Progetto Ingegneria del Software

Davide Bleggi Joshua Chapman

Giugno 2020

Contents

Chapter 1

Introduzione

Progetto per la creazione di un sistema informatico per gestire il servizio di spesa online di un supermercato.

1.1 Organizzazione

Date le distanze, i tempi ristretti, e il nostro team composto da sole due persone abbiamo preferito usare al **XP programming**, in grado di adattarsi meglio alle nostre esigenze, e appoggiandoci ad un servizio di **Version Control System** quale **GIT** per poter proseguire anche in autonomia e non solo con l'ausilio del **pair programming**. Quindi durante la progettazione è stato tenuto un approccio **Test First Development** seguito da costanti aggiornamenti di **refactoring**.

1.1.1 Analisi Dei Requisiti

Prima di tutto abbiamo cominciato con l'analisi dei requisiti forniti dalla documentazione. Quindi abbiamo estrapolato 6 macro sezioni principali:

- Registrazione
  - Gli utenti devono essere registrati.
  - Ogni utente registrato accede con email e password.
  - Gli utenti possono specificare un metodo di pagamento preferito.
- Catalogo
  - Se un utente inserisce un prodotto che al momento della conferma della spesa non risulta più disponibile,il sistema segnala la cosa al cliente ed elimina il prodotto dal carrello.

  - Ogni utente registrato accede con email e password.
  - Gli utenti possono specificare un metodo di pagamento preferito.
  - Dopo aver confermato la spesa, l'utente sceglie data e orario della consegna visualizzando le opzioni possibili.

- Carrello
  - L'utente può visualizzare il carrello per modificare la quantità dei prodotti inseriti o rimuovere qualche prodotto.
  - L'utente può ricercare i prodotti per tipo (uova, biscotti, pasta), per marca o per eventuali caratteristiche.

- Effettua spesa
  - Ad ogni spesa vengono accreditati sulla tessera fedeltà un numero di punti pari agli euro spesi nella spesa considerata.

- Profilo utente
  - Il sistema deve permettere agli utenti di accedere al loro profilo, modificare i dati anagrafici, verificare il saldo punti e lo stato delle loro spese.
  - Ogni utente può vedere tutte le spese che ha effettuato nel tempo con il dettaglio dei prodotti acquistati.

- Responsabili reparto
  - Il sistema deve permettere agli utenti di accedere al loro profilo, modificare i dati anagrafici, verificare il saldo punti e lo stato delle loro spese.
  - Ogni utente può vedere tutte le spese che ha effettuato nel tempo con il dettaglio dei prodotti acquistati.
  - I responsabili del reparto spesa online devono autenticarsi per poter accedere al sistema e devono poter verificare lo stato delle spese e provvedere all'inserimento delle informazioni relative ai prodotti.

1.1.2 Framework

Per il progetto si sono ritenuti necessarie una serie di implementazioni di librerie esterne.

Spring

Abbiamo utilizzato Spring (by Netflix) per un rapido sviluppo di un server compatibile con Java.

Okhttp3

Per la comunicazione tra client e Server, abbiamo optato per un libreria in grado di gestire le richieste http per creare una web application.

JavaSQL

Per la gestione del database javaSQL con il driver di SQLite.

JavaFX

Per l'interfaccia grafica è stato utilizzato JavaFX. Scelta perché è il più moderno e permette di essere utilizzato senza ausilio di librerie esterne.

Gradle

Utilizzato per la Build System modulare e per facilitare la compilazione cross platform (importando automaticamente le librerie necessarie).

Chapter 2

Design del Progetto

2.1 Use Cases

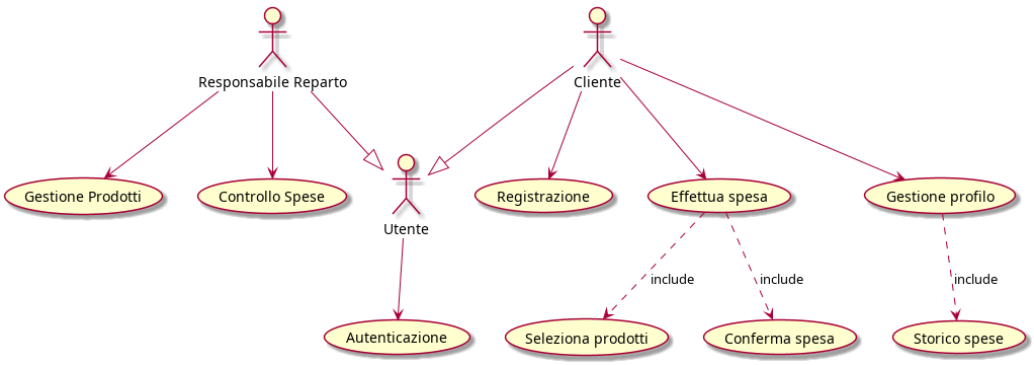

Figure 2.1: Use case diagram

Nel sistema abbiamo identificato 8 use cases. Gli attori necessari per il sistema sono i seguenti:

- Clienti
- Responsabili Reparto
- Utente

I due attori Cliente e Responsabile Reparto sono generalizzati nel attore Utente per tutti gli use case di autenticazione, in quanto andranno ad autenticarsi nello stesso portale. Gli use case possono essere raggruppati in base all'attore a cui appartengono. Le funzioni principali del responsabile reparto

sono di gestire prodotti e lo stato delle spese; gli use case del cliente sono di registrarsi alla piattaforma, effettuare spese e gestire il proprio profilo. Per tutti gli use case diversi dalla autenticazione o registrazione, prendiamo come presupposto che l'utente sia già autenticato.

2.1.1 User Use Cases

Le seguenti sono gli use case del utente con i corrispettivi sequence diagrams.

Autenticazione

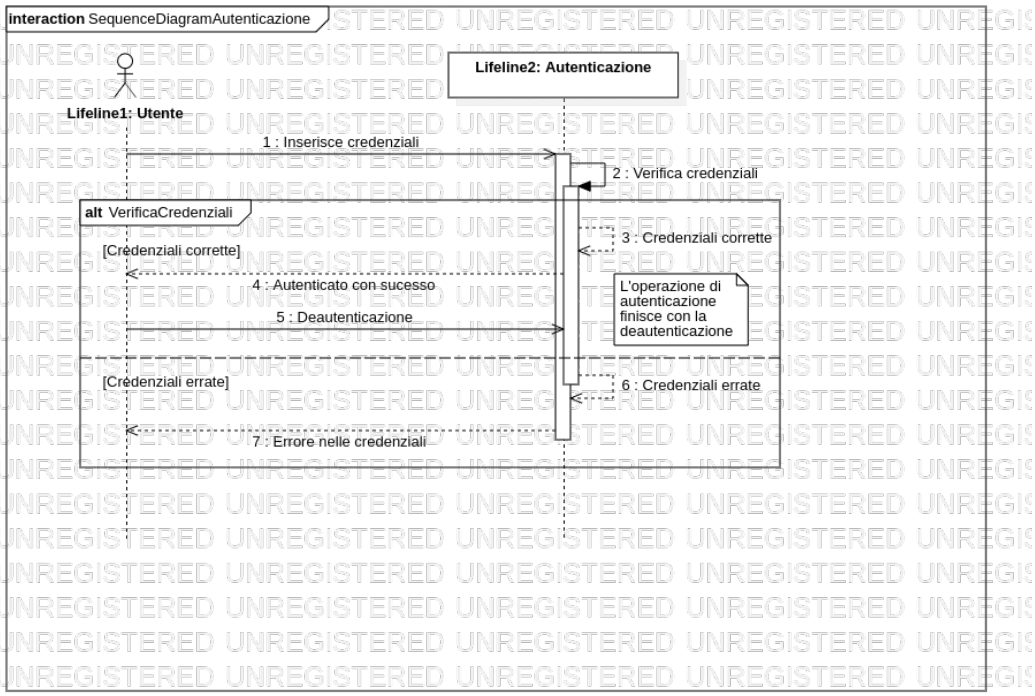

Figure 2.2: Sequence diagram autenticazione

Quando l'utente deve autenticarsi, inserisce le credenziali nel portale. Queste verranno verificate. Nel caso siano corrette verrà autenticato con successo, e potrà successivamente compiere il logout. Nel caso siano incorrette verrà restituito un errore.

2.1.2 Responsabile Use Cases

Le seguenti sono gli use case del responsabile del reparto con i corrispettivi sequence diagrams.

Gestione Prodotti

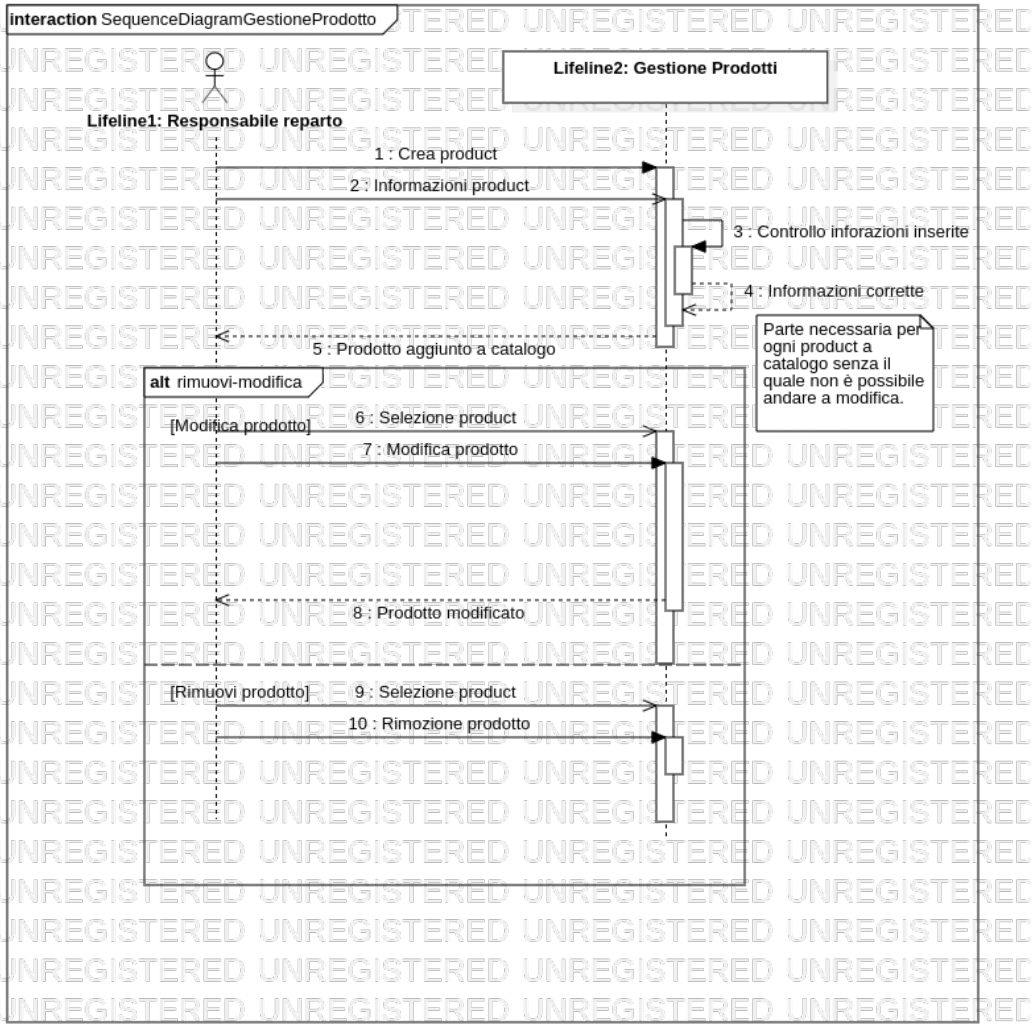

Figure 2.3: Sequence diagram gestione prodotti

Per la gestione dei prodotti il responsabile ha la possibilità di creare un nuovo prodotto, modificarne o rimuoverne uno già esistente. La vista iniziale

è una lista dei prodotto, il responsabile può selezionarne uno e decidere se modificarlo o rimuoverlo direttamente.

Controllo Spese

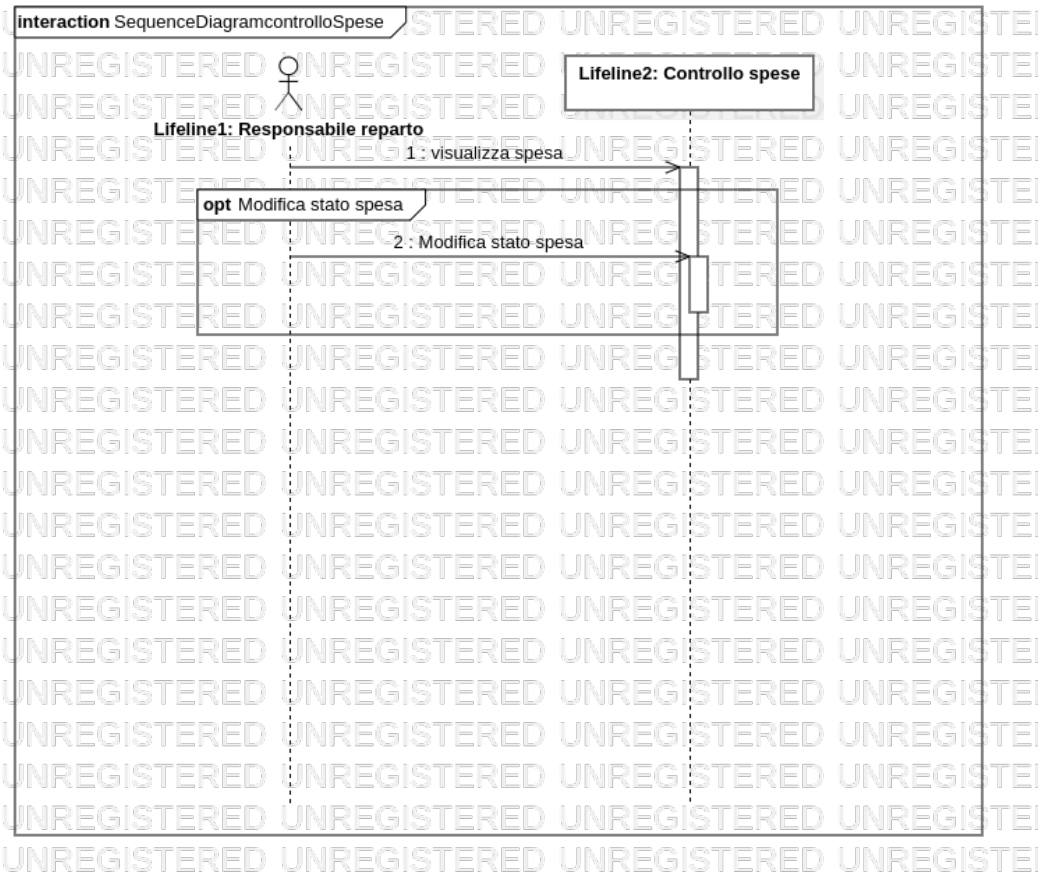

Figure 2.4: Sequence diagram controllo spese

In fine per il controllo delle spese il responsabile ne visualizza una lista con tutte le informazioni riguardanti ogni ordine e ha la possibilità di modificare lo stato delle spese nella lista.

2.1.3 Client Use Cases

Le seguenti sono gli use case del cliente con i corrispettivi sequence diagrams.

Registrazione

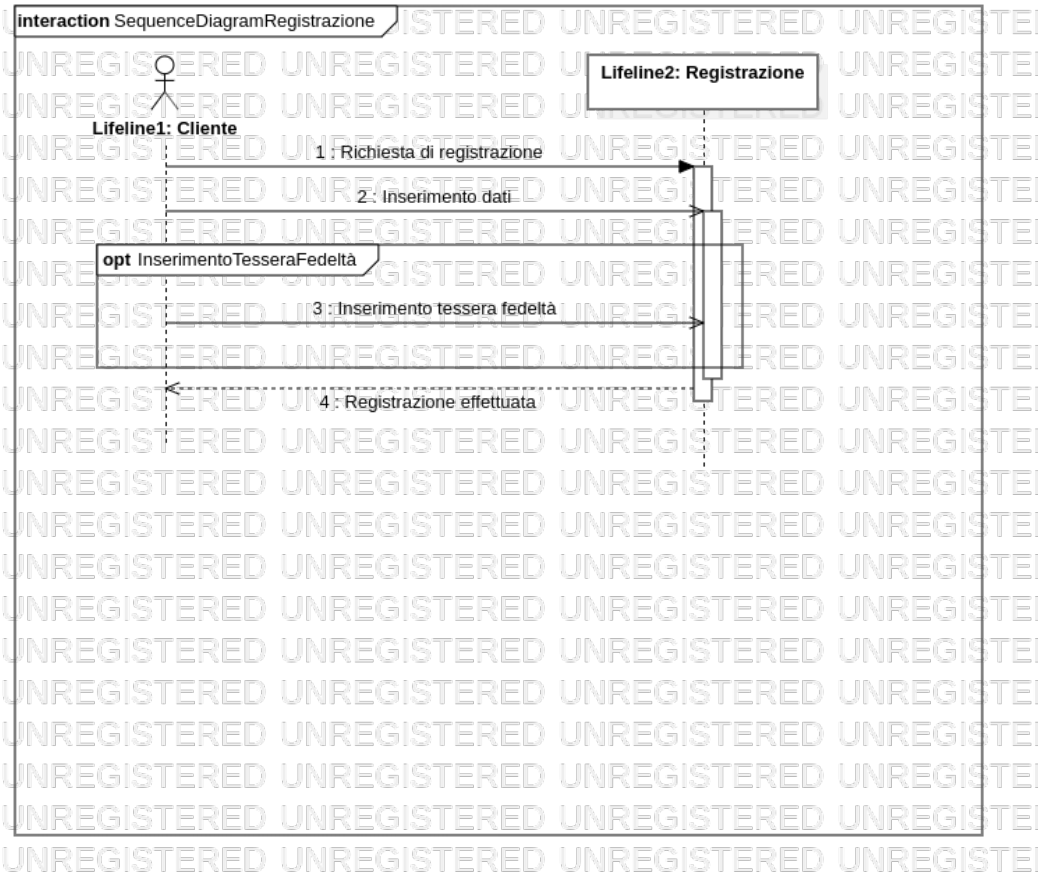

Figure 2.5: Sequence diagram registrazione

Il cliente accede alla vista della registrazione e dopo aver inserito i propri dati ha la possibilità di inserire una tessera di fedeltà, facoltativa.

Effettua Spesa

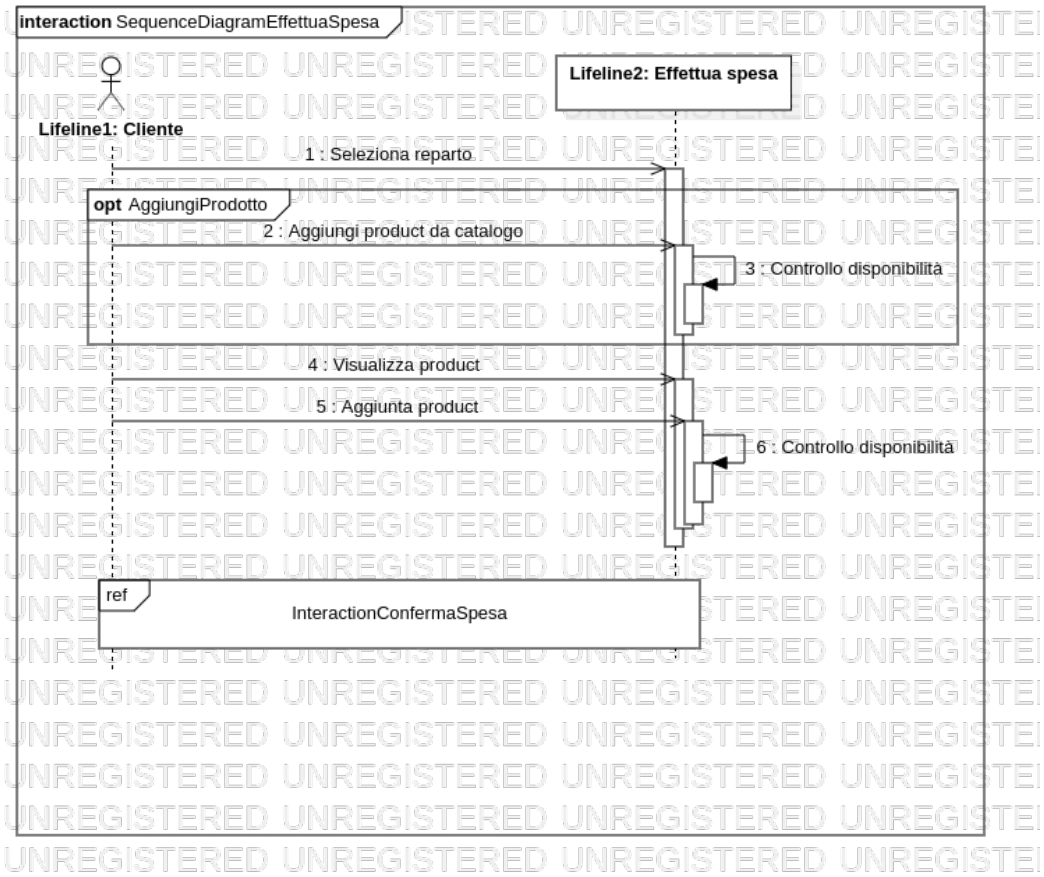

Figure 2.6: Sequence diagram effettua spesa

Per effettuare la spesa il cliente prima visualizza il catalogo dei prodotti. Qui può selezionare le diverse categorie o cercare i prodotti che vuole aggiungere al carrello.

Conferma Spesa

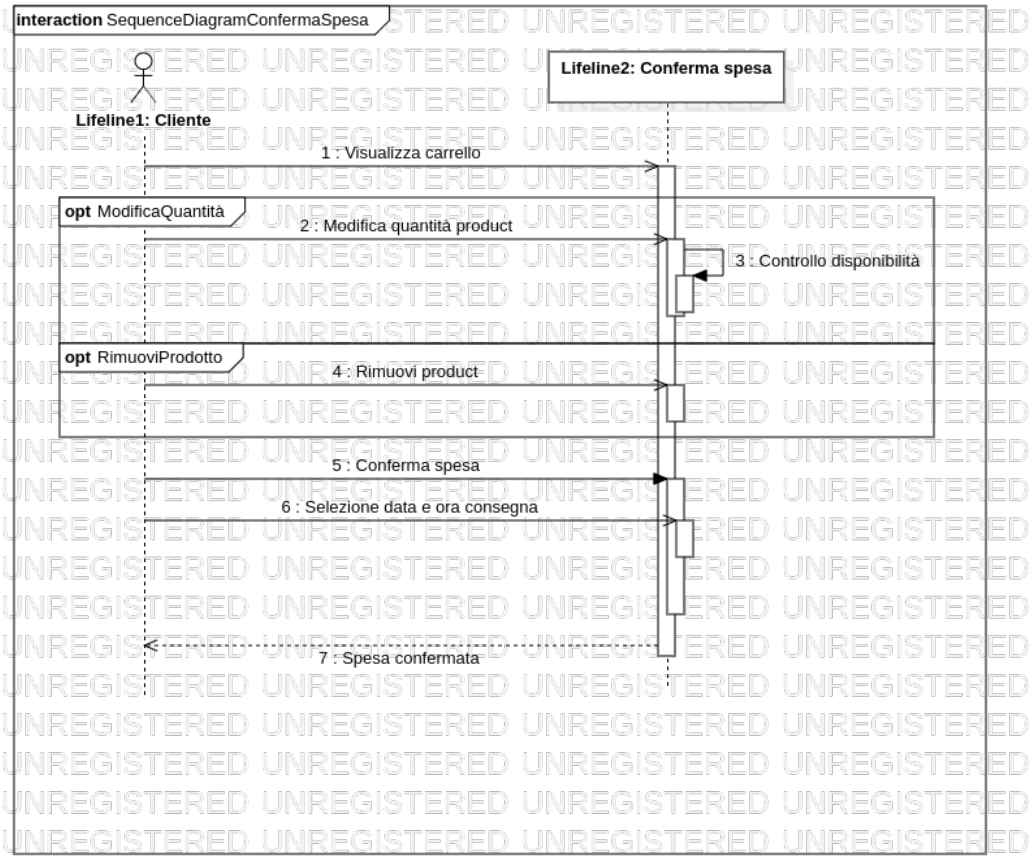

Figure 2.7: Sequence diagram conferma spesa

Quando ha aggiunto almeno un prodotto al carrello può visualizzarli e modificarne la quantità o rimuoverli. Quando ha tutti i prodotto che gli servono può confermare l'ordine e selezionare una data e un ora per la consegna.

Gestione Profilo

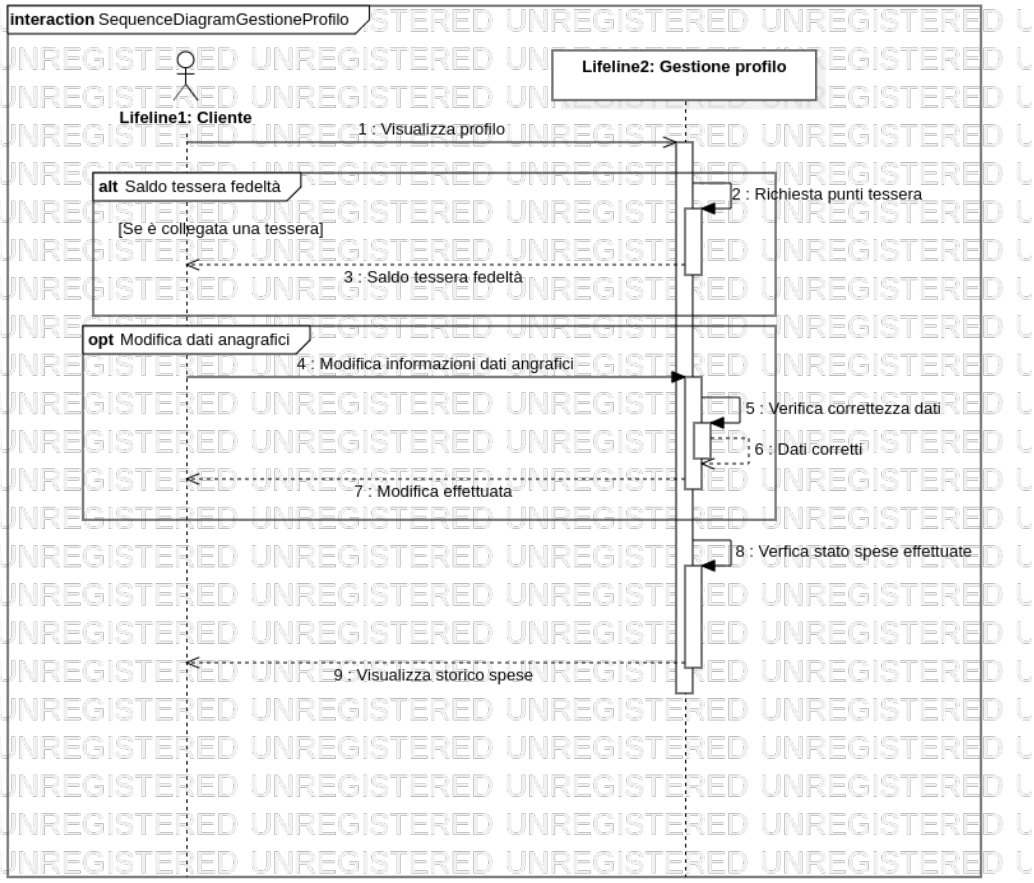

Figure 2.8: Sequence diagram gestione profilo

Nella gestione del profilo il cliente può vedere le proprie informazioni anagrafiche e modificarle. Inoltre se ha registrato una tessera fedeltà, sarà possibile vedere il saldo dei punti guadagnati. Dalla visualizzazione del profilo sarà possibile anche navigare alla visualizzazione dello storico delle spese.

2.2 Activity diagram

Di seguito sono riportati gli activity diagram delle principali funzioni che dovrà avere il sistema.

2.2.1 Autenticazione

Il **cliente o il responsabile del reparto** inseriscono le credenziali username e password. Il sistema le controlla e ritorna un messaggio di errore se queste sono scorrette, altrimenti crea una sessione che si occuperà della deautenticazione nel caso di Logout.[1].

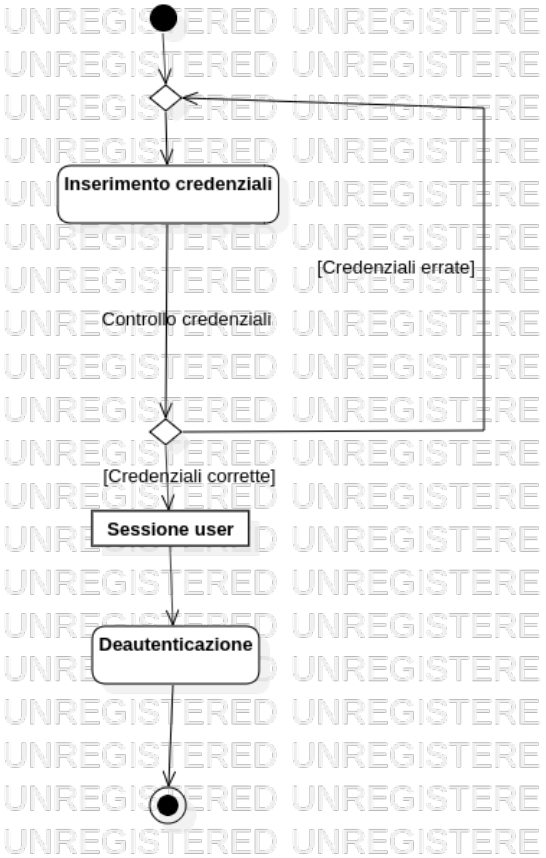

Figure 2.9: Activity diagram autenticazione

[1]Nell'implementazione finale è stato aggiunto un salvataggio della sessione, quindi con la chiusura dell'applicazione non si raggiungerà necessariamente alla deautenticazione e quindi alla conclusione del Activity Diagram

2.2.2 Conferma Spesa

Il **cliente** registrato un Visualizza il proprio carrello. Una volta all'interno del carrello può Modificare le quantità, rimuovere il prodotto o confermare la spesa. Se modifica le quantità bisogna tener in considerazione la quantità di prodotti disponibili. Altrimenti se conferma bisognerà richiedere le informazioni per la consegna.[2]

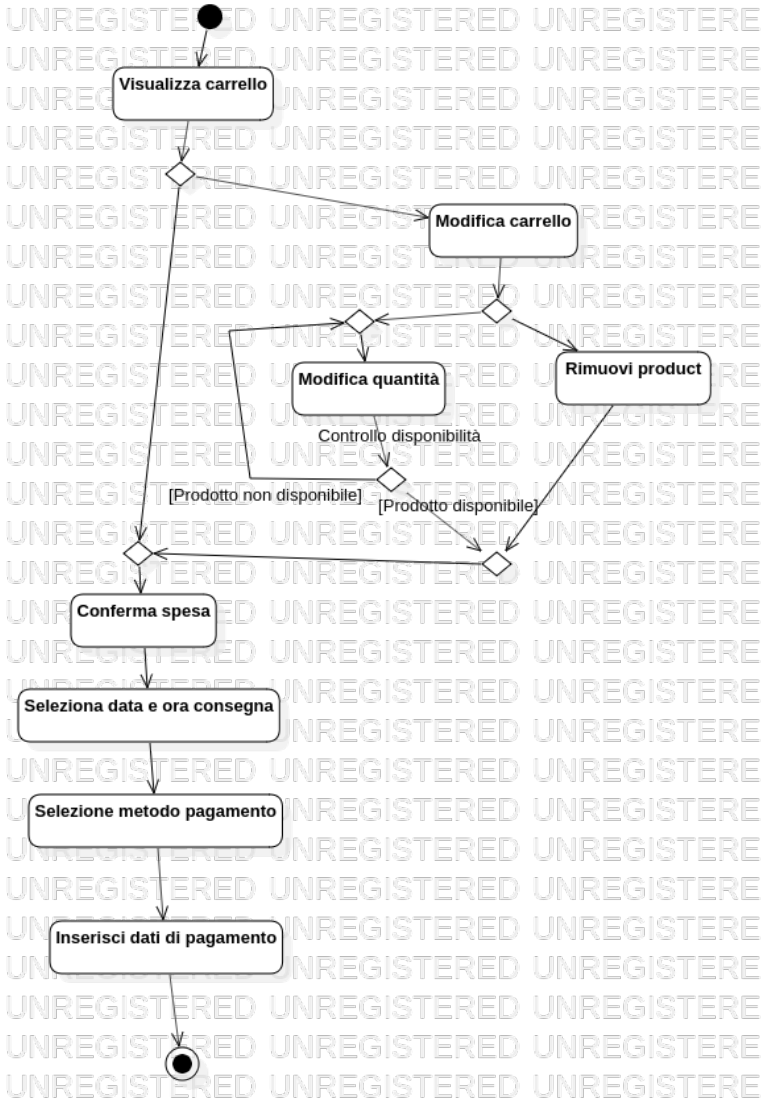

Figure 2.10: Activity diagram Conferma Spesa

[2]Il metodo di pagamento viene ottenuto direttamente dalla session dell'utente

2.2.3 Visualizza Spesa

Il **responsabile di reparto** una volta autenticato e all'interno della visualizzazione della spesa, il responsabile può visualizzarne i prodotti di una spesa o aggiornarne lo stato.

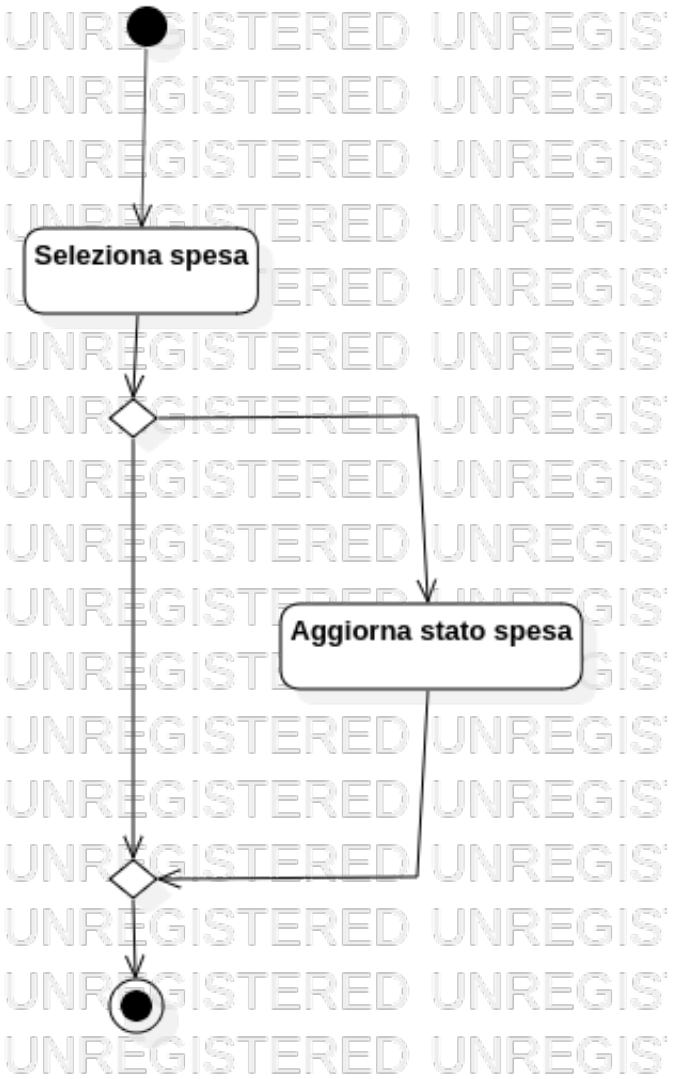

Figure 2.11: Activity diagram Visualizza Spese

2.2.4 Effettua Spesa

Il **cliente** una volta autenticato e all'interno del catalogo potrà selezionare un reparto (di default preselezionato su All, contente tutti i prodotti a catalogo) e potrà aggiungere prodotti al carrello sia dalla lista del catalogo direttamente sia cliccando sul item corrispondente e aggiungiendolo al carrello una volta aperta la scheda del prodotto corrispondente. Prima dell'aggiunta il sistema controlla la disponibilità del prodotto e se è disponibile lo aggiungie al carrello.

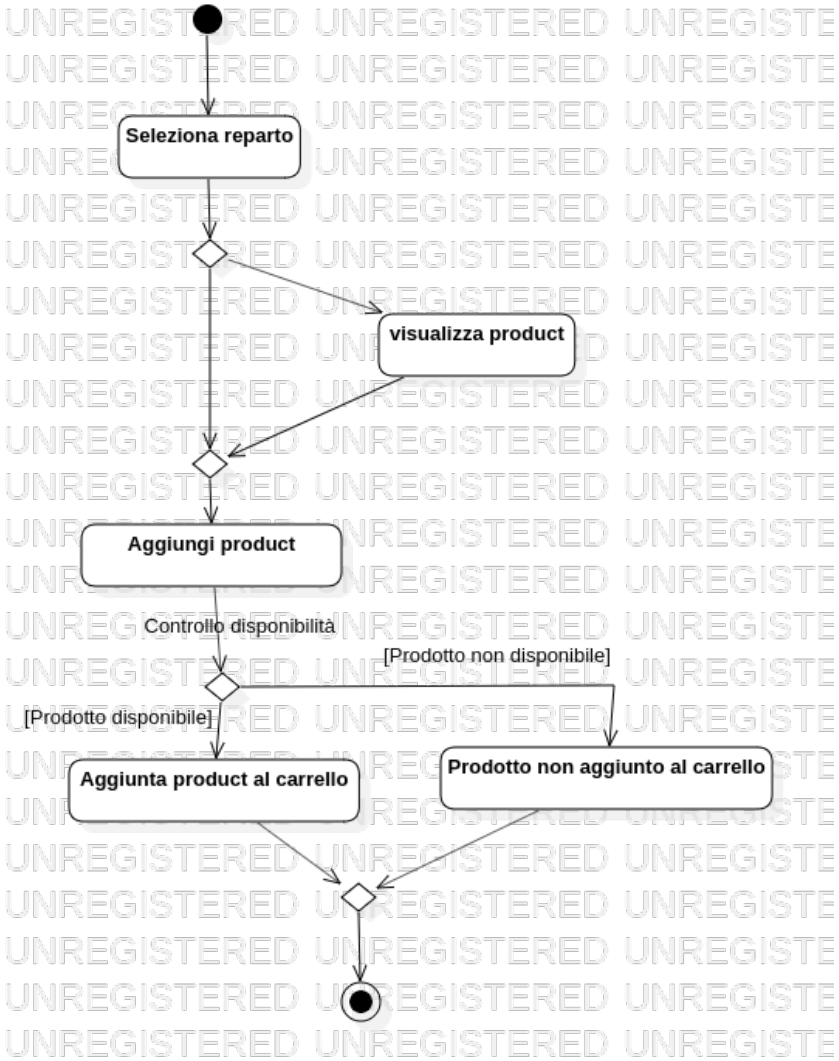

Figure 2.12: Activity diagram Effettua Spesa

2.2.5 Modifica/Aggiungi Prodotto

Il **responsabile del reparto** una volta autenticato ed entrato nella gestione dei prodotti potrà decidere se creare un nuovo prodotto o selezionarne uno da catalogo per procedere alla modifica o alla rimozione. Nel caso di creazione del prodotto inserisce le informazioni del prodotto che vengono controllate dal sistema prima dell'aggiunta nel catalogo. Nel caso di modifica della quantità o modifica del prodotto sarà possibile selezionare il prodotto dalla lista. Una volta superati i controlli del sistema il prodotto verrà aggiornato con le nuove informazioni.[3]

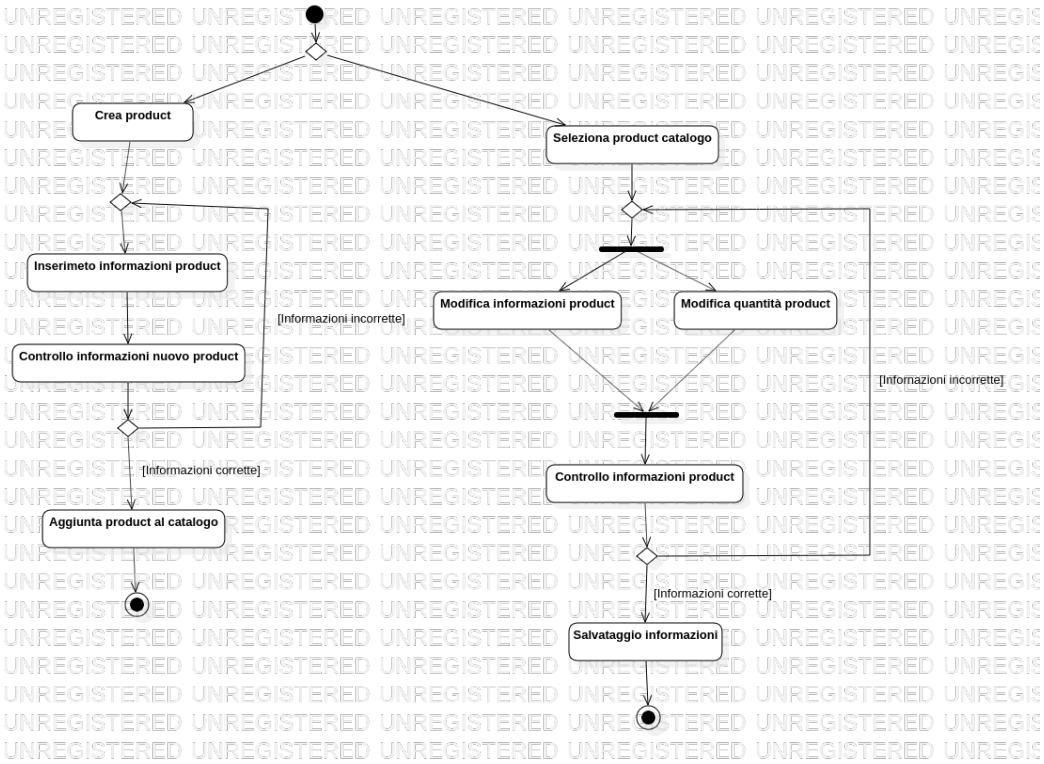

Figure 2.13: Activity diagram Modifica o Aggiungi Prodotto

[3]Nell'implementazione è stata aggiunta la opzione di selezionare il prodotto selezionandolo dalla lista

2.2.6 Gestione Profilo

Il **cliente** una volta autenticato e aver acceduto al proprio profilo può modificare i propri dati o visualizzare lo stato delle proprie spese. Nel caso di modifica dei dati il sistema provvederà alla verifica dei dati inseriti. Se i dati risultano corretti allora i salva.

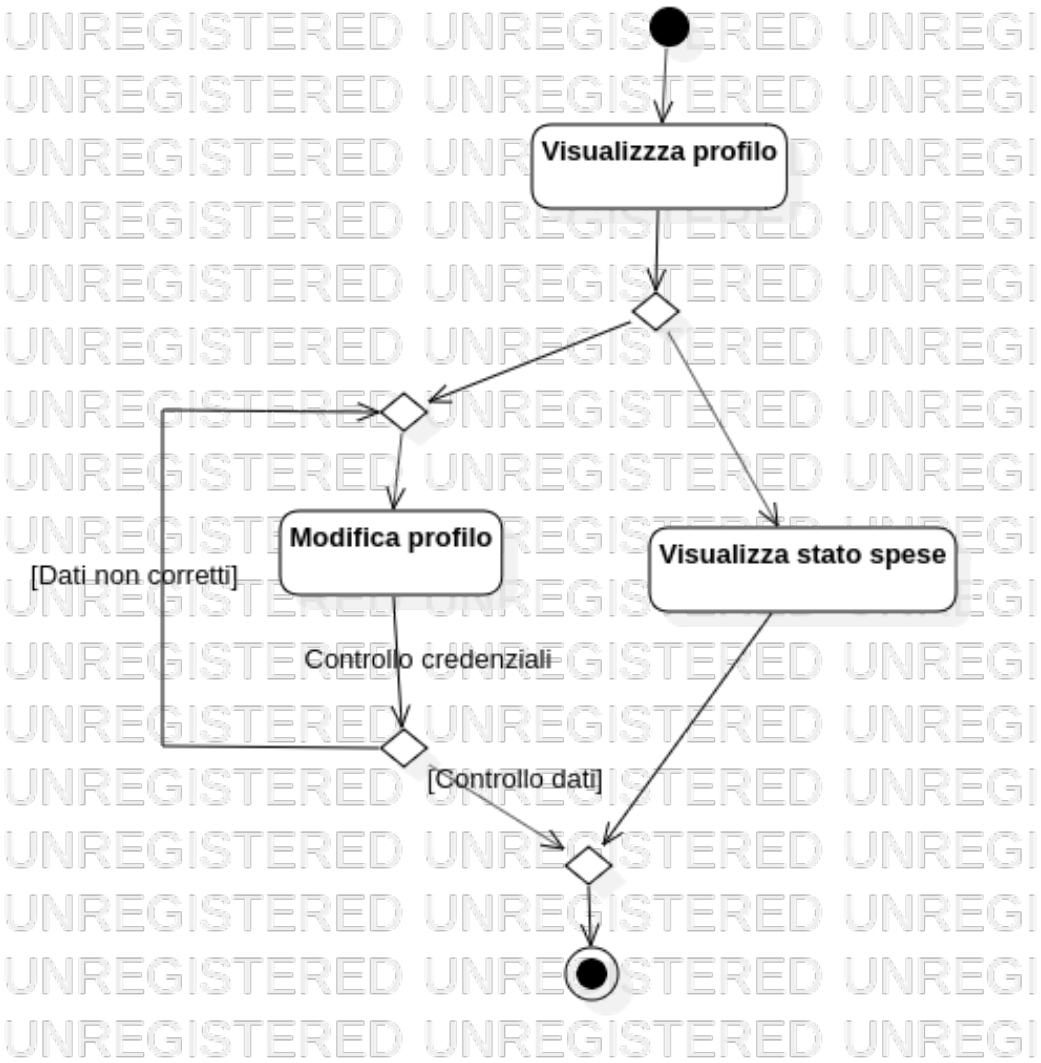

Figure 2.14: Activity diagram Gestione Profilo

2.2.7 Registrazione

Il **cliente** prima di autenticarsi entra in registrazione ed inserisce i propri dati e se vuole la propria tessera fedeltà. Quindi il sistema verifica i dati inseriti, se sono corretti registra il nuovo utente altrimenti segnala un errore nella compilazione dei campi.

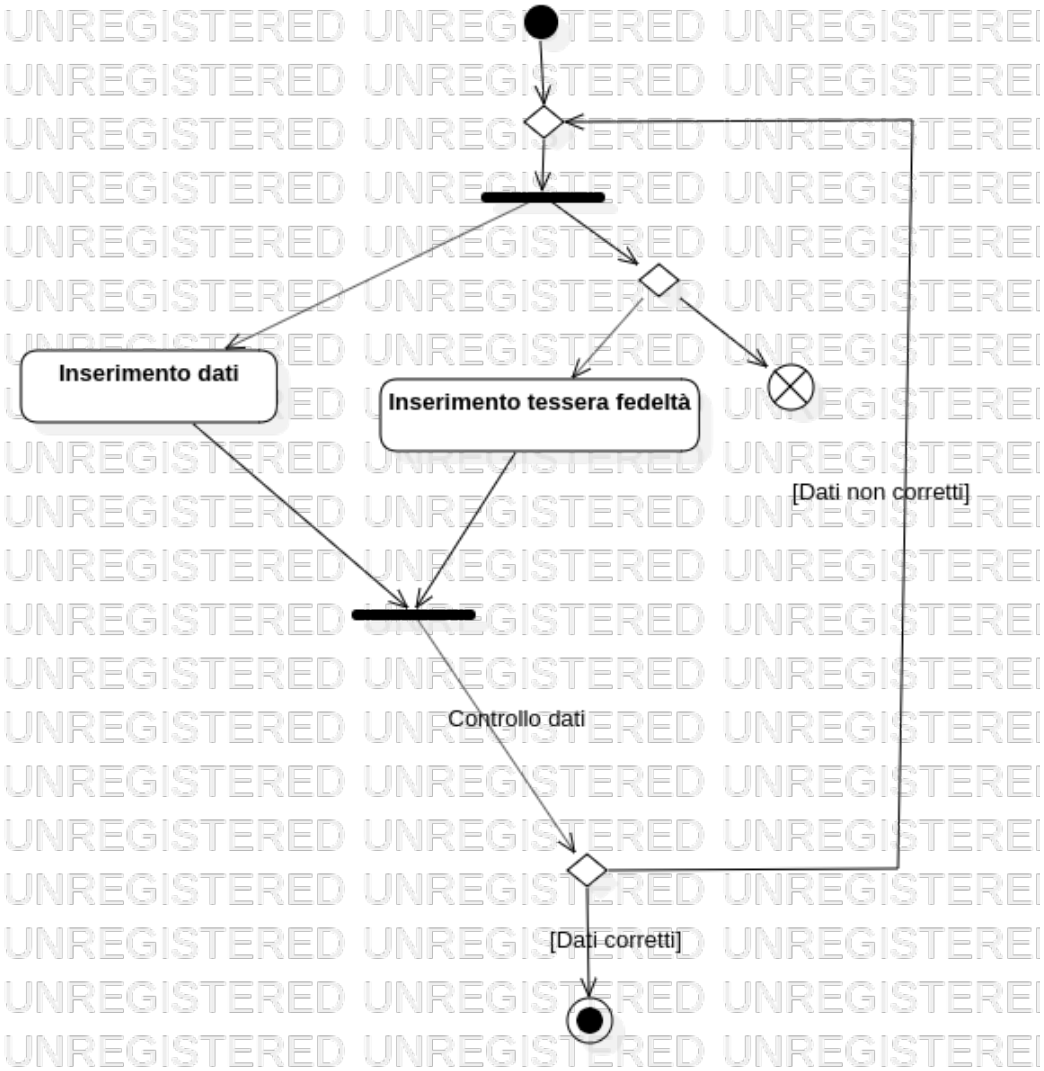

Figure 2.15: Activity diagram Registrazione

2.3 Pattern Architetturali

Il pattern architetturale principale del sistema è il Client e Server, mentre sono stati utilizzati diversi pattern per i due rispettivi sistemi.

2.3.1 MVC Model View Controller

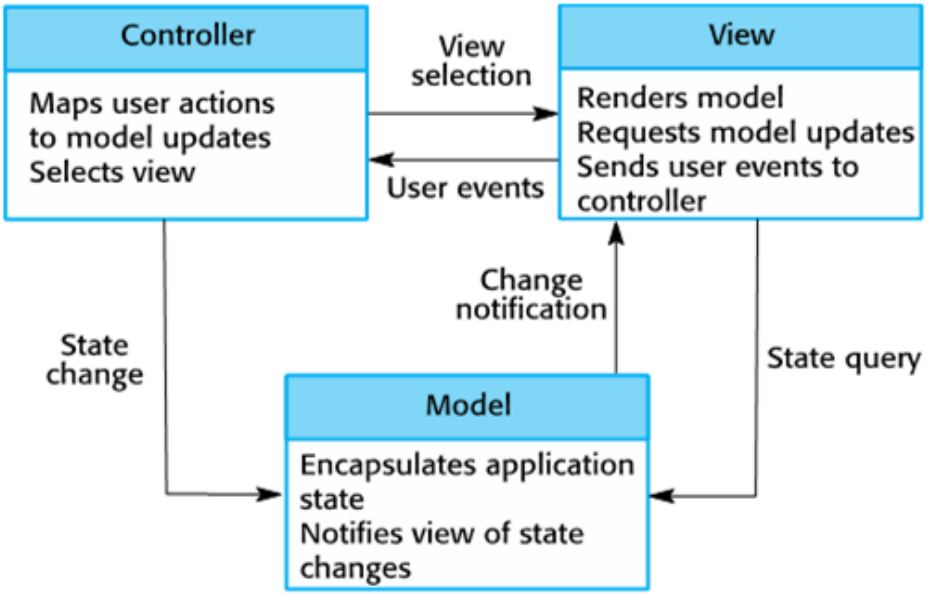

Figure 2.16: Schema Model View Controller

Il model View Controller è stato usato come pattern architetturale dato dall'utilizzo implicito di questo sistema da parte di JavaFX.

2.3.2 Repository

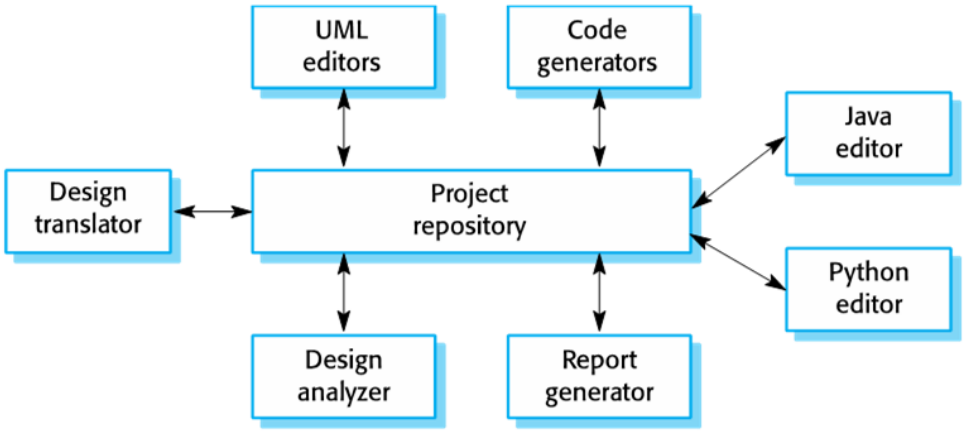

Figure 2.17: Schema esempio di una Repository

Il model Repository è stato utilizzato come pattern architetturale per la gestione del database.

2.3.3 Client Server

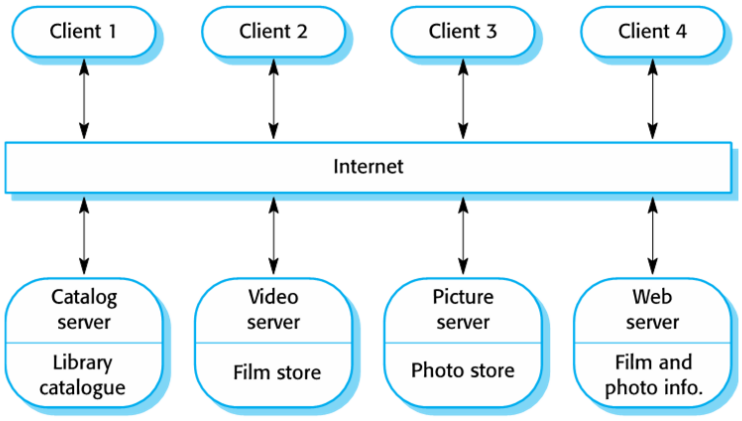

Figure 2.18: Schema Client Server

Il Server Client è stato utilizzato come pattern architetturale di base per la comunicazione col database.

2.4 Class diagram

Le classi nel sistema rispecchiano il pattern architetturale. Possiamo considerarle come due sistemi diversi del client e il server. Nel server ci sono due classi principali il Router, per la gestione delle richieste e il Database, che controlla i modelli. Questi ultimi nel client sono i model del MVC, mentre i controllers gestiscono l'esecuzione dell'applicazione. Non tutti i campi dei model nel Database sono presenti anche sul client, visto che sono utilizzati per facilitare il funzionamento del database e non contengono vere informazioni. Sia il server che il client contengono una classe Utils con solo metodi statici che è stata crata per semplificare dei metodi e rende più modulare il codice.

2.4.1 Server e Database

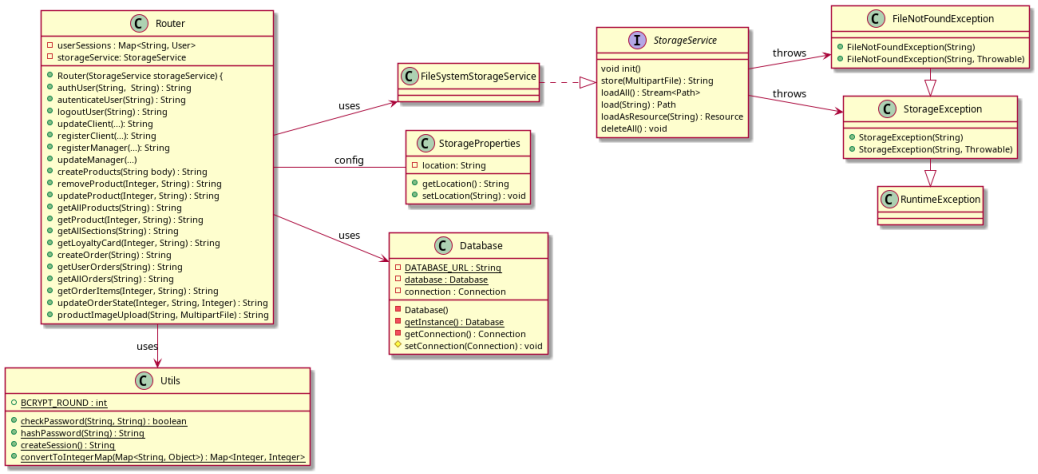

Figure 2.19: Class diagram server

Il Router ha il compito di gestire le chiamate del client e utilizza il database e le classi dei model per ricavare le informazioni. In aggiunta al database viene anche utilizzato lo FilesystemStorageService per salvare le immagini dei prodotti caricate dagli utenti.

2.4.2 Models

Server

Il database utilizza la classe Database per prendere una connessione al database fisico e i diversi models per ricavare le informazioni.

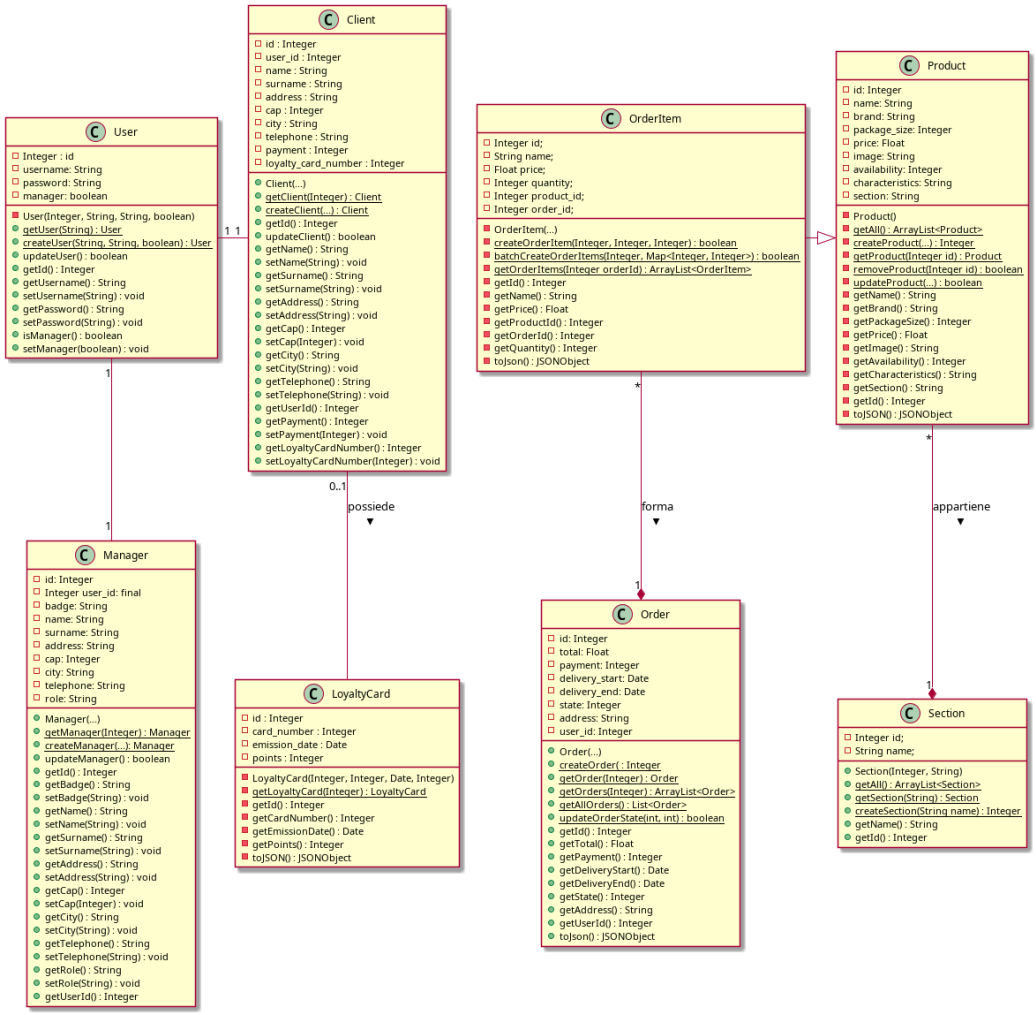

Figure 2.20: Class diagram database

Client

Invece nel client la Section non è più necessaria e non dovendo più rispettare lo schema del database possiamo creare una generalizzazione del User, Client e Manager.

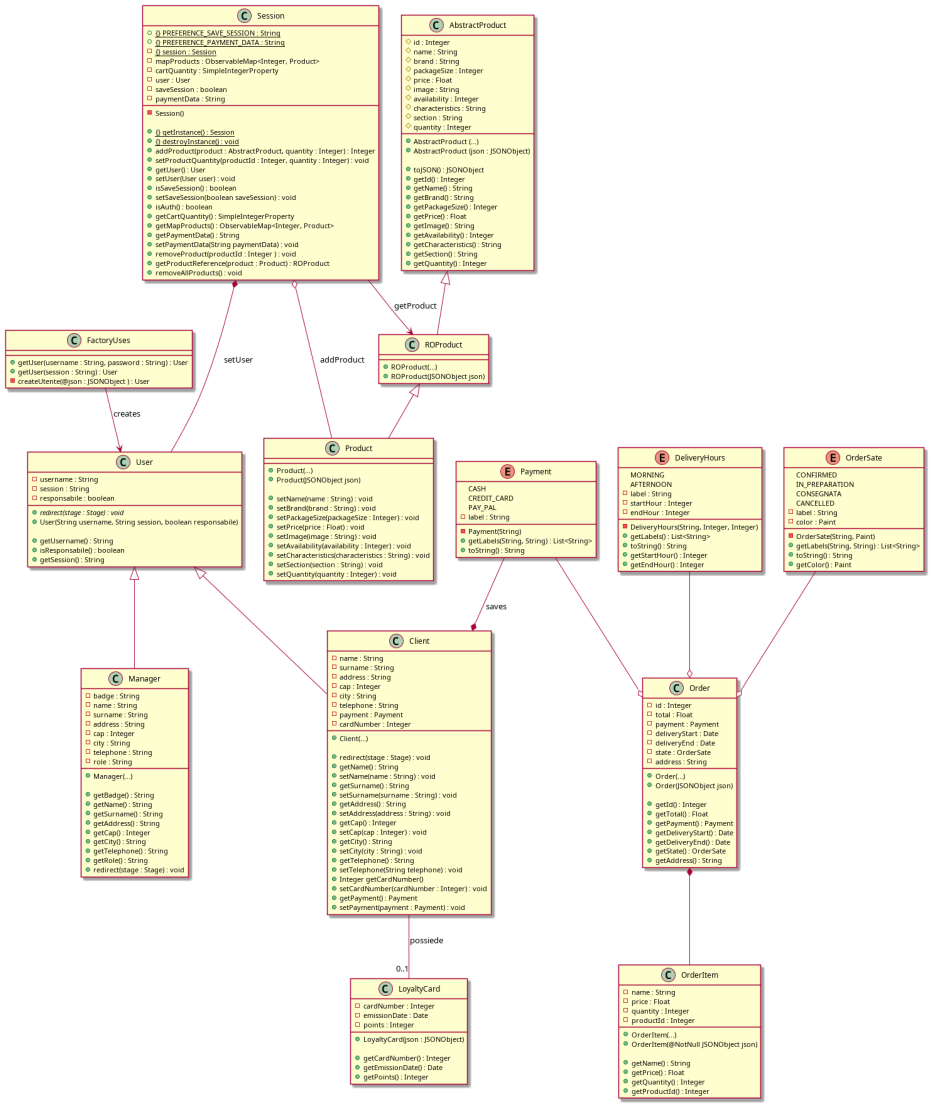

Figure 2.21: Class diagram client

2.4.3 Controllers

Autenticazione

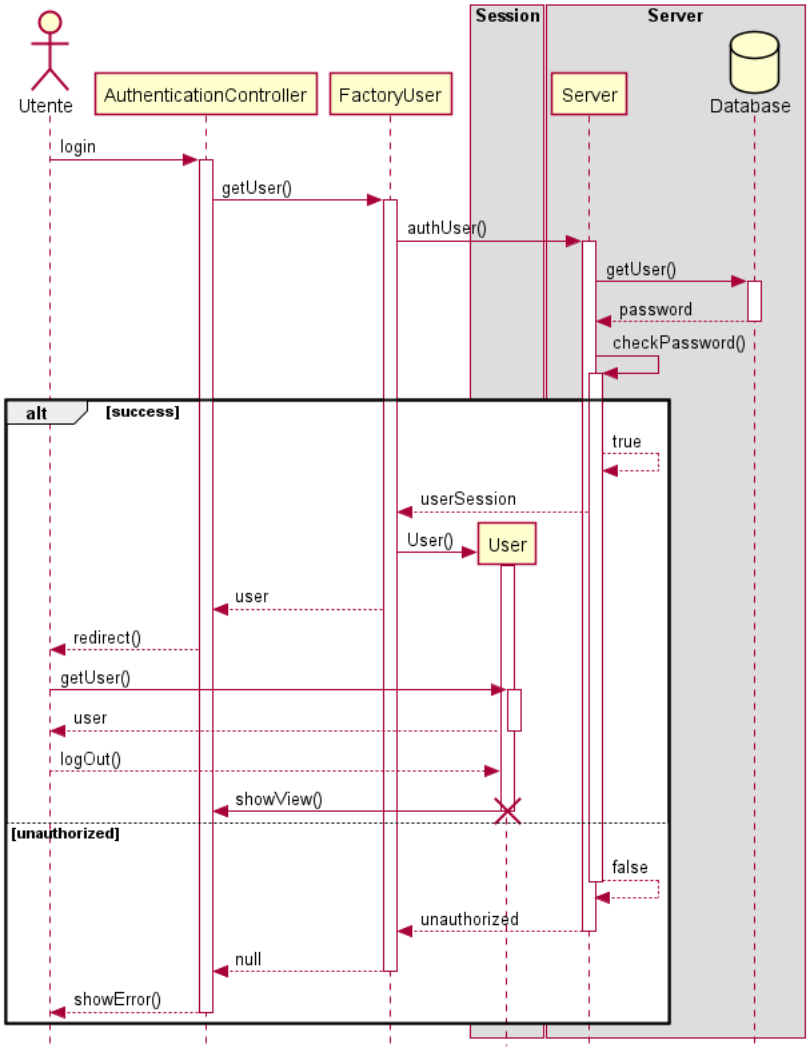

Figure 2.22: Sequence diagram Autenticazione

Gestione Profilo

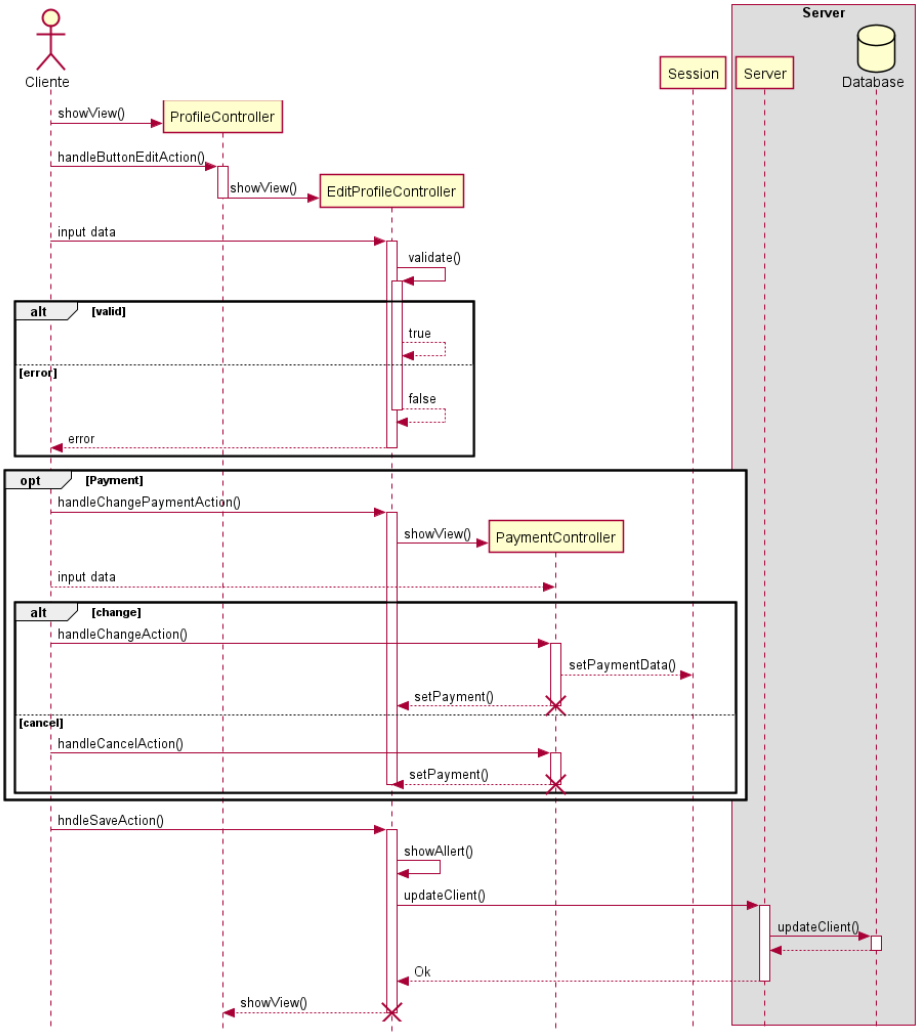

Figure 2.23: Sequence diagram Gestione Profilo

Registrazione

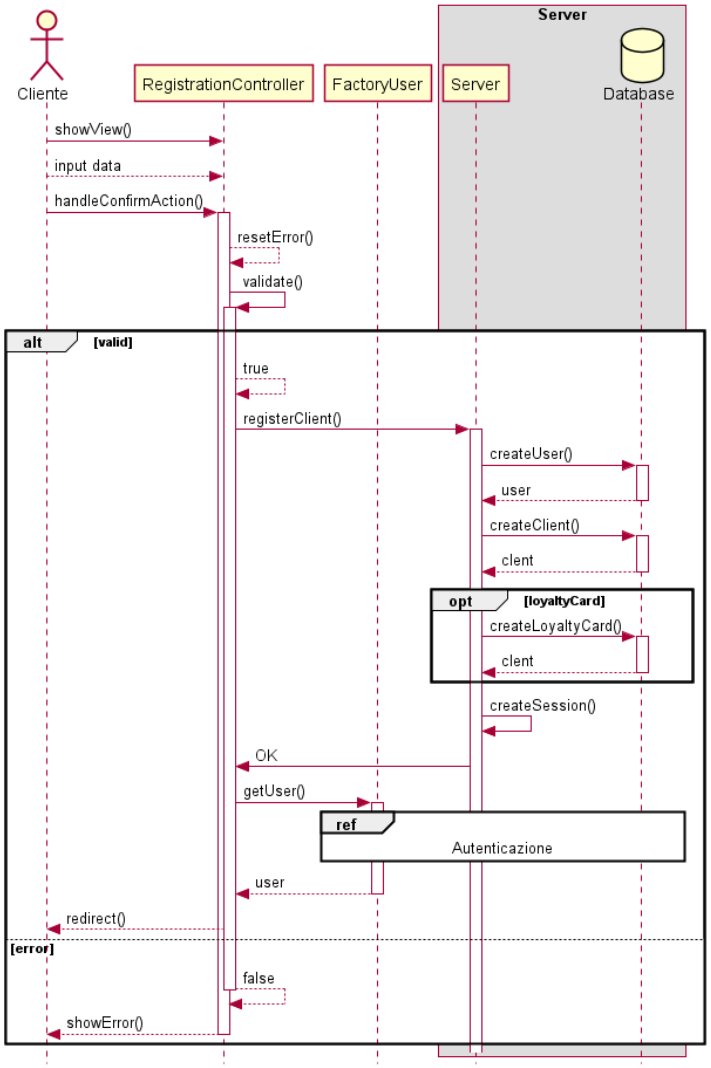

Figure 2.24: Sequence diagram Registrazione

Conferma Spesa

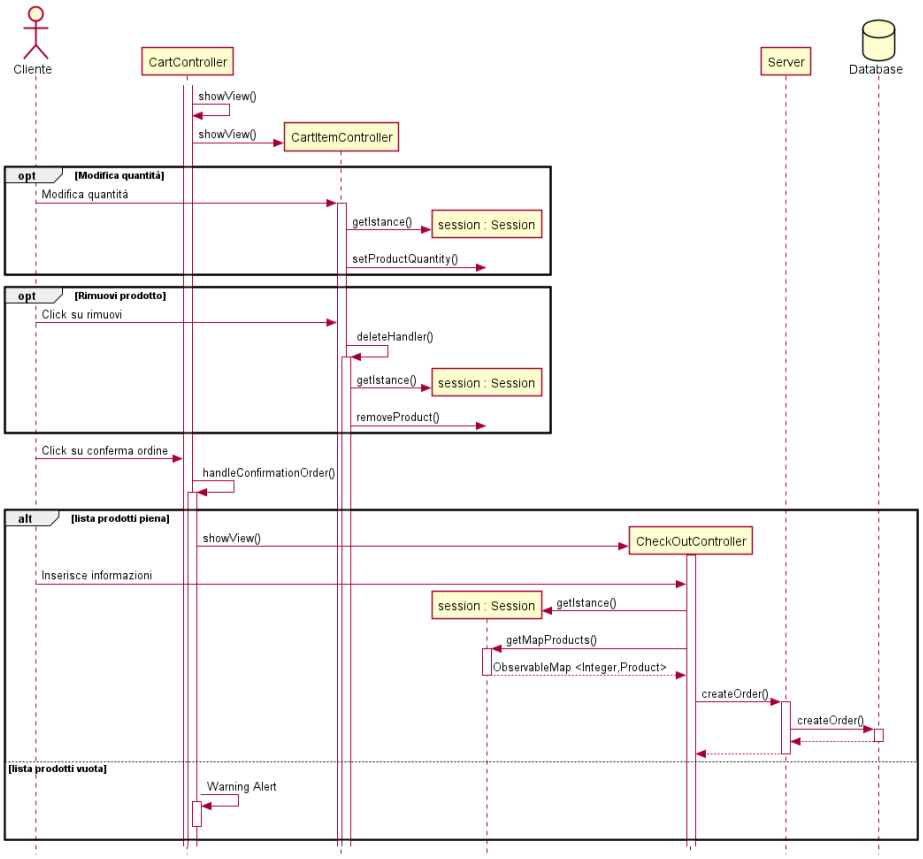

Figure 2.25: Sequence diagram Conferma Spesa

Controlla Spese

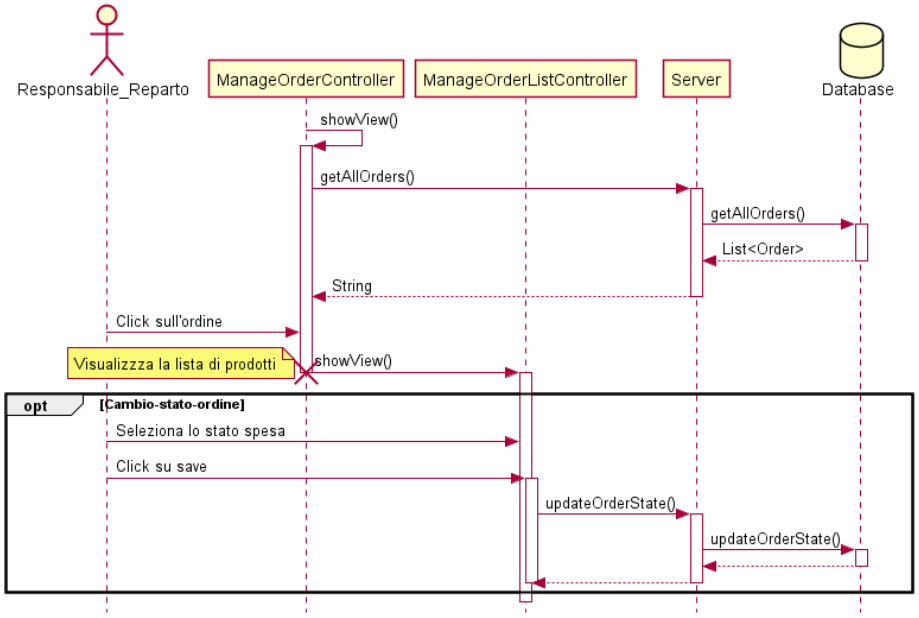

Figure 2.26: Sequence diagram Controlla Spese

Effettua Spesa

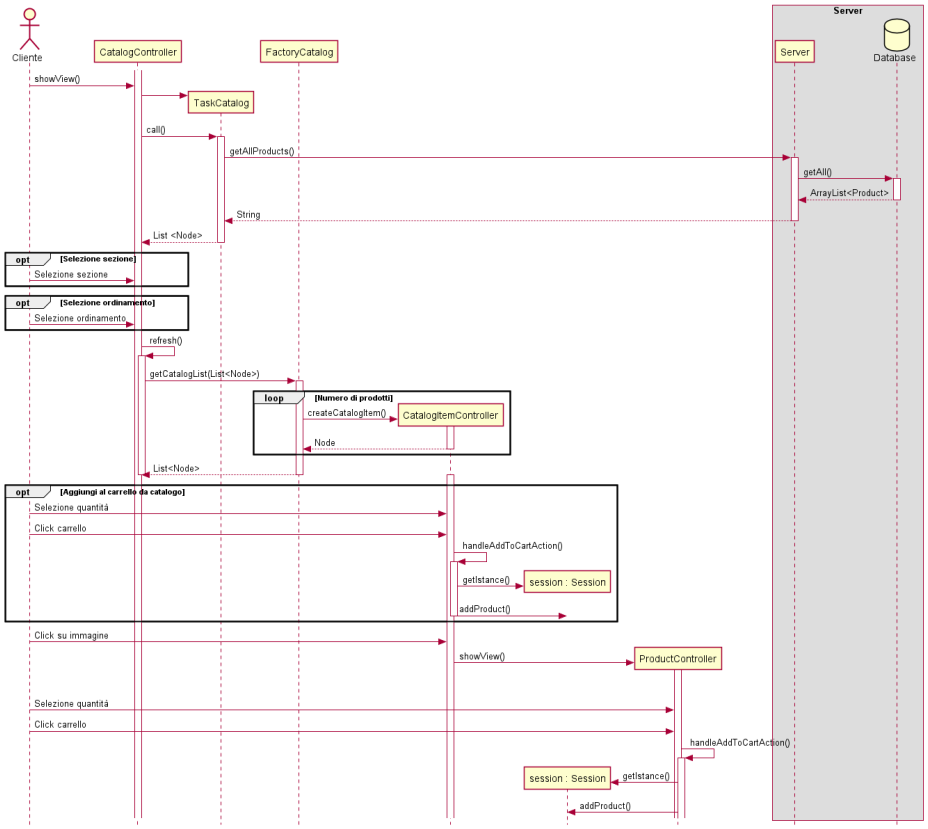

Figure 2.27: Sequence diagram Effettua Spesa

Gestione Prodotti

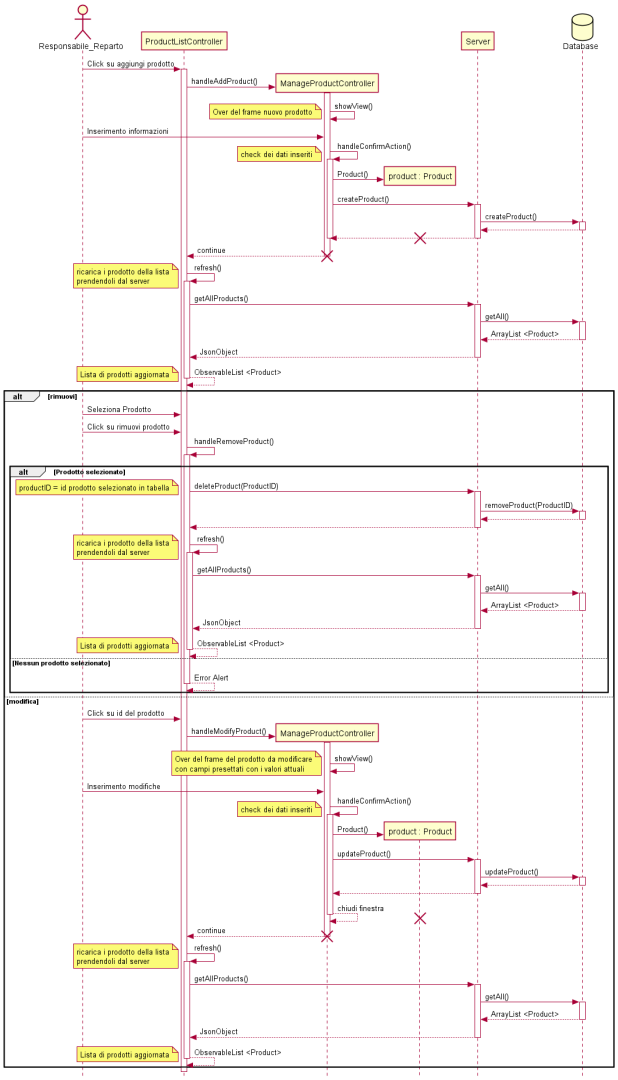

Figure 2.28: Sequence diagram Gestione Prodotti

Storico Spese

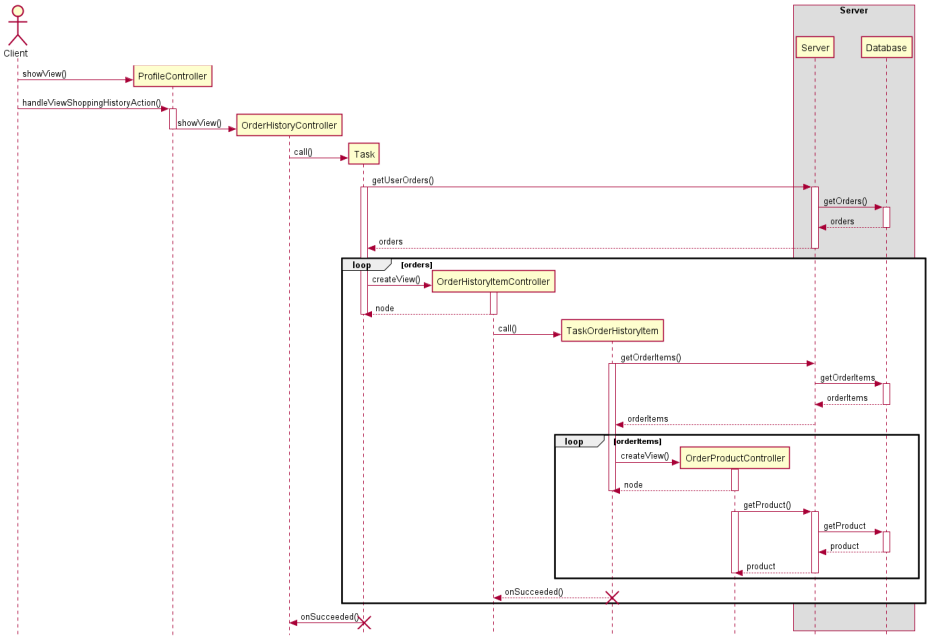

Figure 2.29: Sequence diagram Storico Spese

2.5 Design Patterns

Singleton

Il pattern Singleton viene utilizzato per garantire uno stato globale alle istanze del Database e della Sessione dell'utente. L'utilizzo di questo pattern è necessario in queste due classi ma si è cercato di evitarne l'utilizzo in altri contesti per evitare che la staticità della classe rendesse impossibile il mocking durante il test. In particolare, nella **Sessione** viene tenuta un istanza della classe Utente per gestire tutti i dati relativi all'utente autenticato; nel **Database** tutti i modelli utilizzano la stessa istanza per creare una connessione al database SQLite.

Factory Utente

Abbiamo creato una classe astratta Utente con un metodo `redirect()` astratto, le classi Client e Manager estendono Utente e ne implementano il metodo. Mentre la classe FactoryUtente permette di generare un'istanza delle due classi in base alle credenziali fornite. Quindi quando il metodo `redirect()` verrà chiamato si verrà indirizzati rispettivamente alla pagina del Cliente o del Manager.

DAO Observable Sessione

Molte views e task devono poter accedere ai dati del carrello e della sessione utente durante la fase di acquisto. Più di un thread accede nello stesso momento agli stessi dati, per questo è necessario utilizzare una DAO (Data Accesses Object) per evitare possibili TOCTOU (Time Of Check Time Of Use) e race condition (osservate durante lo sviluppo).

Prodotto

Per riuscire ad utilizzare il prodotto sia a catalogo prodotto sono state utilizzate 2 classi. Visto che è stato utilizzato il multi thread in modo intensivo si è avuto bisogno che i dati prodotti fossero leggibili da più thread ma utilizzati da uno solo. Questa caratteristica è stata soddisfatta dall'utilizzo di una classe astratta come base per la creazione delle classi ROProduct, che ne implementa solo la lettura e Prodotto che oltre ad implementare i metodi di lettura la estende con i metodi di scrittura.

2.5.1 Test suite

Durante la fase di programmazione, il nostro metodo scelto è stato il TDD, con il seguente metodo. Per iniziare il processo di aggiunta di un nuovo metodo, si crea un unit test della funzione da aggiungere, con semplici inputs e outputs e grazie alle funzioni della IDE si va a creare la vera funzione con i tipi delle variabili già presenti nel test. Questo purtroppo non è stato possibile per le funzioni dell'interfaccia grafica. Per questo motivo abbiamo raggiunto un code coverage dei test completo nel server ma scarso nel client. Il tutto è stato testato per ogni iterazione del codice grazie ad un sistema di CI (Continuous Integration) tramite la piattaforma di GitHub. Ulteriori test sono stati aggiunti alla fine della fase di sviluppo, per eventuali casi limite e bug incontrati nella beta.