

Progetto Ingegneria del software

# Byte di Dolcezza

19 ottobre 2023

A cura di:

Benzoni Tea matr. 1065725

Marmo Antonio matr. 1060572

Vaerini Andrea matr. 1067398

# OBIETTIVI



**1**

Gestione degli ordini di una pasticceria

**2**

Aumentare la clientela della  
pasticceria offrendo un servizio  
semplice e comodo

**3**

Aiutare il cliente nella scelta dei prodotti

**4**

Possibilità di sconti esclusivi

# Difficoltà

## Tempo

La principale difficoltà è stata l'organizzazione in termini temporali.

1. studio per la realizzazione del sistema
2. imparare utilizzo di programmi e tool
3. creazione interfacce grafiche



## **implementazione codice**

abbiamo riscontrato difficoltà anche nella fase di implementazione del codice, alcuni bug sono stati risolti ma altri no come:

1. inserimento dell'ordine personalizzato nel db.
2. fallimenti dei test del pacchetto controller



# Software configuration management

In questa fase di gestione della configurazione ci siamo serviti di GitHub, una piattaforma che offre molte funzionalità di gestione del codice sorgente e di collaborazione tra i team.



## Come abbiamo utilizzato?

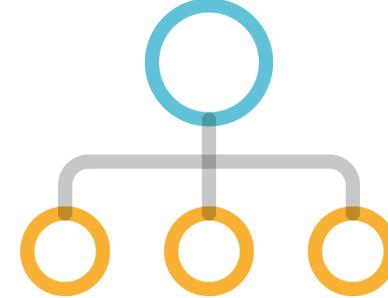
- Issue: per tenere traccia dei problemi e attività del progetto
- Pull request: richiesta inviata da un collaboratore per l'approvazione di modifiche apportate al progetto

## GitHub Desktop:

È un applicazione offerta da GitHub per la gestione delle repository

In questo caso è stata utilizzata per fare i commit e i push di documenti e file.

# Paradigmi e Tool



## **Modellazione**

**Programma e Tool:**  
StarUML, REBEL

**Utilizzo:**  
Diagrammi UML,  
generazione  
codice

## **Scrittura del codice**

**Programma:**  
Eclipse

**Utilizzo:**  
javascript,  
programmazione  
ad oggetti

## **Database**

**Programma:**  
XAMPP

**Utilizzo:**  
Database locale,  
SQL

## **Interfaccia**

**Libreria:**  
Javaswing

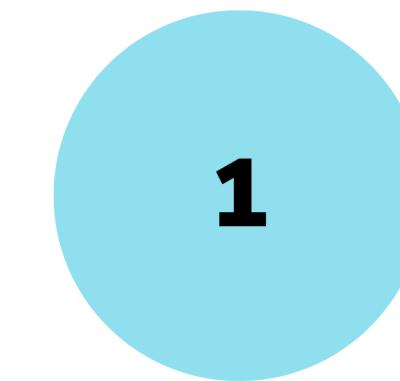
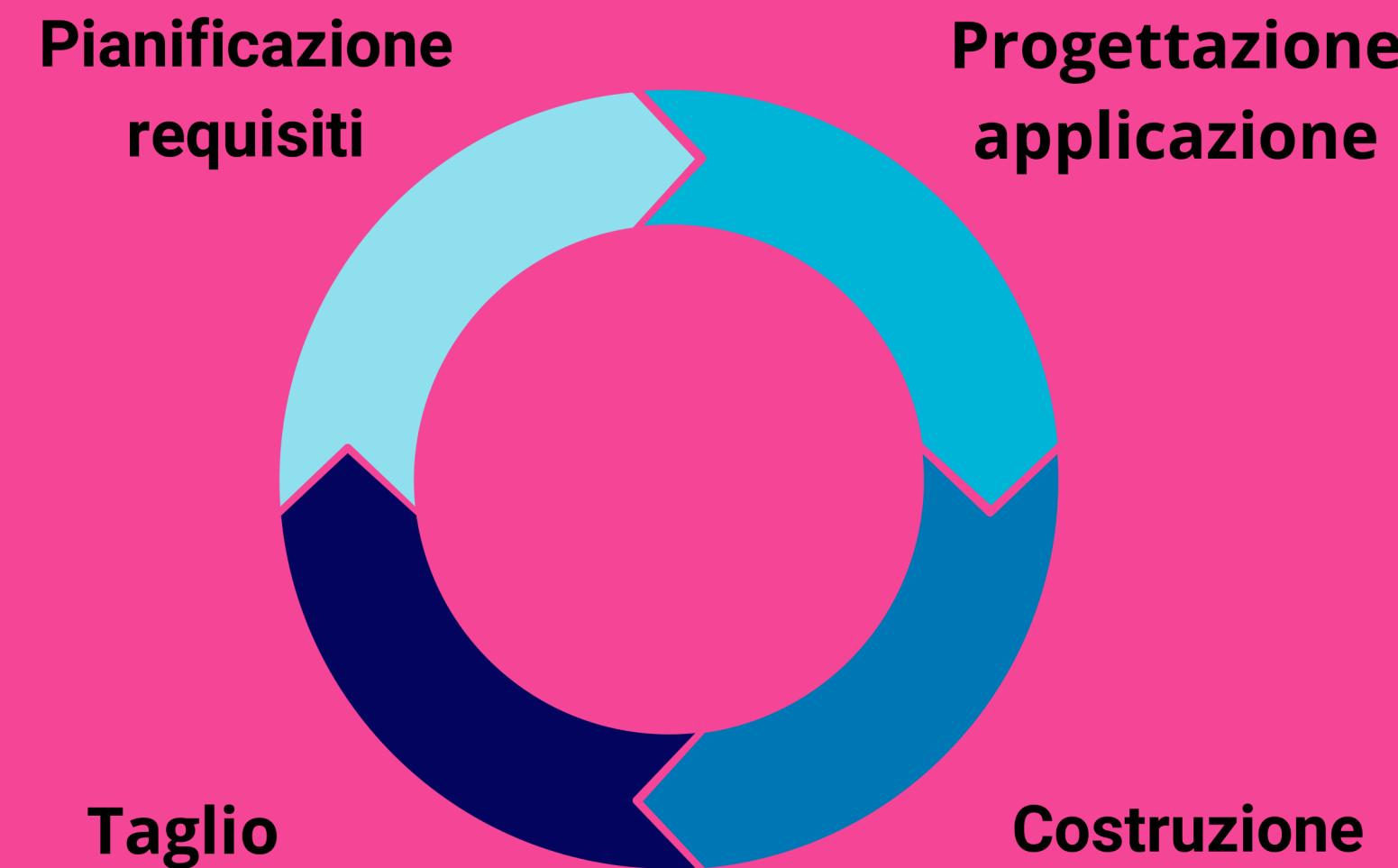
**Utilizzo:**  
Interfacce e  
componenti

## **Testing**

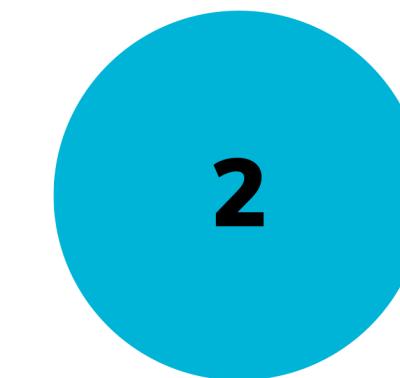
**Framework:**  
JUnit, Mockito

**Utilizzo:**  
implementazione  
dei test sul  
codice

# Ciclo di vita del software



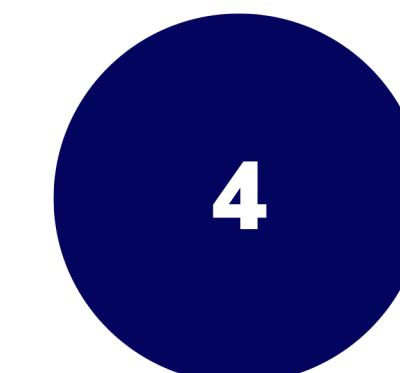
Raccolta dei requisiti iniziali tramite interviste o documentazioni...



Sviluppo di un prototipo per rispecchiare le esigenze dell'utente



Costruzione del sistema in base al prototipo, integrazione delle componenti



Collaudo finale e installazione del sistema, formazione utenti

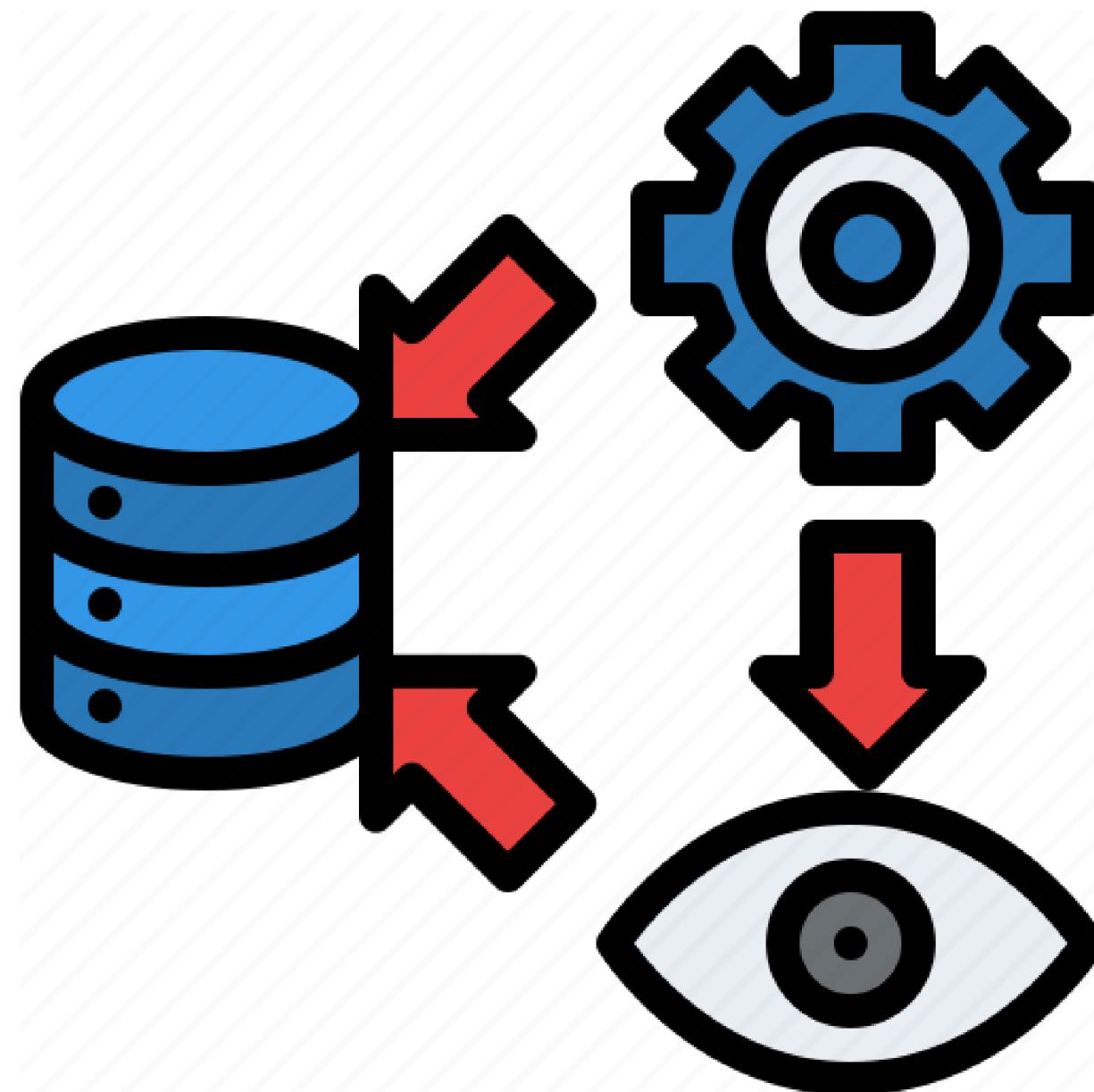
# Requisiti

Requisiti definiti tramite la priorità  
indicata da MoSCoW

FUNCTIONALI	Must Have	Should Have	Could Have	Won't Have
Reg Utente	Visualizza carrello	Cancella carrello	Visualizza Dati Abbonamento	
Catalogo	Modifica prodotto	Visualizza Utente	Riepilogo ordine standard	
Prenotazione	Sconto	Ingredienti Prodotti		
Abbonamento				

NON FUNZIONALI	Must Have	Should Have	Could Have	Won't Have
Sicurezza			Protezione dati utente	
Usabilità	Interfaccia semplice			
Riutilizzabilità		Catalogo, visualizza prodotti		
Correttezza	Effettuare ordini			

# Architettura



## Model View Controller

### Model:

Contiene i dati dell'applicazione e definisce come i dati possono essere manipolati e modificati.

### View:

Visualizza i dati provenienti dal model e consente all'utente di interagire con l'applicazione

### Controller:

Intermediario tra model e view.  
Gestisce le richieste dell'utente e le traduce in azioni.

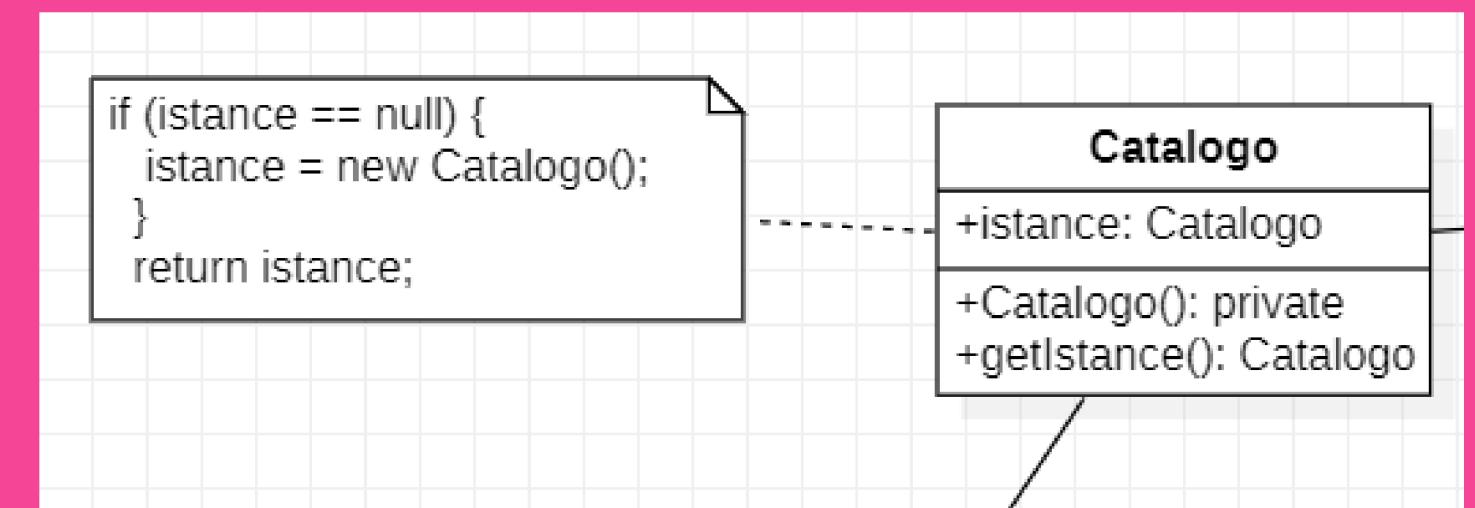
# Design Pattern

## Observer Pattern

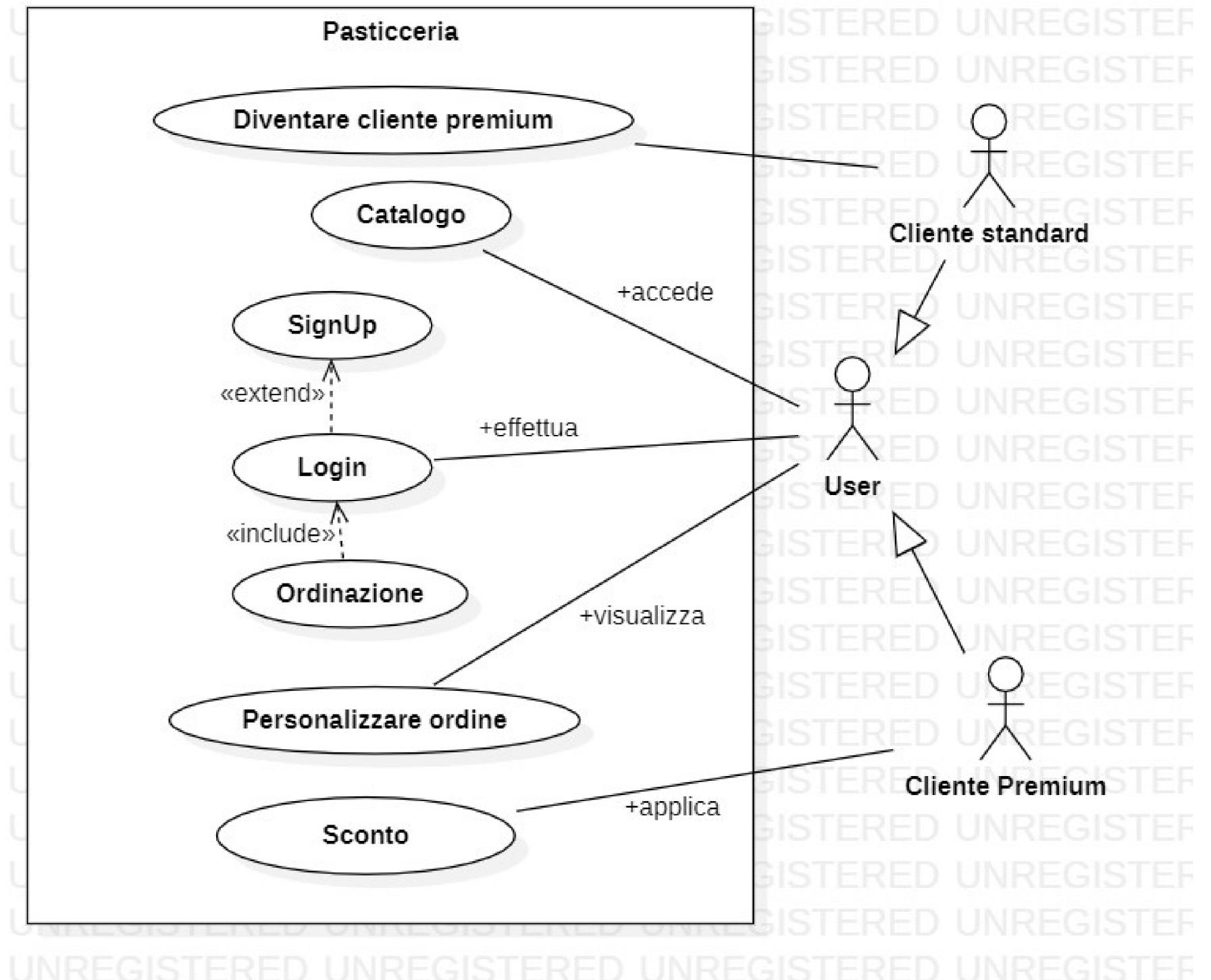
- Gestire le dipendenze tra oggetti in modo efficiente
- Soggetto: tiene traccia del suo stato e mantiene una lista di osservatori interessati
- Osservatori: oggetti interessati al cambiamento del Soggetto
- Per la nostra app è stato utilizzato per gestire gli eventi di azioni che si verificano tramite l'interazione con le componenti grafiche dell'interfaccia.

## Singleton Pattern

- Garantisce che una classe abbia una sola istanza
- Fornisce un punto di accesso globale a questa istanza
- Creata solo quando richiesta la prima volta
- Nella nostra app è stata utilizzata per il catalogo



# Modellazione

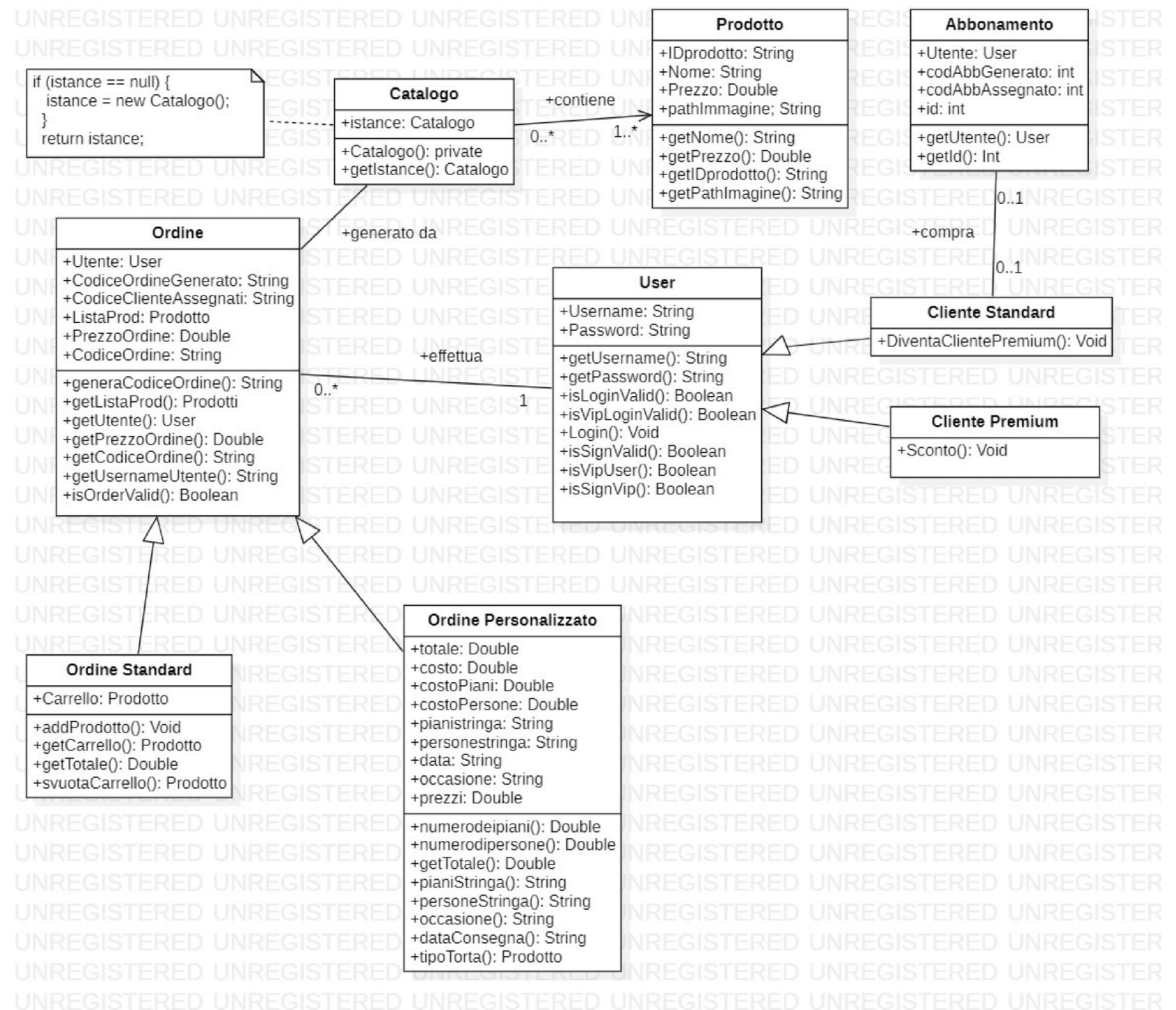


## Diagramma dei Casi d'uso

utilizzato per modellare i requisiti funzionali di un sistema e per visualizzare in modo chiaro come gli attori interagiscono con il sistema

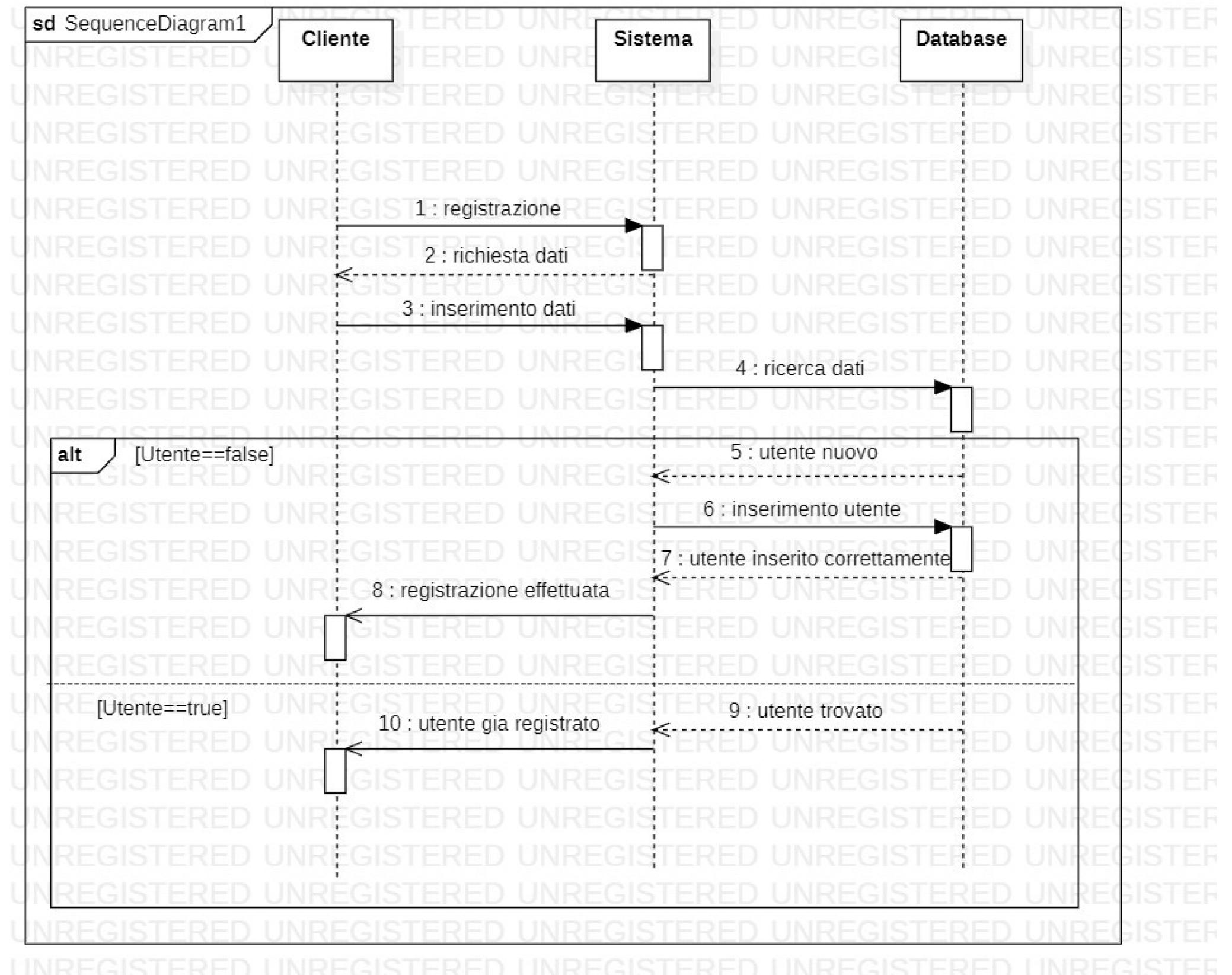
# Diagramma delle classi

modella la struttura di un sistema software rappresentando le classi, le loro relazioni, gli attributi e i metodi di ciascuna classe



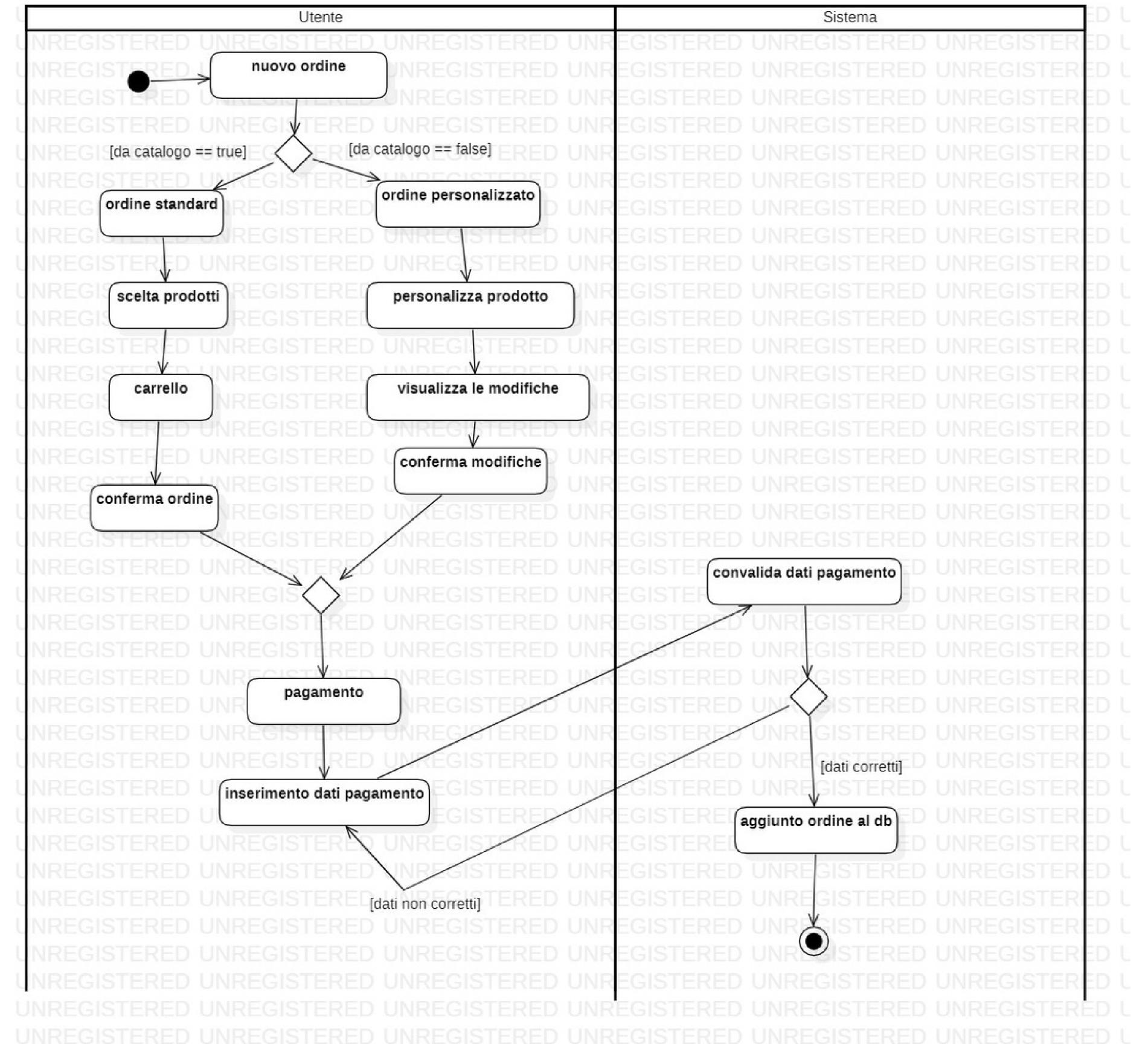
# Diagramma di Sequenza

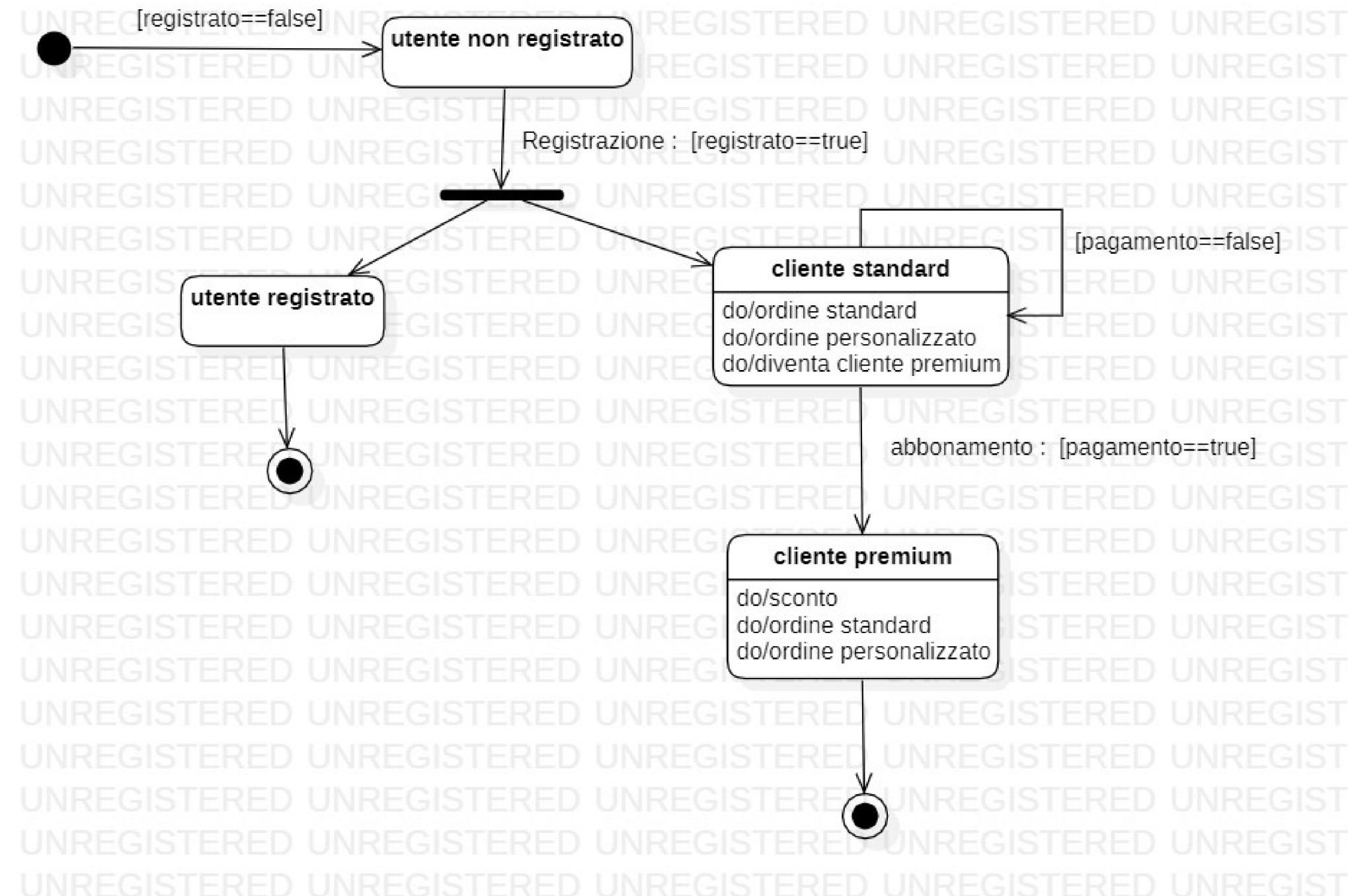
utilizzati per modellare le interazioni dinamiche tra gli oggetti di un sistema in una sequenza temporale.



# Diagramma delle Attività

utilizzati per modellare dei processi o qualsiasi flusso di lavoro che coinvolge una serie di azioni o decisioni

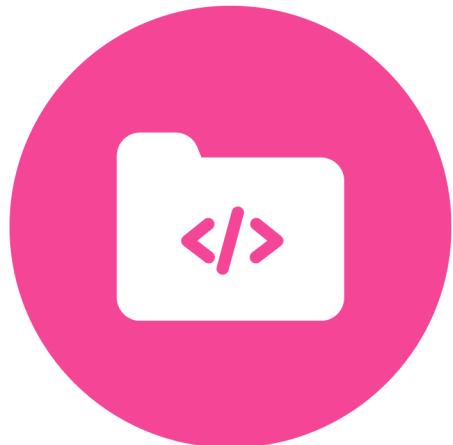




# Diagramma dello Stato

utilizzati per rappresentare il comportamento di un oggetto nel tempo, descrivendo i vari stati in cui può trovarsi e le transizioni tra di essi

# Implementazione



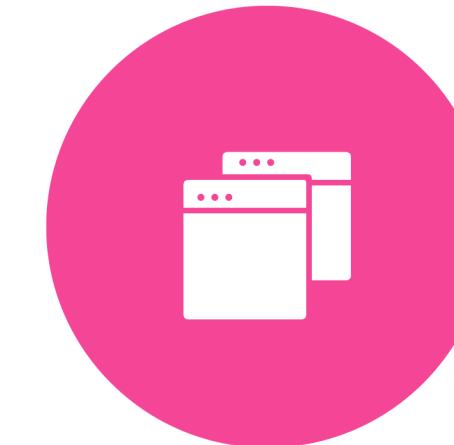
## Generazione del codice:

Dopo aver effettuato i diagrammi UML abbiamo utilizzato **Rebel** per tradurre il diagramma delle classi in codice Java.



## Implementazione e gestione Backend:

Implementazione dei metodi delle classi, settaggio per la gestione dei controllori e aggiornamento del DB.



## Gestione Frontend:

Implementazione delle opportune interfacce grafiche e collegamento della logica di Backend ad esse.

# Testing

- Test realizzati con JUnit e Mockito
- Test svolti su tutti i metodi delle classi
- La maggior parte è risultante funzionante
- Coverage ottenuto : 66,4%

```
└── testingModel
    ├── AbbonamentoTest.java
    ├── CatalogoTest.java
    ├── ClientePremiumTest.java
    ├── ClienteTest.java
    ├── HomeTest.java
    ├── LoginTest.java
    ├── OrdinePersonalizzatoTest.java
    ├── OrdineStandardTest.java
    ├── OrdineTest.java
    ├── PagamentoTest.java
    └── ProdottoTest.java
```

```
└── testingControl
    ├── AbbonamentoControllerTest.java
    ├── HomeControllerTest.java
    ├── IngredientiControllerTest.java
    ├── IntroControllerTest.java
    ├── LoginControllerTest.java
    ├── OrdinePersonalizzatoControllerTest.java
    ├── OrdineStandardControllerTest.java
    ├── PagamentoControllerTest.java
    ├── SignControllerTest.java
    ├── VisualizzaOrdineControllerTest.java
    └── VisualizzaOrdinePersonalizzatoControllerTest.java
```

```
└── testingView
    ├── AbbonamentoIFTest.java
    ├── HomeIFTest.java
    ├── IngredientiIFTest.java
    ├── IntroIFTest.java
    ├── LoginIFTest.java
    ├── OrdinePersonalizzatoIFTest.java
    ├── OrdineStandardIFTest.java
    ├── PagamentoIFTest.java
    ├── SignIFTest.java
    └── VisualizzaOrdineIFTest.java
```

# Demo

