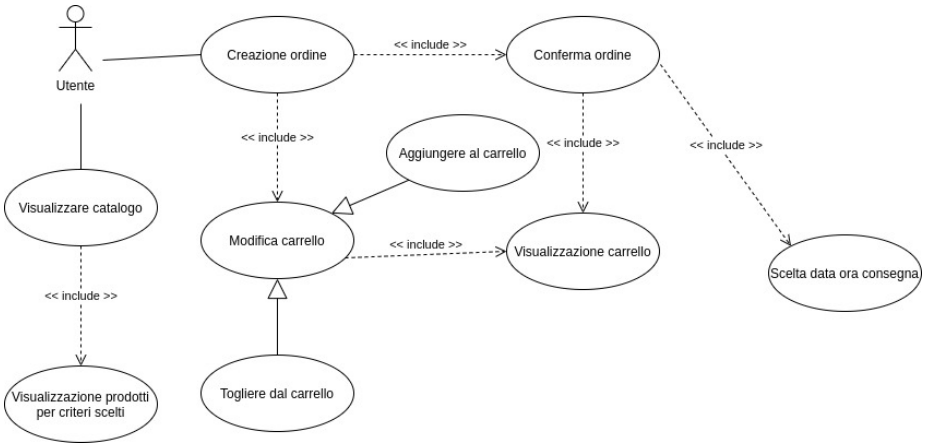

Figure 4: Use case del carrello

Attori :

Utente

Scopo e Descrizione sintetica :

Il sistema permette all'utente di gestire il proprio carrello della spesa. L'utente può visualizzare il proprio carrello, inserire prodotti e togliere i prodotti già inseriti e confermare l'ordine.

Sequenza di eventi :

Questo caso d'uso viene attivato quando l'utente vuole modificare il proprio carrello.

1. L'utente sceglie la funzione richiesta
2. Uno dei seguenti casi d'uso viene utilizzato:
   (a) Modifica al carrello(aggiungere e togliere prodotti)
   (b) Visualizzazione carrello
   (c) Conferma ordine: Viene richiesta la data, l'ora di inizio e di fine in cui può avvenire la consegna della spesa. Il metodo di pagamento utilizzato per effettuare la spesa è quello scelto dall'utente durante la fase di registrazione o di modifica del profilo.

Pre condizioni:

L'utente deve essere registrato ed aver fatto il login nel sistema

Post-condizioni:

Se le operazioni di modifica vanno a buon fine lo stato del carrello viene modificato, altrimenti il contenuto del carrello rimane invariato. Se la scelta della data e orario di consegna vanno a buon fine l'acquisto viene fatto, altrimenti l'acquisto non viene effettuato.

Sequenza Alternativa :

Se durante la conferma il prodotto non è disponibile, viene visualizzato un messaggio di errore per informare l'utente che il prodotto non è disponibile. Se i dati inseriti non sono validi il sistema non effettua modifiche.

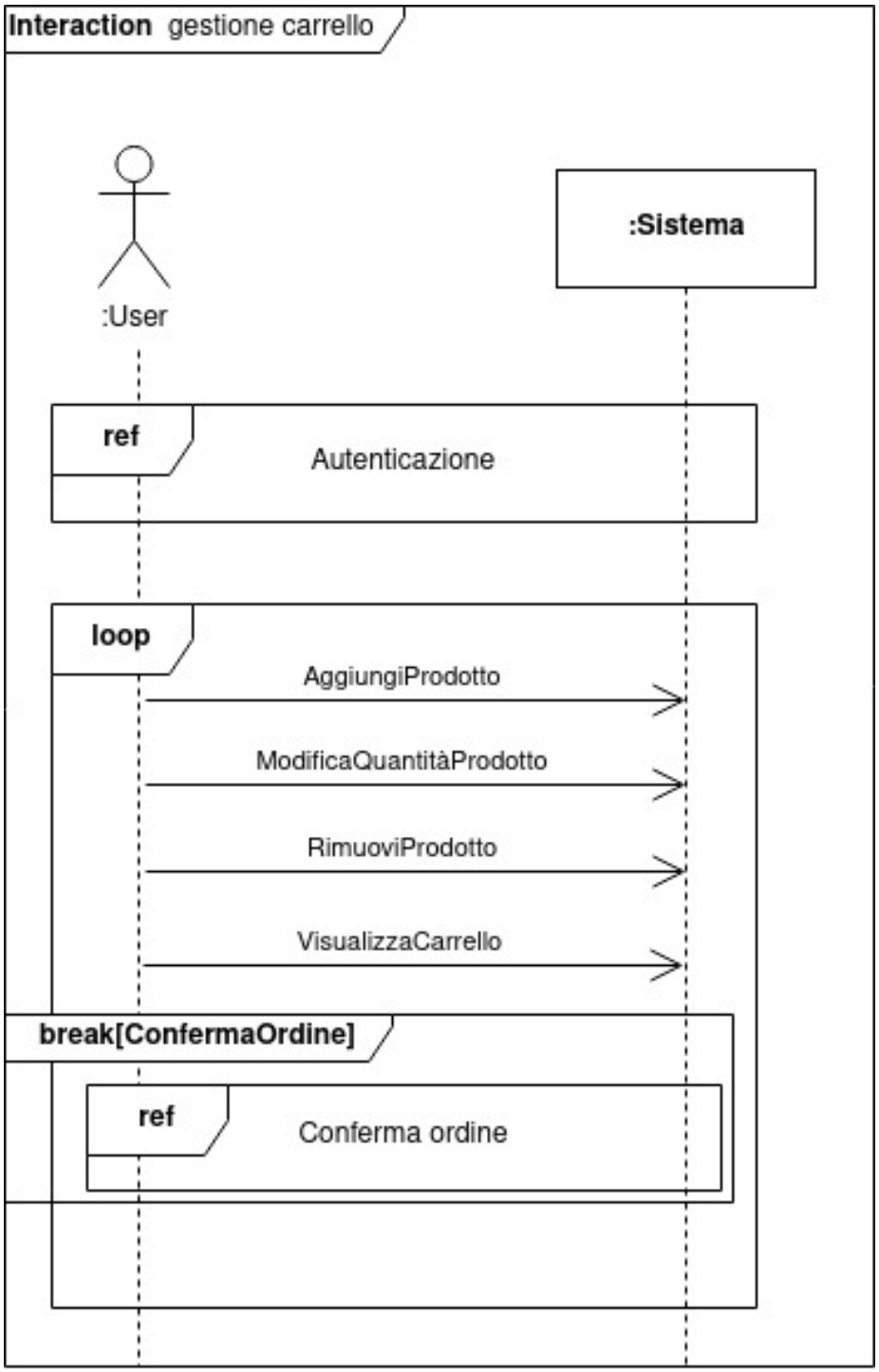

Figure 5: Sequence Diagramm della gestione del carrello

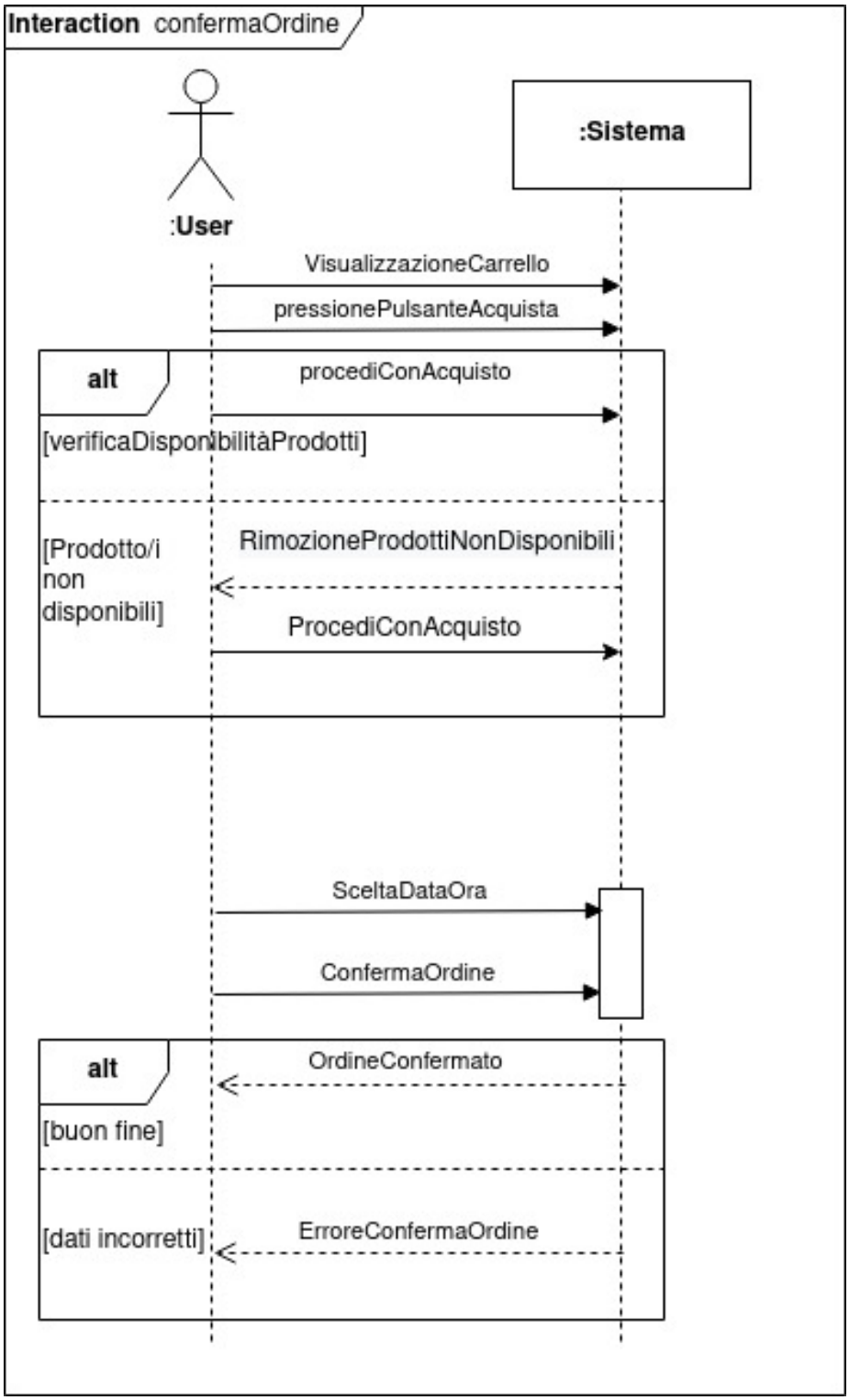

Figure 6: Sequence Diagramm della conferma dell'ordine

1.1.3 Casi d'uso relativi al responsabile reparto

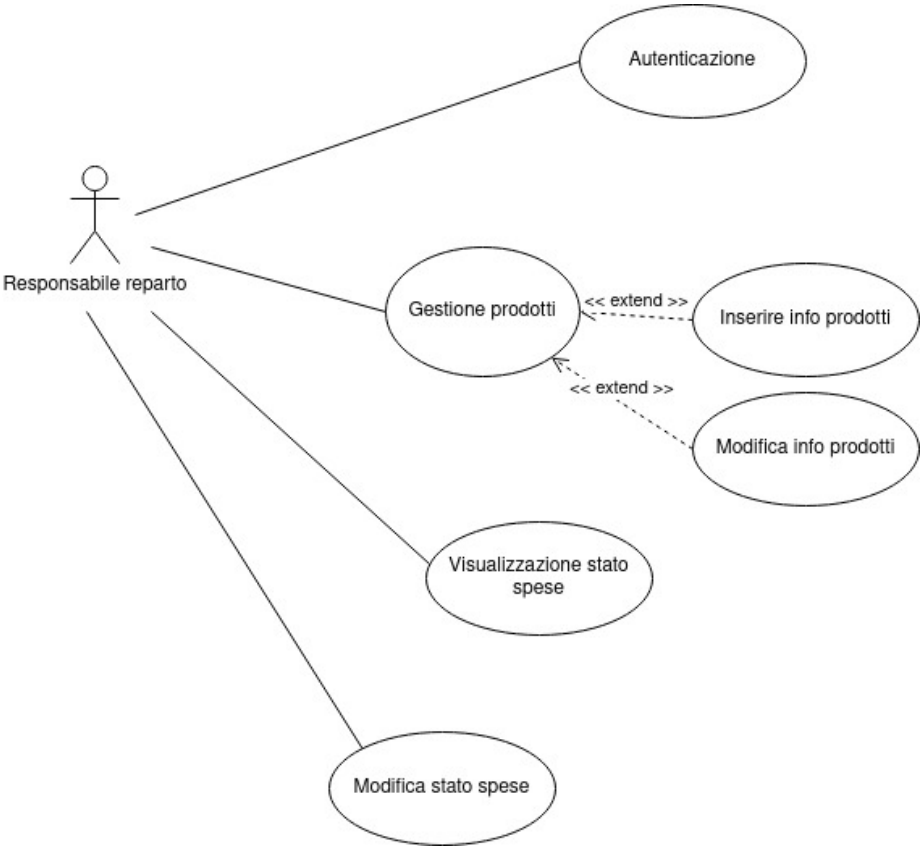

Figure 7: Use case di responsabile reparto per l'autenticazione la gestione delle spese e dei prodotti

Autenticazione:

Attori :
Responsabile reparto
Descrizione :
Procedura di autenticazione del responsabile di reparto
Sequenza di eventi :
Il responsabile inserisce i dati necessari per l'autenticazione
Pre-condizioni:
Il responsabile deve essere presente nel sistema
Sequenza Alternativa :
Se i dati inseriti non sono validi il sistema non effettua l'autenticazione

Visualizzazione stato spese:

Attori :
Responsabile reparto
Descrizione :
Sono visualizzati i dati relativi alle spese dei clienti.
Sequenza di eventi :
Il responsabile accede all'area dedicata
Pre-condizioni:
Il responsabile di reparto aver fatto il login nel sistema
Sequenza alternativa:
Se i dati inseriti non sono validi il sistema non effettua modifiche

Gestione prodotti

Attori :
Responsabile reparto
Descrizione :
Il sistema permette al responsabile di gestire i prodotti del proprio reparto Con la possibilità di modificare e aggiungere i prodotti che sono terminati o che stanno per terminare. Inotre può inserire nel sistema nuovi prodotti che appariranno sul catalogo
Sequenza di eventi :
Il responsabile accede all'area dedicata
Pre-condizioni:
Il responsabile di reparto aver fatto il login nel sistema
Sequenza alternativa:
Se i dati inseriti non sono validi il sistema non effettua modifiche

2 Sviluppo: progetto dell'architettura ed implementazione del sistema

2.1 Note sul processo di sviluppo

Durante lo sviluppo del software sono state utilizzate le seguenti tecnologie:

- Netbeans: per scrittura complilazione del codice.
- Maven: per la gestione e sincronizzazione delle librerie.
- SceneBuilder: per realizzazione interfaccia grafica.
- Git: come strumento di controllo versione.
- GitHub: come servizio di hosting del codice.
- MariaDB: come sistema della gestione del database.
- PhpMyAdmin: come strumento di supporto per visualizzare il database durante lo sviluppo.
- Latex: per scrivere la documentazione.
- draw.io: per disegnare grafici online.
- EasyUml: un plug-in di NetBeans per generare l'UML
- :
- :
- :

tecnologie scenebuilder maven per librerie git e github vcs phpmyadmin mariadb mysql dbms draw.io latex easy uml initdb metodo di sviluppo sviluppo agile e incrementale scrum e trello pair programming pair designing bozze di design interfaccia initdb database relazionale

2.2 Progettazione e pattern architetturali usati

2.2.1 Pattern MVC

Il progetto è strutturato con il Pattern MVC, perché fornisce una coerenza strutturale solida semplificando le fasi di progettazione e implementazione. Inoltre crea un'ottima sinergia con la piattaforma JavaFX.
Il sistema è suddiviso in tre parti:

- **Model:** Sono presenti le classi che definiscono i dati manipolati e implementano la logica business. Inoltre sono presenti le classi che permettono la comunicazione con il Database organizzate seguendo il DAO pattern.

- **View:** Parte esclusivamente dedicata alla visualizzazione dei dati presenti nel modello. In particolare si utilizzano file del formato fxml per rappresentare l'interfaccia grafica.
- **Controller:** Le classi Controller sono utilizzate come mezzo di comunicazione tra il Model e la View, permettono l'interazione dell'utente con il sistema e implementano della logica per validare i dati inseriti dall'utente.

2.3 Pattern Repository

La memorizzazione dei dati avviene attraverso l'utilizzo di un database centrale chiamato Spesa nella quale è stata creata una tabella per ciascuna classe che implementa il DAO pattern.

Le motivazioni che hanno portato alla scelta del repository pattern sono la semplicità d'uso e la centralità del sistema di recupero delle informazioni infatti è necessaria una sola classe per creare la connessione con il database ed eseguire le query (vedi classe ConnectionDb).

2.4 Note su JavaFX

L'implementazione dell'interfaccia grafica è stata fatta utilizzano file fxml costruiti con SceneBuilder. Questo permette di comporre in modularmene l'interfaccia grafica. In varie parti del progetto si utilizza una vista che viene richiamata più volte per generare un contenuto dinamico all'interno di una vista più complessa.(Es. Vedi Catalogo, contiene più istanze di un Prodotto)

2.4.1 CSS

JavaFX permette di utilizzare file .css per modificare l'aspetto dei componenti grafici. Per migliorare l'estetica dell'applicativo é stato utilizzato un tema bootstrap chiamato jbootx.

2.5 Implementazione e design pattern usati

2.5.1 Pattern Sigleton

Il pattern singleton è stato utilizzato nell'implementazione della classe ConnectionDb, allo scopo di instaurare una connessione con il database, eseguire interrogazioni generiche e recuperare i risultati. L'oggetto connectionDb viene istanziato una volta all'avvio dell'applicativo e viene utilizzato da tutta le classi DaoImpl contenti le interrogazioni effettive.

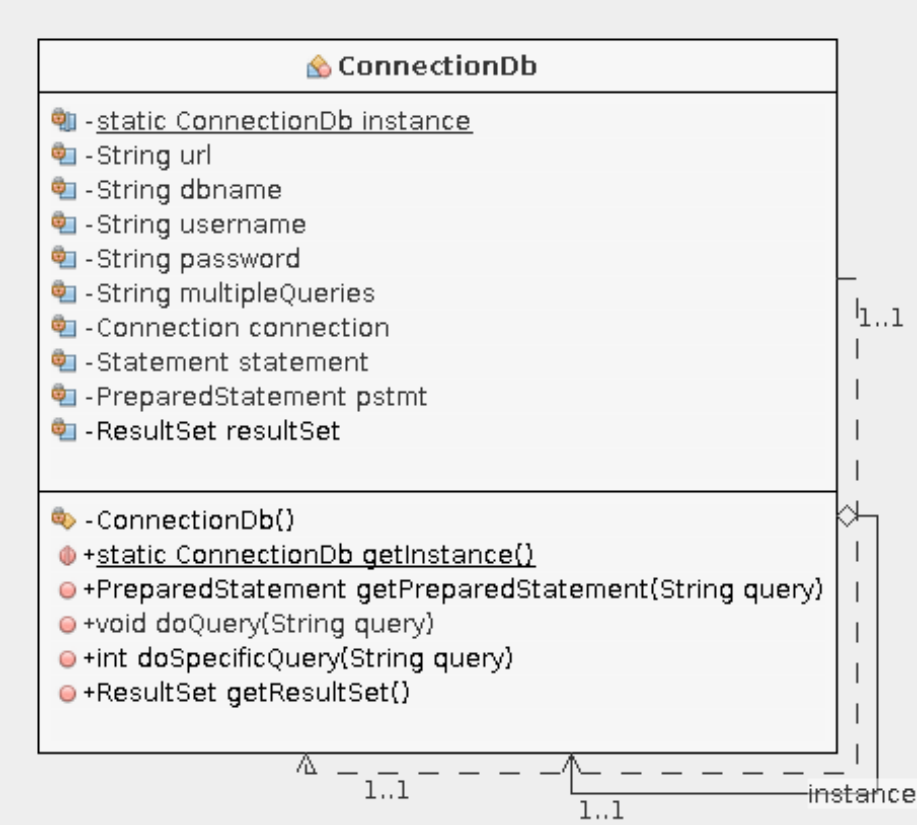

Figure 8: UML della classe ConnectionDb

2.5.2 Pattern Observer e gestione della sessione

Durante lo sviluppo si è presentata la necessita di terner traccia dello stato di alcune istanze di oggetti che sono visualizzati o modificati in più viste. Per esempio se aggiungiamo al carrello un prodotto nel catalogo ci aspettiamo che la lista di prodotti nel carrello sia aggiornata inserendo il prodotto scelto. Per risolvere il problema di un'istanza di un oggetto condivisa tra più viste utilizziamo la classe SessionStorage alla quale si è applicato il pattern Subject-Observer per notificare il cambiamento di stato.
Quindi la classe **SessionStorage** svolge il ruolo di **Subject** che notifica agli Observer il cambiamento del suo stato.
Le classi **Controller** hanno il ruolo di **Observer** ad aggiornano la vista quando il Subject cambia stato.

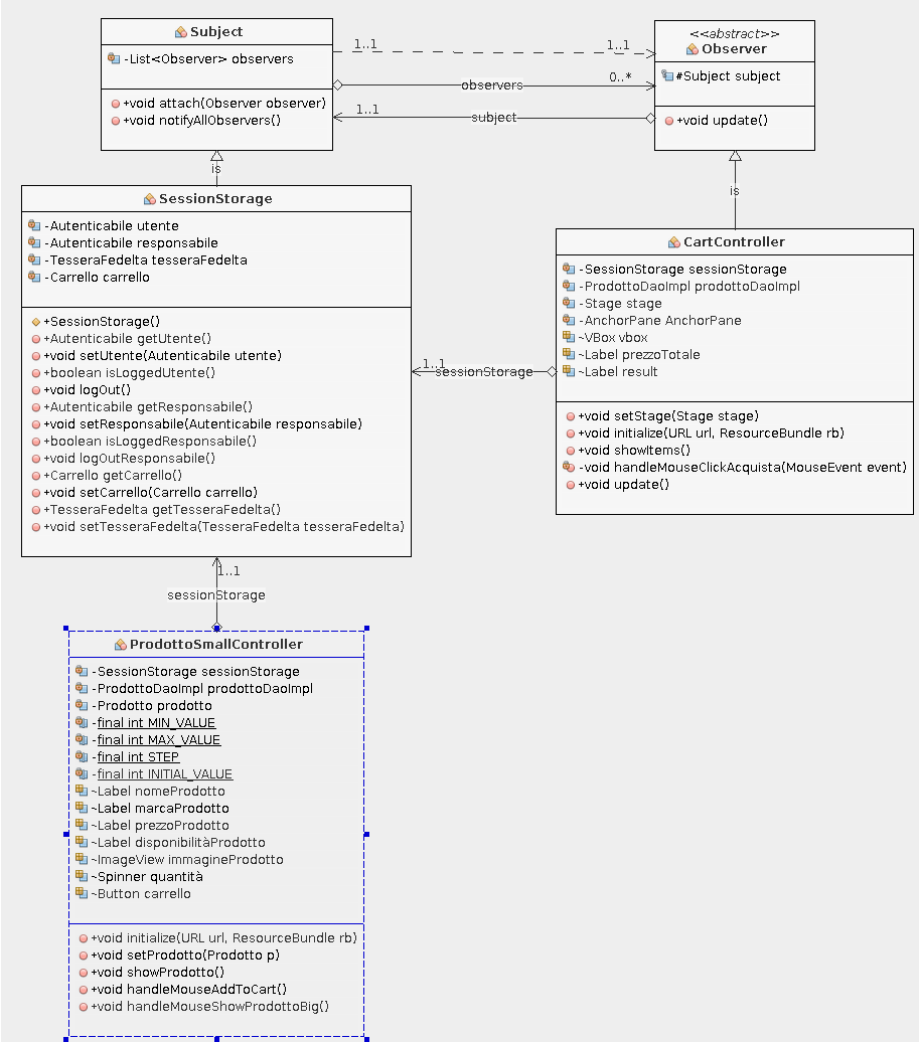

Figure 9: UML delle classi che utilizzano il pattern Observer

2.5.3 Pattern Data Access Object

É stato utilizzato il DAO pattern per garantire accesso standardizzato alle risorse presenti nel database tramite l'utilizzo di interrogazioni SQL. Ogni oggetto nel package Model corrisponde a una tabella nel database e contiene metodi per reperire, inserire e modificare le ennuple contenute nella tabelle.
La seguenti classi utilizzano il pattern DAO:

- Caratteristica
- Prodotto
- Reparto
- ResponsabileReparto
- Spesa
- TesseraFedeltà
- Tipo
- Utente

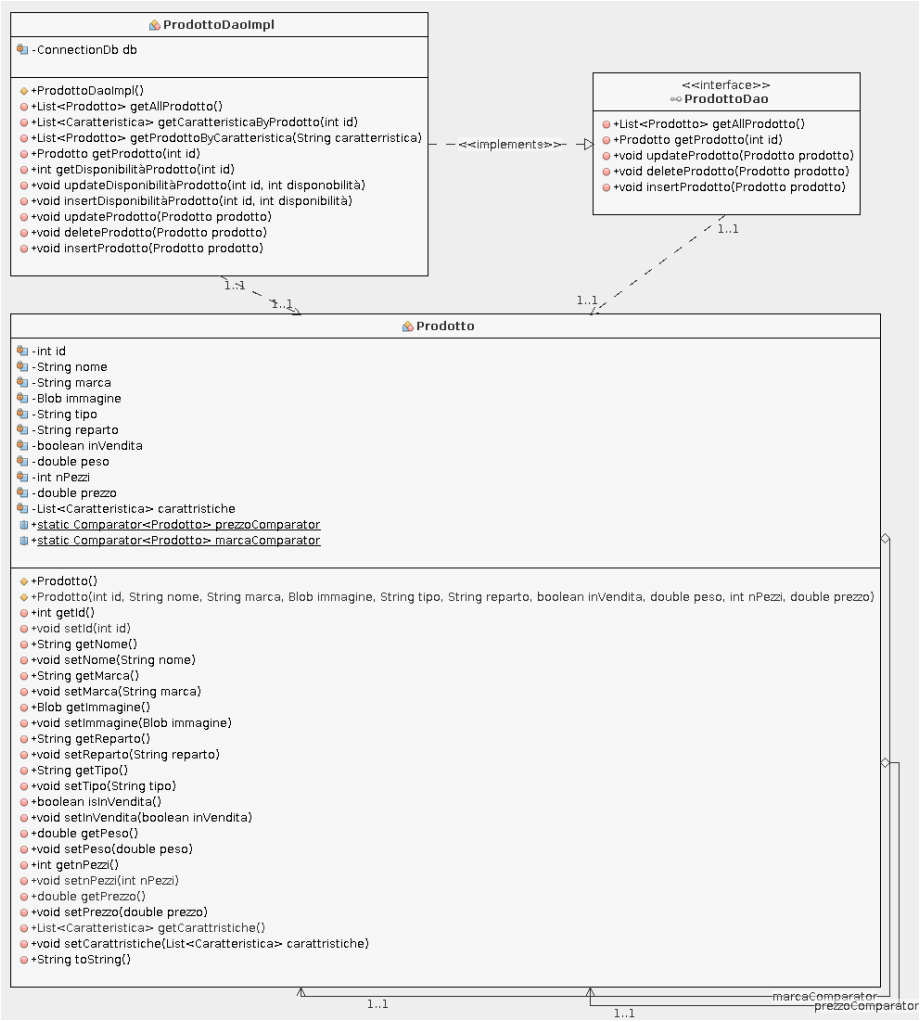

Figure 10: UML della classe Prodotto con pattern DAO

2.5.4 UML delle Classi

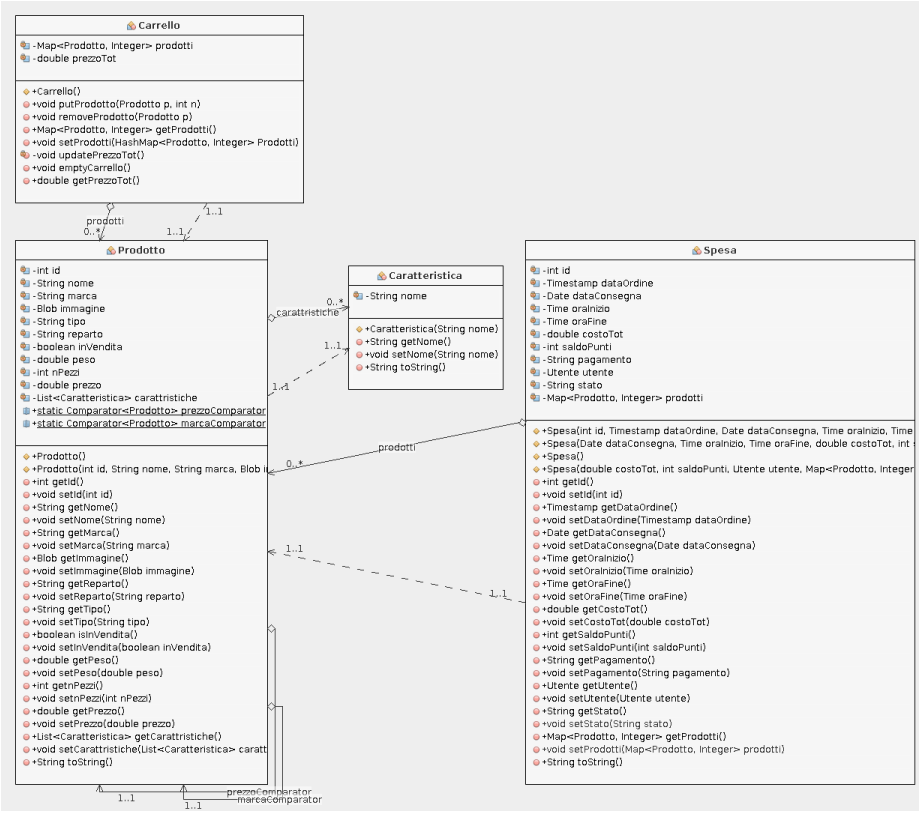

Figure 11: UML della classe Spesa e Carrello in relazione con Prodotto presenti nel model

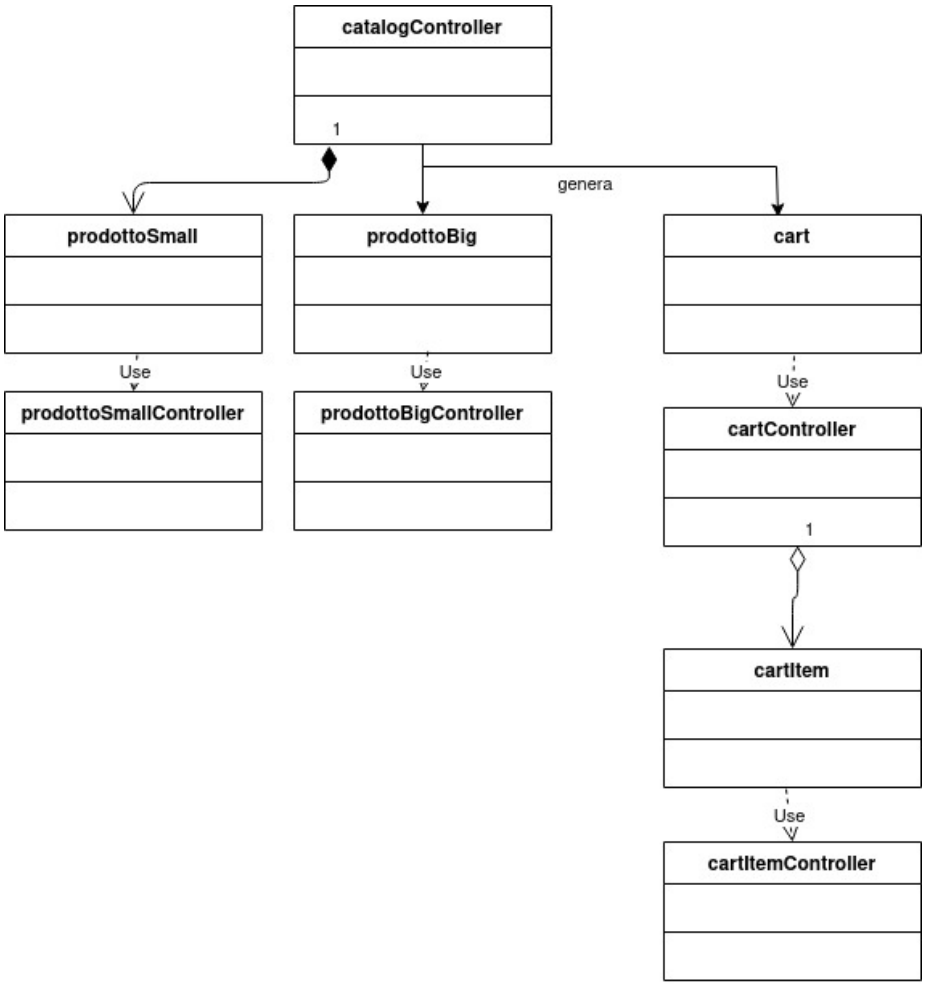

Figure 12: UML delle classi della vista Catalogo

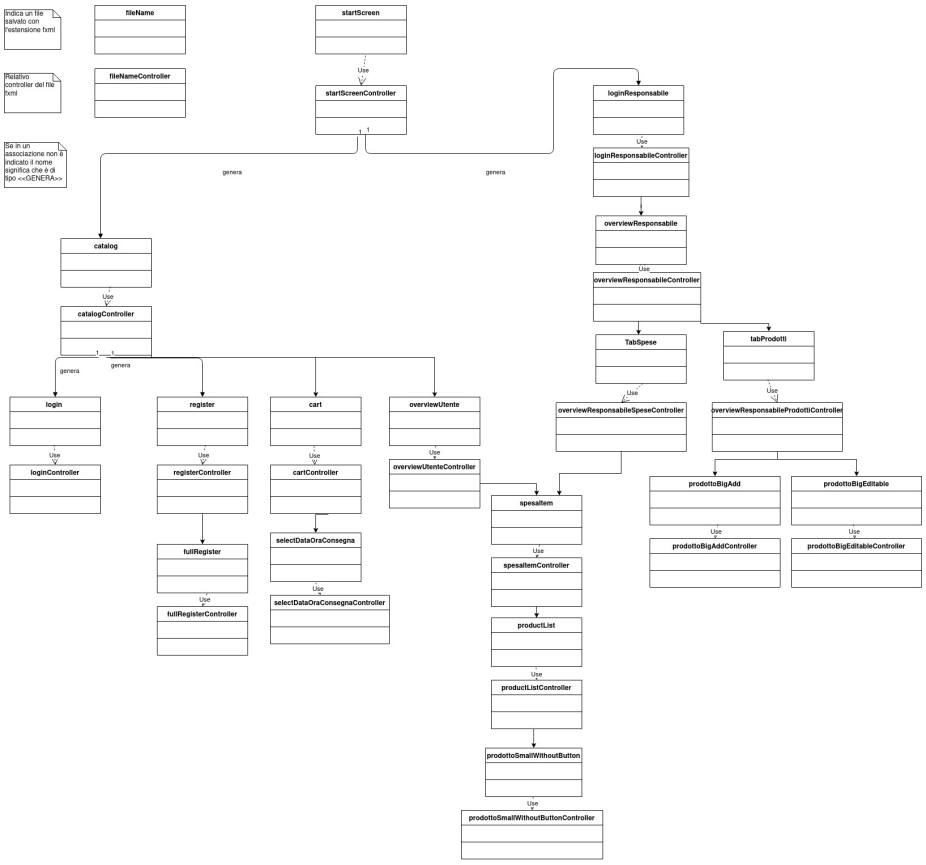

Figure 13: UML delle Classi di tutte le viste

2.6 Diagrammi di sequenza del software implementato.

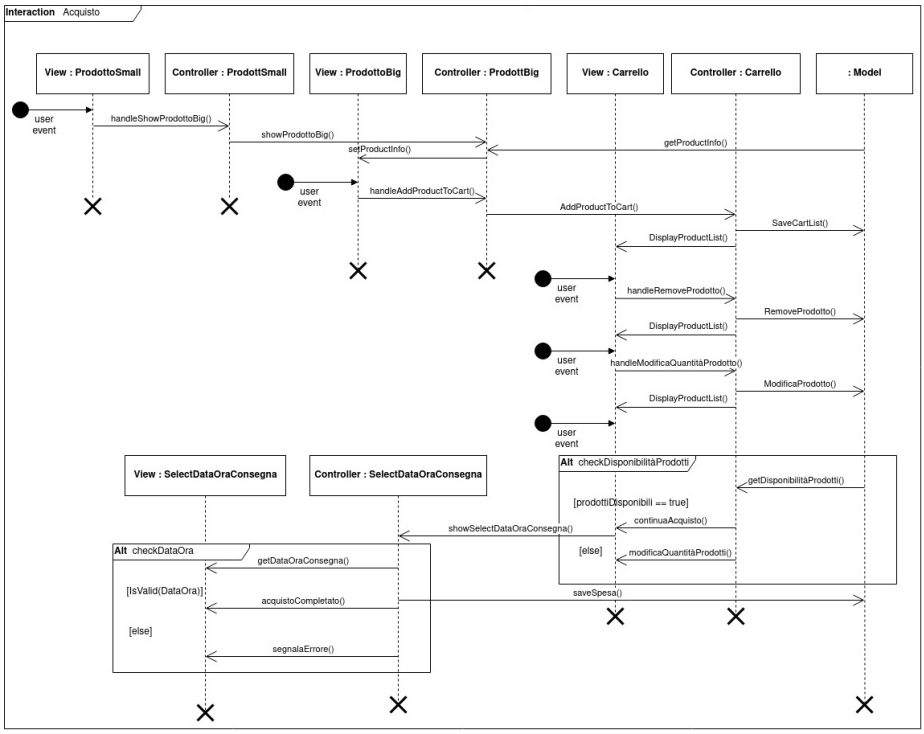

Figure 14: Diagramma di sequenza di un acquisto.

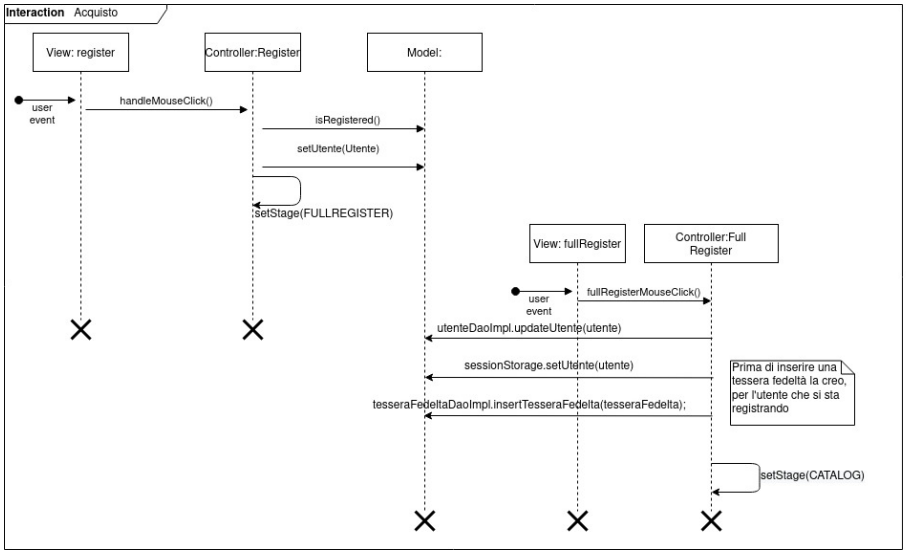

Figure 15: Sequence Diagram della registrazione.

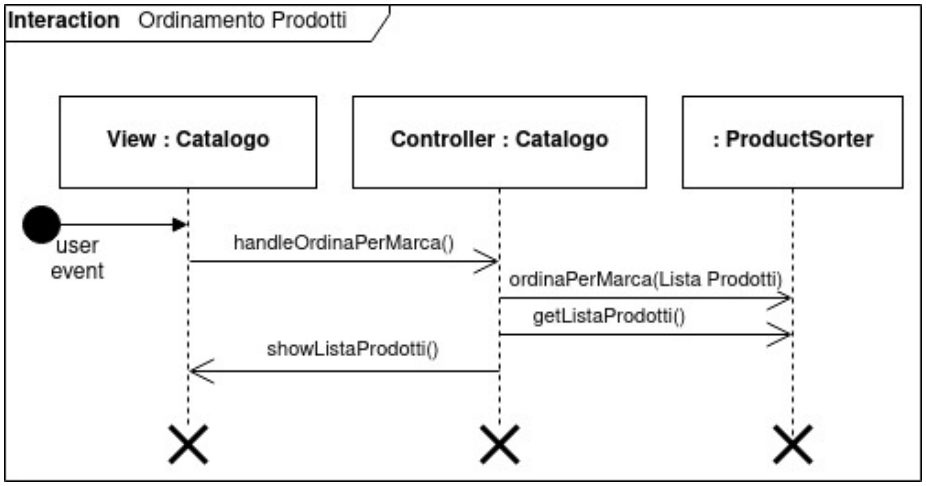

Figure 16: Diagramma di sequenza dell'ordinamento dei prodotti per marca

2.7 Persistenza dati con Database SQL

Utilizzare un database ha portato i seguenti vantaggi:

- L'applicativo può utilizzare un qualsiasi database remoto specificando l'indirizzo e le credenziali d'accesso.
- Possibilità di utilizza più istanze contemporaneamente dell'applicativo affidando la gestione della concorrenza al DBMS.
- Accesso e modifica dei dati facilitato con l'utilizzo delle interrogazioni.

È stato utilizzato un database relazione e come DBMS MariaDB (fork open source du MySql), per implementare la connessione si utilizza **JDBC**.

```
    private static ConnectionDb instance = new ConnectionDb();
    private String url = "jdbc:mysql://localhost:3306/";
    private String dbname = "Spesa";
    private String username = "spesa";
    private String password = "spesa";
    private String multipleQueries = "?allowMultiQueries=true";
    private Connection connection = null;
    private Statement statement = null;
    private PreparedStatement pstmt = null;
    private ResultSet resultSet = null;

    private ConnectionDb() {
    }

    public static ConnectionDb getInstance() {
        return instance;
    }

    public PreparedStatement getPreparedStatement(String query)
         throws SQLException {
        this.connection = DriverManager.getConnection(url +
            dbname, username, password);
        this.pstmt = connection.prepareStatement(query);
        return pstmt;
    }

    public void doQuery(String query) throws SQLException {
        this.connection = DriverManager.getConnection(url +
            dbname + multipleQueries, username, password);
        this.statement = connection.createStatement();
        this.resultSet = statement.executeQuery(query);
        connection.close();
    }

    public int doSpecificQuery(String query) throws
        SQLException {
        this.connection = DriverManager.getConnection(url +
            dbname + multipleQueries, username, password);
        this.statement = connection.createStatement(java.sql.
            ResultSet.TYPE_FORWARD_ONLY,
                java.sql.ResultSet.CONCUR_UPDATABLE);
        this.statement.executeUpdate(query, Statement.
            RETURN_GENERATED_KEYS);
        int id = -1;
        this.resultSet = this.statement.getGeneratedKeys();
        if (resultSet.next()) {
            id = resultSet.getInt(1);
        }
        return id;
```

```
43      }
44
45      public ResultSet getResultSet() {
46          return resultSet;
47      }
```

2.7.1 Diagramma ER

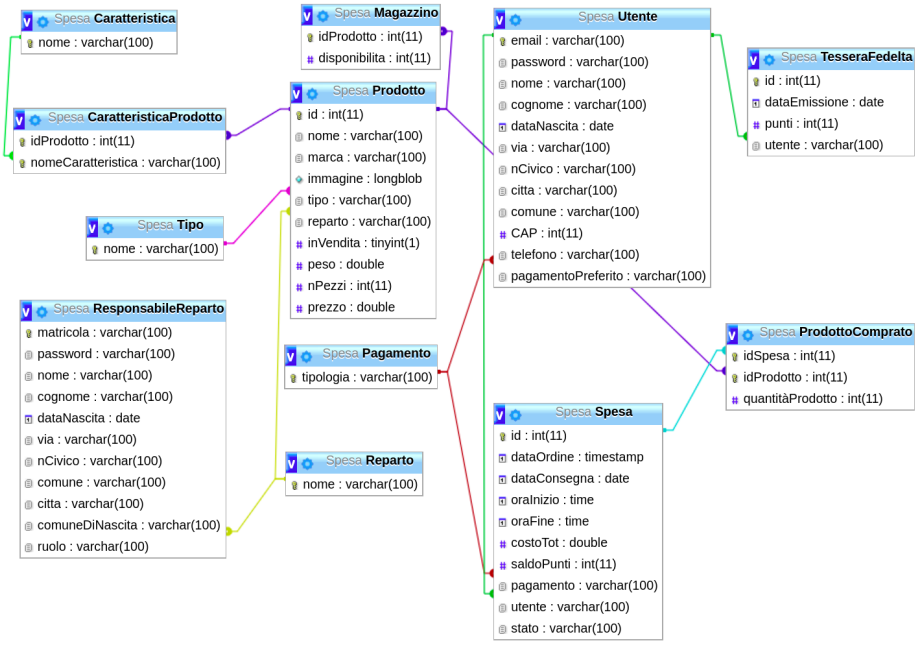

Figure 17: Diagramma Entity Relationship

3 Attività di test e validazione

Per verificare la solidità del prototipo prodotto, sono state effettuate le seguenti attività:

- Verifica della correttezza dei diagrammi rispetto al documento delle specifiche
- Controllo di consistenza tra diagrammi e codice
- Test di unità generici
- Test degli sviluppatori sul prototipo di software
- Test utente generico sul prototipo di software

3.1 Ispezione codice e documentazione

In questa fase è stato effettuato un confronto dei use case diagram con il documento delle specifiche. Successivamente si è passati a produrre i sequence diagram relativi ai use case. Sulla base della documentazione prodotta è stato sviluppato il codice da cui in seguito sono stati generati i diagrammi delle classi. Infine dopo aver accertato l'uso corretto dei pattern pensati durante la fase di progettazione, è stata fatta una nuova ispezione per trovare infrazioni o mancanze all'interno del codice.

3.2 Unit test

Durante questa fase è stata creata una versione alternativa del codice per valutarne il corretto funzionamento. E' stato fatto un test delle classi che presentavano funzionalità più complesse e per verificare che i dati fossero inseriti correttamente all'interno del database.

```
48 import java.sql.Date;
49 import java.sql.SQLException;
50 import java.util.List;
51 import univr.spesaonline.model.Prodotto;
52 import univr.spesaonline.model.ProdottoDaoImpl;
53 import univr.spesaonline.model.Utente;
54 import univr.spesaonline.model.UtenteDaoImpl;
55
56 public class TestRunner {
57     public static void main(String[] args) throws SQLException
          {
58        Prodotto prodotto;
59         List<Prodotto> prodotti;
60         ProdottoDaoImpl prodottoDaoImpl;
61
62         prodottoDaoImpl = new ProdottoDaoImpl();
63
64         prodotto = new Prodotto(0, "Mele", "PinkLady", null, "
               Frutta", "Frutta", true, 1000, 10, 1.75);
65
66         //inserimento prodotto
67         prodottoDaoImpl.insertProdotto(prodotto);
68         prodotti = prodottoDaoImpl.getAllProdotto();
69         //controllo inserimeto
70         for (Prodotto p : prodotti) {
71             if (p.equals(prodotto)) {
72                 System.out.println("Prodotto inserito");
73             }
74         }
75
76         //modifica prodotto
77         prodotto.setPrezzo(3.0);
78         prodottoDaoImpl.updateProdotto(prodotto);
79         prodotti = prodottoDaoImpl.getAllProdotto();
80         //controllo modifica
81         for (Prodotto p : prodotti) {
82             if (p.equals(prodotto)) {
83                 System.out.println("Prodotto modificato");
84             }
85         }
86         //inserimento di un utente
87         Utente u;
88         List <Utente> utenti ;
```

```
89          UtenteDaoImpl utenteDaoImpl = new UtenteDaoImpl();
90
91          Date date = Date.valueOf("1989-03-31");
92          u = new Utente("b.rosi@gmail.com", "Ciao12", "Bianca","
                Rosi", date , "Via De Gasperi", "15","Verona", "
                Verona", 37030, "045521982","Paypal");
93          utenteDaoImpl.register(u.getEmail(), u.getPassword());
94          utenteDaoImpl.updateUtente(u);
95          if(u.equals(utenteDaoImpl.login(u.getEmail(), u.
                getPassword())))
96              System.out.println("Utente aggiunto nel db");
97
98      }
```

3.3 Test degli sviluppatori

3.4 Test utente generico