

Ingegneria del Software

Corso di Laurea Magistrale in Ingegneria Informatica A.A. 2024/2025



Studenti:

Giorgia Grazia Mucciarella

Luca Camiolo

## Sommario

<b>DOCUMENTO DI VISIONE .....</b>	<b>3</b>
<b>SPECIFICHE SUPPLEMENTARI.....</b>	<b>4</b>
<b>GLOSSARIO .....</b>	<b>5</b>
<b>CASI D'USO .....</b>	<b>7</b>
<b>Elaborazione .....</b>	<b>19</b>
<b>Iterazione 1: .....</b>	<b>19</b>
<b>Iterazione 2 .....</b>	<b>31</b>
<b>Iterazione 3: .....</b>	<b>46</b>
<b>TESTING.....</b>	<b>52</b>
<b>Istruzioni di Configurazione ed Esecuzione di MyResto .....</b>	<b>59</b>

# DOCUMENTO DI VISIONE

## 1 Introduzione

Il sistema di gestione del ristorante, chiamato **MyResto**, è progettato per ottimizzare e semplificare le operazioni quotidiane di un ristorante, migliorando l'efficienza operativa interna e l'esperienza del cliente. L'applicazione gestirà prenotazioni, ordini, pianificazione del personale, sconti e feedback dei clienti

## 2. Posizionamento

### 2.1 Opportunità di business

**MyResto** sostituisce i metodi tradizionali di gestione manuale dei ristoranti, offrendo un sistema integrato che automatizza processi come la gestione degli ordini e l'analisi delle preferenze dei clienti. Ciò consente di ridurre errori, risparmiare tempo e aumentare la soddisfazione dei clienti.

### 2.2 Formulazione del problema

Molti ristoranti si affidano a strumenti manuali o a sistemi scollegati per gestire le varie attività operative, causando:

- Gestione inefficiente delle prenotazioni e degli ordini.
- Mancanza di strumenti per analizzare le preferenze dei clienti e le performance del ristorante.
- Pianificazione manuale del personale, spesso poco ottimizzata.

### 2.3 Formulazione della posizione del prodotto

Il software è rivolto ai ristoratori che desiderano digitalizzare e ottimizzare la propria attività.

## 3 Descrizione delle parti interessate

Gli utenti necessitano di un sistema che soddisfi i seguenti obiettivi:

- **Amministratore:** Gestire il menu, il personale, generare report, configurare promozioni e programmi fedeltà.
- **Cliente:** Effettuare prenotazioni, lasciare feedback e partecipare a programmi fedeltà.

## 4 Riepilogo delle caratteristiche del sistema

- Gestione delle prenotazioni (creazione, modifica, annullamento).
- Gestione degli ordini (inserimento, modifica, cancellazione).
- Pianificazione del personale.
- Generazione di report sulle performance e preferenze dei clienti.
- Programma fedeltà per clienti.
- Form per il feedback dei clienti.

## **SPECIFICHE SUPPLEMENTARI**

### **1. Introduzione**

Il presente documento descrive le specifiche supplementari del sistema **MyResto**, una web-app per la gestione di prenotazioni, ordinazioni e pagamenti in un contesto ristorativo. Queste specifiche completano i requisiti funzionali, dettagliando aspetti tecnici, qualitativi, di sicurezza e ambientali del sistema.

### **2. Funzionalità**

Il sistema supporterà le seguenti funzionalità non direttamente mappate sui casi d'uso:

- Monitoraggio delle prestazioni in tempo reale.
- Accesso differenziato in base ai ruoli (cliente, cameriere, amministratore).
- Notifiche automatiche (email) per conferme e aggiornamenti delle prenotazioni.

### **3. Regole inseribili**

- L'amministratore potrà definire orari di apertura/chiusura, regole di prenotazione (es. massimo 10 persone), disponibilità dei tavoli.
- Sarà possibile configurare menu del giorno o eventi speciali.

### **4. Sicurezza**

- Accesso con ruoli e privilegi.

### **5. Usabilità**

#### **5.1 Fattori umani**

- Interfaccia progettata per utenti non esperti.
- Design compatibile con dispositivi mobili.

### **6. Affidabilità**

#### **6.1 Ripristinabilità**

- Backup automatico giornaliero.

#### **6.2 Prestazioni**

- Tempi di risposta inferiori a 2 secondi per il 95% delle operazioni.
- Sistema capace di gestire almeno 50 richieste concorrenti

## 7. Sostenibilità

### 7.1 Adattabilità

- Supporto multilingua configurabile.

### 7.2 Configurabilità

- Amministratori potranno modificare menu, prezzi, orari, disponibilità tavoli via interfaccia grafica.

## 8. Vincoli di implementazione

- Linguaggio di programmazione: Java.
- Ambiente di Sviluppo Integrato (IDE): IntelliJ IDEA.
- Strumento di Build Automation: Apache Maven.
- Framework di Test: JUnit 5.
- Framework di Mocking: Mockito.
- Non è stato utilizzato un database relazionale ma i dati sono salvati in file di testo.

## 9. Altri vincoli di progettazione e implementativi

- Struttura modulare per facilitare manutenzione e test.
- Separazione chiara tra logica di business, presentazione e persistenza dei dati.

## 10. Norme e standard

- Codice conforme agli standard di codifica Java.

## GLOSSARIO

Di seguito sono riportati i termini significativi utilizzati nel progetto. Per ciascun termine sono fornite definizioni, formati e altre informazioni utili alla corretta interpretazione e implementazione.

Termine	Definizione	Formato	Relazioni	Valori ammessi	Regole di validazione	Sinonimi
Cliente	Utente che effettua un ordine nel ristorante	Oggetto Java	Legato a Ordine	Nome, email, telefono validi	Email obbligatoria, telefono valido	Utente finale

Ordine	Insieme di piatti richiesti da un cliente	Oggetto Java	Contiene Piatti	Stato: In attesa, In preparazione , Consegnato	Stato obbligatorio , almeno un piatto incluso	Comanda
Piatto	Voce del menu ordinabile	Oggetto Java	Relazione con menù, Ordine	Nome, prezzo, descrizione	Prezzo > 0, nome non vuoto	Portata
Menù	Insieme strutturato di piatti offerti dal ristorante	Oggetto Java	Composto da più piatti	Lista di piatti	Deve avere almeno un piatto	Carta
Tavolo	Posizione fisica all'interno del ristorante	Int	Legato a Ordine	Numeri interi positivi	<p>ID Tavolo: Deve essere un valore unico e positivo.</p> <p>Capacità: Deve essere un numero intero positivo maggiore di zero.</p> <p>Posizione: Deve essere un valore predefinito o descrittivo della sua ubicazione.</p> <p>Stato: Deve corrispondere a uno dei valori ammessi (es. "libero", "occupato",</p>	Postazione

					"prenotato").	
Amministratore	Utente con privilegi gestionali	Oggetto Java	Può creare/modificare tutto	Email, password	Password sicura, email univoca	Gestore, Proprietario
Pagamento	Informazioni relative alla transazione dell'ordine	Oggetto Java	Associato a Ordine	Carta, Contanti	Metodo obbligatorio, importo >= totale dell'ordine	Transazione
Riga ordine	Rappresenta un singolo piatto all'interno di un ordine	Oggetto Java	Associato a Ordine	ID: Numerico intero (generato automaticamente). Quantità: Numerico intero. Prezzo Unitario: float. Note: Testo libero (opzionale).	ID: Deve essere un valore unico e positivo. Quantità: Deve essere un numero intero positivo (almeno 1). Prezzo Unitario: Deve essere un numero positivo e corrispondere al prezzo del Piatto associato al momento dell'aggiunta.	Dettaglio Ordine

## CASI D'USO

### 1. Requisiti

**MyResto** è un software che ottimizzi la gestione delle attività quotidiane del ristorante, migliorando l'efficienza operativa e l'esperienza del cliente. Il software deve rappresentare una soluzione integrata

che copra tutti gli aspetti della gestione, dalla prenotazione dei tavoli alla gestione degli ordini, dall'analisi delle vendite alla pianificazione del personale. In particolare:

✓ **Gestione delle prenotazioni:**

- Il sistema deve permettere ai clienti di prenotare un tavolo specificando data, orario, numero di persone.
- Deve verificare in tempo reale la disponibilità dei tavoli e assegnare automaticamente il tavolo più adatto.
- In caso di indisponibilità, deve suggerire alternative (orari diversi, lista d'attesa).

✓ **Gestione degli ordini:**

- I camerieri devono poter registrare gli ordini dei clienti selezionando i piatti dal menu, con eventuali modifiche (es. allergie, preferenze alimentari).
- Il sistema deve calcolare automaticamente il conto
- Deve essere possibile modificare o annullare un ordine prima che venga preparato.

✓ **Gestione del menu:**

- L'amministratore deve poter aggiungere, modificare o eliminare piatti, specificando nome, descrizione, prezzo.

✓ **Modifica ordine già effettuato:**

- Le modifiche devono essere notificate.
- Devono essere gestite variazioni di prezzo,mostrando la nuova differenza al cliente.

✓ **Pianificazione del personale:**

- L'amministratore deve poter assegnare i turni al personale in base alle prenotazioni e all'affluenza prevista.

✓ **Generazione Report:**

- Il sistema deve generare report sulle vendite (piatti più ordinati, ricavi giornalieri/settimanali).

✓ **Feedback dei clienti:**

- I clienti devono poter lasciare valutazioni e commenti dopo il pasto.
- Il sistema deve analizzare i feedback per identificare tendenze e aree di miglioramento.

✓ **Programma fedeltà:**

- I clienti abituali devono poter accumulare punti per ogni acquisto.
- I Clienti devono poter usufruire dei premi che desiderano,spendendo i punti necessari riscattarli.

## **1. Obiettivi e casi d'uso**

Analizzando i requisiti riportati nel paragrafo precedente, sono stati individuati l'attore principale a cui è destinato il sistema e gli obiettivi che egli intende portare a termine; da queste informazioni infine sono stati ricavati i casi d'uso principali



Cliente	Prenotare, modificare, annullare una prenotazione	UC1: Gestione Prenotazione (CRUD)
Cliente	Effettuare un ordine in merito a piatti e bevande desiderate	UC2: Gestione ordine (CRUD)
Amministratore	Gestione del pagamento di ogni singolo tavolo	UC3: Gestione Pagamento
Amministratore	Aggiungere, modificare, rimuovere piatti dal menu	UC4: Gestione menu (CRUD)
Cliente	Permettere ad un cliente che aveva già ordinato di effettuare modifiche prima della preparazione.	UC5: Modifica ordine già effettuato
Amministratore	Pianificare i turni del personale	UC6: Pianificazione del personale
Amministratore	Analizzare prestazioni	UC7: Generazione Report
Cliente	Partecipare al programma fedeltà	UC8: Fidelity card
Cliente	Fornire feedback sull'esperienza	UC9: Feedback clienti
Amministratore	Effettuare l'organizzazione dei tavoli	UC10: Gestione Tavoli (CRUD)

## 2. Modello dei casi d'uso

UC1: Gestione prenotazione (CRUD)

<b>Nome del caso d'uso</b>	UC1: Gestione prenotazione (CRUD)
<b>Portata</b>	Sistema di prenotazione tavoli
<b>Livello</b>	Obiettivo utente
<b>Attore primario</b>	Cliente
<b>Parti interessate e interessi</b>	<ul style="list-style-type: none"> <li>● Cliente: vuole effettuare facilmente le prenotazioni.</li> </ul>
<b>Pre-condizioni</b>	Sistema prenotazioni attivo
<b>Garanzia di successo (o post-condizioni)</b>	Prenotazione creata, modificata o annullata correttamente. Assegnazione del tavolo dopo la creazione della prenotazione
<b>Scenario principale di successo</b>	<ol style="list-style-type: none"> <li>1. Il Cliente chiede al sistema l'inserimento di una nuova prenotazione.</li> <li>2. Il Sistema chiede al Cliente di inserire le informazioni relative alla prenotazione.</li> <li>3. Il Cliente Inserisce data, orario, numero Persone.</li> <li>4. Il sistema verifica la disponibilità dei tavoli in base all'orario richiesto dall'utente e presenta una lista dei tavoli liberi.</li> <li>5. Il Sistema chiede al Cliente di inserire le informazioni per la prenotazione</li> </ol>

	6. Il Cliente fornisce le informazioni ed effettua la prenotazione. 7. Il Sistema salva la prenotazione.
Estensioni (o scenari alternativi)	<ul style="list-style-type: none"> <li>● In qualsiasi momento il sistema potrebbe arrestarsi.             <ol style="list-style-type: none"> <li>1. Il titolare riavvia il software e ripristina lo stato precedente del sistema.</li> <li>2. Il sistema ricrea lo stato precedente.                 <ol style="list-style-type: none"> <li>1.1 Il cliente chiede al sistema la <b>modifica</b> di una prenotazione.                     <ol style="list-style-type: none"> <li>1. Il sistema chiede al Cliente di aggiornare le nuove informazioni per modificare la prenotazione</li> <li>2. Il Cliente inserisce nome, cognome, data, orario, numeroPersone.</li> <li>3. Il sistema verifica la disponibilità dei tavoli in base all'orario richiesto dall'utente. Se disponibile questo lo assegna in maniera automatica fornendo il relativo numero.</li> <li>4. Il Sistema stampa il riepilogo della prenotazione modificata.</li> <li>5. Il Cliente conferma.</li> <li>6. La prenotazione viene modificata.</li> </ol> </li> <li>1.2 Il titolare chiede al sistema di <b>ricercare</b> una prenotazione.                     <ol style="list-style-type: none"> <li>1. Il sistema chiede al cliente di inserire le informazioni.</li> <li>2. Il cliente inserisce le informazioni.</li> <li>3. Il sistema mostra l'elenco delle prenotazioni che soddisfano i criteri di ricerca.</li> </ol> </li> <li>1.3 Il cliente chiede al sistema di <b>eliminare</b> una prenotazione                     <ol style="list-style-type: none"> <li>1. Il sistema chiede al Cliente di inserire il codice della prenotazione da eliminare.</li> <li>2. Il Cliente inserisce i dati.</li> <li>3. Il Sistema mostra la prenotazione e chiede al Cliente di confermare l'eliminazione.</li> <li>4. Il Cliente conferma.</li> <li>5. Il Sistema elimina la prenotazione dal sistema.</li> </ol> </li> </ol> </li> <li>3.1. Il sistema informa che l'inserimento dei dati è incompleto.</li> <li>7.1. Prenotazione già esistente nel medesimo giorno, orario e con ugual cognome.</li> </ol> </li> </ul>
Requisiti speciali	Blocco prenotazioni multiple stesso orario.
Elenco delle varianti tecnologiche e dei dati	Non specificati.

Frequenza di ripetizione	Quotidiana.
Problemi aperti	Gestione prenotazioni last-minute

#### UC2: Gestione ordine (CRUD)

Nome del caso d'uso	UC2: Gestione ordine
Portata	Sistema ordinazione digitale
Livello	Obiettivo utente
Attore primario	Cliente
Parti interessate e interessi	Cliente: vuole ordinare velocemente i piatti desiderati.
Pre-condizioni	Tavolo assegnato all'utente.
Garanzia di successo (o post-condizioni)	Ordine registrato e confermato.
Scenario principale di successo	<ol style="list-style-type: none"> <li>1. Il cliente accede al menu.</li> <li>2. Seleziona i piatti.</li> <li>3. Per ciascun piatto selezionato e aggiunto alla riga dell'ordine, il cliente può decidere di: <ul style="list-style-type: none"> <li>o Modificare gli ingredienti nel piatto.</li> <li>o Specificare una nota.</li> </ul> <i>Il passo 2 e 3 viene ripetuto finché serve.</i> </li> <li>4. Il Sistema chiede al cliente se vuole confermare la creazione dell'ordine.</li> <li>5. Il cliente conferma l'ordine.</li> <li>6. Il sistema salva l'ordine..</li> </ol>
Estensioni (o scenari alternativi)	<ul style="list-style-type: none"> <li>● In qualsiasi momento il sistema potrebbe arrestarsi. <ol style="list-style-type: none"> <li>1. Il titolare riavvia il software e ripristina lo stato precedente del sistema.</li> <li>2. Il sistema ricrea lo stato precedente. <ol style="list-style-type: none"> <li>2.1. Il piatto non è disponibile. <ol style="list-style-type: none"> <li>1. Viene comunicato al cliente.</li> </ol> </li> <li>3.1. Ingrediente extra non disponibile.</li> <li>3.2. Modifica non possibile da applicare al piatto.</li> </ol> </li> </ol> </li> </ul>
Requisiti speciali	Tempi di risposta rapidi.
Elenco delle varianti tecnologiche e dei dati	Menu con foto.
Frequenza di ripetizione	Quotidiana.
Problemi aperti	Gestire allergie segnalate nei piatti.

#### UC3: Gestione Pagamento

Nome del caso d'uso	UC3: Gestione Conto
Portata	Sistema pagamento
Livello	Obiettivo operativo
Attore primario	Amministratore

<b>Parti interessate e interessi</b>	Cliente: vuole effettuare il pagamento. Amministratore: gestire in maniera corretta la transazione
<b>Pre-condizioni</b>	<ul style="list-style-type: none"> <li>● Il cliente abbia effettuato l'ordinazione e sia stato servito.</li> <li>● Se il cliente ha riscattato un premio dalla fidelity card, il sistema li applica.</li> </ul>
<b>Garanzia di successo (o post-condizioni)</b>	Pagamento registrato nel sistema con annessa ricevuta
<b>Scenario principale di successo</b>	<ol style="list-style-type: none"> <li>1. L'amministratore accede alla sezione Pagamenti.</li> <li>2. L'Amministratore seleziona il tavolo.</li> <li>3. Il sistema recupera l'ordine effettuato.</li> <li>4. Il sistema calcola il totale che il cliente deve pagare.</li> <li>5. Il cliente fornisce i dati all'amministratore.</li> <li>6. L'amministratore digita il nome, cognome e email del cliente e lo associa.</li> <li>7. Il sistema chiede all'amministratore di confermare.</li> <li>8. L'amministratore conferma, registrando il pagamento effettuato dal cliente e aggiornando il suo saldo punti.</li> </ol>
<b>Estensioni (o scenari alternativi)</b>	<ul style="list-style-type: none"> <li>● In qualsiasi momento il sistema potrebbe arrestarsi. <ol style="list-style-type: none"> <li>1. Il titolare riavvia il software e ripristina lo stato precedente del sistema.</li> <li>2. Il sistema ricrea lo stato precedente.</li> </ol> </li> <li>5.1. Il cliente non desidera registrarsi e continua il pagamento normalmente.</li> </ul>
<b>Requisiti speciali</b>	Supporto multi-pagamento.
<b>Elenco delle varianti tecnologiche e dei dati</b>	Non specificati.
<b>Frequenza di ripetizione</b>	Quotidiana.
<b>Problemi aperti</b>	Gestione dello split del conto al momento non supportata, ma identificata come possibile miglioramento per ottimizzare l'esperienza.

#### UC4: Gestione Menu (CRUD)

<b>Nome del caso d'uso</b>	UC4: Gestione menu
<b>Portata</b>	Sistema di Gestione del Ristorante
<b>Livello</b>	Obiettivo utente
<b>Attore primario</b>	Amministratore
<b>Parti interessate e interessi</b>	<ul style="list-style-type: none"> <li>● Amministratore: vuole mantenere il menu aggiornato per offrire piatti disponibili e aggiornare prezzi o descrizioni.</li> <li>● Cliente: desidera consultare un menu aggiornato e corretto per poter scegliere i piatti senza errori.</li> </ul>
<b>Pre-condizioni</b>	<ul style="list-style-type: none"> <li>● L'amministratore è autenticato nel sistema.</li> </ul>
<b>Garanzia di successo (o post-condizioni)</b>	<ul style="list-style-type: none"> <li>● Il menù viene aggiornato correttamente.</li> </ul>

Scenario principale di successo	<ol style="list-style-type: none"> <li>1. L'amministratore accede alla sezione "Gestione Menu".</li> <li>2. L'amministratore chiede al sistema l'inserimento di un nuovo piatto.</li> <li>3. Il sistema chiede le informazioni necessarie.</li> <li>4. L'amministratore inserisce nome, descrizione, prezzo, ingredienti.</li> </ol> <p><i>Il passo da 2 a 4 viene ripetuto finché serve.</i></p> <ol style="list-style-type: none"> <li>5. Il sistema chiede all'amministratore di confermare gli inserimenti che sono stati fatti.</li> <li>6. L'amministratore Conferma l'operazione</li> <li>7. Il sistema salva e aggiorna il menu.</li> </ol>
Estensioni (o scenari alternativi)	<ul style="list-style-type: none"> <li>• 1. Il titolare riavvia il software e ripristina lo stato precedente del sistema.</li> <li>• 2. Il sistema ricrea lo stato precedente.</li> </ul> <p>4.1 L'amministratore inserisce dati incompleti e il sistema mostra un messaggio di errore richiedendo la correzione prima di salvare.</p> <p>4.2. L'amministratore inserisce un piatto che è già presente nel menù. Errore di duplicazione segnalato.</p>
Requisiti speciali	<ul style="list-style-type: none"> <li>• Interfaccia utente intuitiva per la gestione dei piatti.</li> <li>• Validazione dei campi obbligatori (nome piatto, prezzo).</li> <li>• Tempi di risposta rapidi nell'aggiornamento del database.</li> </ul>
Elenco delle varianti tecnologiche e dei dati	Non specificati.
Frequenza di ripetizione	Qualche volta alla settimana/mese
Problemi aperti	Non specificati.

#### UC5: Modifica ordine già effettuato

Nome del caso d'uso	UC5: Modifica ordine già effettuato
Portata	Sistema ordinazione digitale
Livello	Obiettivo utente
Attore primario	Cliente
Parti interessate e interessi	<ul style="list-style-type: none"> <li>• Cliente: desidera correggere un ordine già inviato.</li> </ul>
Pre-condizioni	<ul style="list-style-type: none"> <li>• Ordine già inviato ma non ancora in preparazione.</li> </ul>
Garanzia di successo (o post-condizioni)	Ordine aggiornato nel sistema e modifiche notificate
Scenario principale di successo	<ol style="list-style-type: none"> <li>1. Il cliente accede alla sezione ordini.</li> <li>2. Il cliente seleziona l'ordine da modificare.</li> <li>3. Il cliente decide di aggiungere un piatto</li> </ol> <p><i>Il passo 3 viene ripetuto finché serve.</i></p> <ol style="list-style-type: none"> <li>4. Il sistema richiede la conferma.</li> <li>5. Il cliente conferma.</li> <li>6. Il sistema aggiorna l'ordine.</li> </ol>

Estensioni (o scenari alternativi)	<ul style="list-style-type: none"> <li>1. Il titolare riavvia il software e ripristina lo stato precedente del sistema.</li> <li>2. Il sistema ricrea lo stato precedente.</li> </ul> <p>2.1. Ordine già in preparazione. Modifica non consentita</p> <p>2.2. Ordine già completato. Impossibile modificare.</p> <p>3.1. Il cliente decide di <b>modificare</b> una portata ordinata:</p> <ol style="list-style-type: none"> <li>1. Il sistema chiede al Cliente di aggiornare le nuove informazioni per modificare il piatto</li> <li>2. Il Cliente inserisce gli ingredienti che vuole aggiungere.</li> <li>3. Il sistema chiede al Cliente di confermare le modifiche.</li> <li>4. Il cliente conferma le modifiche effettuate.</li> <li>5. Il sistema aggiorna le modifiche all'ordine.</li> </ol> <p>3.2. Il cliente decide di <b>eliminare</b> una portata ordinata:</p> <ol style="list-style-type: none"> <li>1. Il sistema chiede al Cliente quale piatto debba eliminare.</li> <li>2. Il cliente sceglie il piatto da eliminare dall'ordine.</li> <li>3. Il sistema chiede al cliente di confermare l'eliminazione.</li> <li>4. Il cliente conferma l'eliminazione.</li> <li>5. Il sistema aggiorna l'importo dopo l'eliminazione.</li> <li>6. Il sistema aggiorna l'ordine.</li> </ol> <p>3.3. Piatto non disponibile.</p>
Requisiti speciali	Non specificati.
Elenco delle varianti tecnologiche e dei dati	Interfaccia per modifica rapida dell'ordine e tracciamento.
Frequenza di ripetizione	Occasionale.
Problemi aperti	Gestione modifiche multiple consecutive da parte dello stesso cliente.

#### UC6: Pianificazione del personale

Nome del caso d'uso	UC6: Pianificazione del personale
Portata	Sistema di Gestione del Ristorante
Livello	Obiettivo Amministrativo
Attore primario	Amministratore
Parti interessate e interessi	<ul style="list-style-type: none"> <li>Amministratore: vuole garantire personale sufficiente nei turni critici.</li> <li>Dipendenti: vogliono ricevere comunicazioni chiare sui turni.</li> </ul>
Pre-condizioni	Presenza di dati aggiornati su prenotazioni e disponibilità del personale.
Garanzia di successo (o post-condizioni)	Turni di lavoro creati e comunicati allo staff.

Scenario principale di successo	<ol style="list-style-type: none"> <li>1. L'amministratore accede alla sezione "Pianificazione turni".</li> <li>2. Il sistema chiede di selezionare un intervallo temporale.</li> <li>3. Amministratore inserisce data inizio e fine.</li> <li>4. Il sistema mostra le prenotazioni e la disponibilità dei dipendenti.</li> <li>5. L'amministratore assegna i turni disponibili ai dipendenti. <i>Il passo 5 viene ripetuto finchè serve</i></li> <li>6. Il sistema chiede all'amministratore di confermare</li> <li>7. L'amministratore conferma.</li> <li>8. Il sistema salva e pubblica i turni.</li> </ol>
Estensioni (o scenari alternativi)	<ul style="list-style-type: none"> <li>• 1. Il titolare riavvia il software e ripristina lo stato precedente del sistema.</li> <li>2. Il sistema ricrea lo stato precedente.</li> </ul> <p>5.1 In caso di sovrapposizione di richieste di ferie, viene segnalato un conflitto. L'amministratore deve intervenire.</p>
Requisiti speciali	Gestione delle preferenze di orario dei dipendenti.
Elenco delle varianti tecnologiche e dei dati	Non specificati.
Frequenza di ripetizione	Settimanale.
Problemi aperti	Non specificati.

#### UC7: Generazione Report

Nome del caso d'uso	UC7: Generazione Report
Portata	Sistema di Gestione del Ristorante
Livello	Sotto-funzione
Attore primario	Amministratore
Parti interessate e interessi	Amministratore: vuole dati chiari su preferenze clienti e vendite.
Pre-condizioni	Necessaria la registrazione di tutti i dati.
Garanzia di successo (o post-condizioni)	Report prodotti e disponibili.
Scenario principale di successo	<ol style="list-style-type: none"> <li>1. L'amministratore accede alla sezione Report.</li> <li>2. Il Sistema mostra i report disponibili: "Feedback Clienti", "Piatti più ordinati", "Ricavi".</li> <li>3. Il sistema chiede all'amministratore di selezionare il report desiderato e specificare l'intervallo di tempo dalla quale prendere i dati.</li> <li>4. L'amministratore seleziona il report e inserisce le date.</li> <li>5. Il sistema genera un file di testo contenente le informazioni del report richiesto, in particolar modo: <ul style="list-style-type: none"> <li>• Feedback Clienti: <ul style="list-style-type: none"> <li>○ Recupera i feedback nel periodo specificato.</li> <li>○ Calcola la media delle valutazioni.</li> </ul> </li> </ul> </li> </ol>

	<ul style="list-style-type: none"> <li>● Piatti più ordinati: <ul style="list-style-type: none"> <li>○ Recupera gli ordini nel periodo specificato.</li> <li>○ Conta le occorrenze di ogni piatto negli ordini.</li> </ul> </li> <li>● Ricavi: <ul style="list-style-type: none"> <li>○ Somma il totale degli ordini completati.</li> </ul> </li> </ul> <p><i>Il passo 4 e 5 viene ripetuto finchè serve</i></p>
\Estensioni (o scenari alternativi)	<ul style="list-style-type: none"> <li>● 1. Il titolare riavvia il software e ripristina lo stato precedente del sistema.</li> <li>2. Il sistema ricrea lo stato precedente.</li> <li>4.1. Date selezionate non valide (es. Data di fine antecedente alla data di inizio)</li> <li>4.2. Dati insufficiente per la generazione del report richiesto.</li> </ul>
Requisiti speciali	Grafici e dashboard dinamici.
Elenco delle varianti tecnologiche e dei dati	Non specificati.
Frequenza di ripetizione	Mensile.
Problemi aperti	Non specificati.

#### UC8: Fidelity card

Nome del caso d'uso	UC8: Fidelity card
Portata	Programma Fedeltà
Livello	Obiettivo utente
Attore primario	Cliente
Parti interessate e interessi	<ul style="list-style-type: none"> <li>● Cliente: riceve sconti e aggiornamenti.</li> <li>● Amministratore: Inserisci dei premi per i clienti.</li> </ul>
Pre-condizioni	<ul style="list-style-type: none"> <li>● Il Cliente deve essere registrato.</li> </ul>
Garanzia di successo (o post-condizioni)	<ul style="list-style-type: none"> <li>● Punti aggiornati</li> <li>● Premi applicati.</li> </ul>
Scenario principale di successo	<ol style="list-style-type: none"> <li>1. Il cliente accede alla sezione fidelity.</li> <li>2. Il sistema mostra il saldo e i premi riscattabili.</li> <li>3. Il cliente seleziona un premio da riscattare.</li> <li>4. Il sistema registra il premio riscattato e aggiorna i punti.</li> </ol>
Estensioni (o scenari alternativi)	<ul style="list-style-type: none"> <li>● 1. Il titolare riavvia il software e ripristina lo stato precedente del sistema.</li> <li>2. Il sistema ricrea lo stato precedente.</li> <li>3.1. Punti insufficienti per riscattare premi.</li> <li>3.2. Premio non più disponibile.</li> </ul>
Requisiti speciali	Calcolo automatico dei punti in base all'importo speso.
Elenco delle varianti tecnologiche e dei dati	Non specificati.
Frequenza di ripetizione	Ogni acquisto.
Problemi aperti	Non specificati.



## UC9: Feedback clienti

Nome del caso d'uso	UC9: Feedback dei clienti
Portata	Sistema valutazioni
Livello	Obiettivo utente
Attore primario	Cliente
Parti interessate e interessi	Cliente: esprime opinioni sulla propria esperienza Amministratore: migliora la qualità del servizio tramite opinioni dei clienti
Pre-condizioni	Cliente ha effettuato almeno un ordine.
Garanzia di successo (o post-condizioni)	Feedback salvato e consultabile.
Scenario principale di successo	<ol style="list-style-type: none"> <li>1. Il cliente accede alla sezione Feedback.</li> <li>2. Il sistema mostra le prenotazioni recenti.</li> <li>3. Il cliente seleziona la prenotazione per la quale lasciare un feedback.</li> <li>4. Il Cliente compila il form con le valutazioni. <i>Il passo 3 e 4 si ripete finchè serve.</i></li> <li>5. Il sistema chiede all'utente di confermare.</li> <li>6. Il cliente conferma.</li> <li>7. Il sistema salva i feedback inseriti dal cliente.</li> </ol>
Estensioni (o scenari alternativi)	<ul style="list-style-type: none"> <li>● 1. Il titolare riavvia il software e ripristina lo stato precedente del sistema.</li> <li>2. Il sistema ricrea lo stato precedente</li> <li>2.1 Il sistema informa che il form è incompleto.</li> </ul>
Requisiti speciali	Non specificati.
Elenco delle varianti tecnologiche e dei dati	Non specificati.
Frequenza di ripetizione	Dopo ogni visita.
Problemi aperti	Recensioni non veritiere.

## UC10: Gestione Tavoli (CRUD)

Nome del caso d'uso	UC10: Gestione Tavoli
Portata	Sistema Gestione del Ristorante
Livello	Obiettivo amministrativo
Attore primario	Amministratore
Parti interessate e interessi	Amministratore: vuole gestire (creare, modificare, cercare, cancellare e visualizzare) i tavoli disponibili nel locale, per poter assegnare correttamente le prenotazioni in base a disponibilità e numero posti.
Pre-condizioni	L'amministratore è autenticato nel sistema.
Garanzia di successo (o post-condizioni)	I dati dei tavoli vengono aggiornati correttamente e sono disponibili nel sistema.
Scenario principale di successo	<ol style="list-style-type: none"> <li>1. L'amministratore richiede la <b>creazione</b> di un nuovo tavolo.</li> <li>2. Il sistema richiede le informazioni del tavolo.</li> </ol>

	<p>3. L'amministratore inserisce numeroTavolo, numeroPosti, Stato.</p> <p><i>Il passo 2 e 3 si ripete finchè serve.</i></p> <p>4. Il sistema richiede la conferma.</p> <p>5. L'amministratore effettua la conferma.</p> <p>6. Il sistema salva il nuovo tavolo nell'elenco.</p>
Estensioni (o scenari alternativi)	<ul style="list-style-type: none"> <li>● 1. Il titolare riavvia il software e ripristina lo stato precedente del sistema.</li> <li>2. Il sistema ricrea lo stato precedente</li> </ul> <p>1.1. L'amministratore richiede la <b>modifica</b> di un tavolo:</p> <ol style="list-style-type: none"> <li>1. Il sistema richiede le informazioni del tavolo da modificare.</li> <li>2. L'amministratore inserisce numeroTavolo, numeroPosti, Stato.</li> <li>3. Il sistema richiede la conferma.</li> <li>4. L'amministratore conferma.</li> <li>5. Il sistema aggiorna le informazioni del tavolo modificato.</li> </ol> <p>1.2. L'amministratore richiede la <b>ricerca</b> di un tavolo:</p> <ol style="list-style-type: none"> <li>1. Il sistema richiede di inserire le informazioni.</li> <li>2. L'amministratore inserisce le informazioni.</li> <li>3. Il sistema mostra l'elenco dei tavoli disponibili.</li> </ol> <p>1.3. L'amministratore richiede l'<b>eliminazione</b> di un tavolo:</p> <ol style="list-style-type: none"> <li>1. Il sistema richiede di inserire il numero di tavolo da eliminare.</li> <li>2. L'amministratore inserisce le informazioni.</li> <li>3. Il Sistema mostra il tavolo e chiede all'amministratore di confermare l'eliminazione.</li> <li>4. L'Amministratore conferma.</li> <li>5. Il Sistema elimina il tavolo dal sistema.</li> </ol> <p>2.1 Il sistema informa che il form è incompleto.</p> <p>6.1. Tavolo già presente nel sistema con il medesimo numero.</p>
Requisiti speciali	Non specificati.
Elenco delle varianti tecnologiche e dei dati	Non specificati.
Frequenza di ripetizione	Mensile.
Problemi aperti	Non specificati.

## Elaborazione

### Introduzione

La fase di elaborazione ha l'obiettivo di: raffinare la Visione, implementare il nucleo dell'architettura del software, risolvere le eventuali problematiche collegate ai rischi, individuare i requisiti e fornire stime realistiche del piano di lavoro. In questa fase si analizzano e si implementano i seguenti casi d'uso:

### Iterazione 1:

*UC1: Gestione Prenotazione*, è un caso d'uso CRUD e si occupa di inserire, modificare, cercare ed eliminare le prenotazioni memorizzate nel software.

*UC4: Gestione Menu*, è un caso d'uso CRUD e con questo caso d'uso l'Amministratore può inserire, modificare, eliminare dei piatti dal menù.

*UC6: Pianificazione del personale*, è un caso d'uso che si occupa di gestire l'organizzazione del personale in base all'esigenze previste, dato dal numero di prenotazioni.

*UC10: Gestione Tavoli*, è un caso d'uso CRUD e con questo caso d'uso l'Amministratore può inserire, modificare, ricercare ed eliminare dei tavoli, in modo da organizzare al meglio il ristorante.

### Iterazione 2:

*UC2: Gestione Ordine*, questo caso d'uso permette ai clienti di effettuare ordini di piatti dal menu.

*UC3: Gestione Pagamento*, questo caso è gestito dall'amministratore e si occupa della gestione dei pagamenti per ogni tavolo.

*UC5: Modifica ordine già effettuato*, permette a un cliente di apportare modifiche a un ordine già inviato, purché non sia ancora in preparazione. Il cliente può aggiungere, modificare o eliminare portate.

*UC8: Fidelity Card*, questo caso d'uso gestisce il programma fedeltà per i clienti.

### **Iterazione 3:**

*UC7: Gestione Report*, è un caso d'uso in cui il sistema deve generare report sulle vendite (piatti più ordinati, ricavi e feedback dei clienti).

*UC9: Feedback clienti*, con questo caso d'uso i clienti devono poter lasciare valutazioni e commenti dopo il pasto.

### **Analisi Orientata agli Oggetti**

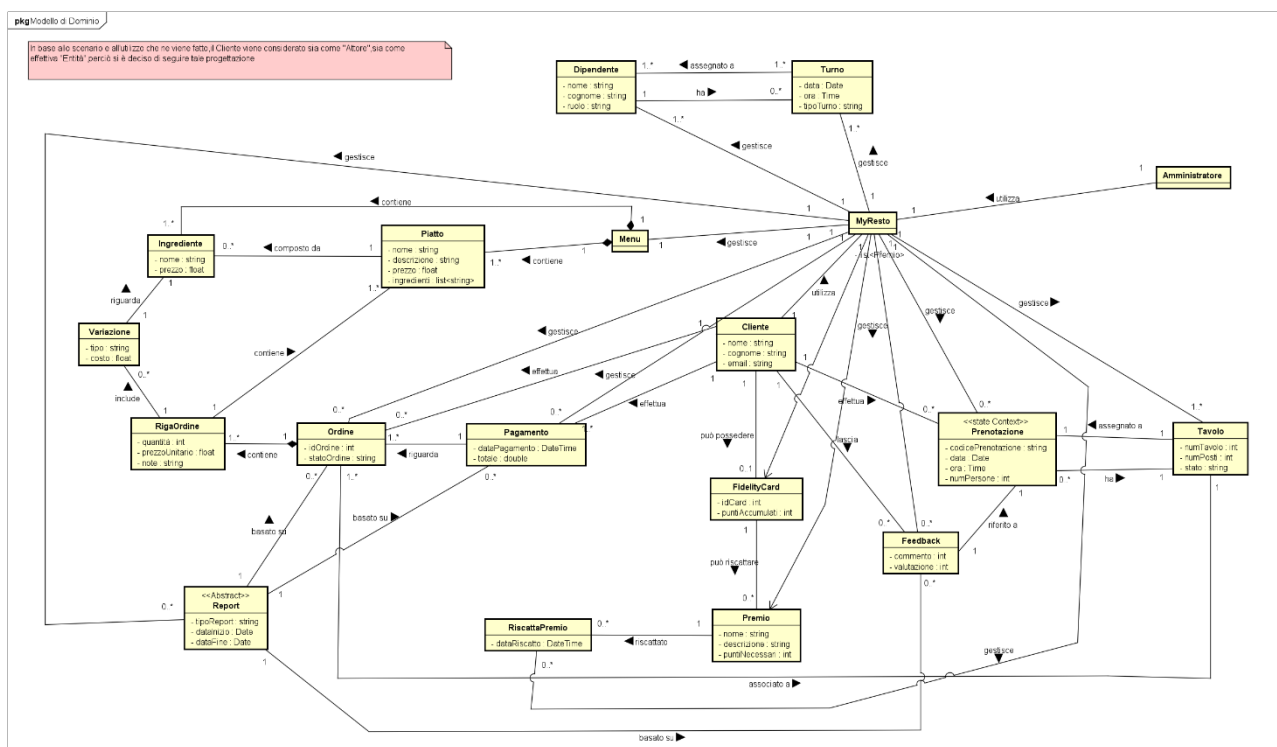
L'analisi orientata agli oggetti è basata sulla descrizione del dominio pensata ad oggetti. Gli strumenti utilizzati sono: Modello di Dominio, SSD (Sequence System Diagram) e Contratti delle operazioni.

### **Modello di Dominio**

La Modellazione del Business comprende la stesura del Modello di Dominio ed un elaborato grafico che identifica i concetti, gli attributi e le associazioni significative. Dopo una valutazione dello scenario principale di successo di tutti i casi d'uso sono state identificate le seguenti classi concettuali:

- **Cliente:** È sia attore che entità nel modello. Rappresenta l'utente finale del ristorante, interagisce con il sistema (come attore) per effettuare prenotazioni e altre operazioni.
- **MyResto:** È il sistema informatico che gestisce tutte le operazioni del ristorante.
- **Amministratore:** Colui che interagisce con il sistema.
- **Feedback:** Entità che rappresenta il giudizio lasciato dal cliente dopo un'esperienza.
- **Report:** Entità generata dal sistema su richiesta dell'amministratore.
- **Ordine:** Rappresenta l'insieme dei piatti e bevande richiesti da un cliente a un tavolo in un determinato momento.
- **RigaOrdine:** Rappresenta una singola voce all'interno di un Ordine, specificando la quantità di un piatto e il suo prezzo al momento dell'ordine.
- **Piatto:** Voce del menù. Include nome, descrizione, prezzo, ingredienti, allergeni, categoria.
- **Menù:** contiene l'elenco dei piatti e bevande disponibili.
- **Pagamento:** Entità che rappresenta il pagamento di un ordine, incluso importo, metodo, data e stato.
- **Ingrediente:** Rappresenta un singolo componente alimentare utilizzato per comporre un Piatto.
- **Variazione:** Rappresenta una modifica o un'aggiunta specifica a un ingrediente o a un piatto (es. "senza glutine", "doppio formaggio").
- **Turno:** Indica un intervallo temporale di lavoro assegnato ai dipendenti; include data e fascia oraria.

- **Dipendente:** Rappresenta un membro dello staff del ristorante (es. cameriere, cuoco). Include nome, ruolo, disponibilità e i turni assegnati.
- **RiscattaPremio:** Registra l'evento in cui un cliente riscatta un premio utilizzando i suoi punti fedeltà.
- **FidelityCard:** Rappresenta la tessera fedeltà associata a un cliente, che accumula punti in base agli acquisti e permette il riscatto di premi.
- **Premio:** Rappresenta un regalo che i clienti possono riscattare utilizzando i punti della loro Fidelity Card.
- **Prenotazione:** Rappresenta la richiesta di un cliente per riservare un tavolo in una specifica data e orario, indicando anche il numero di persone.
- **Tavolo:** Rappresenta un tavolo fisico nel ristorante. Include informazioni come numero identificativo, numero massimo di posti, posizione (intero o esterno) e stato (libero, occupato, prenotato).



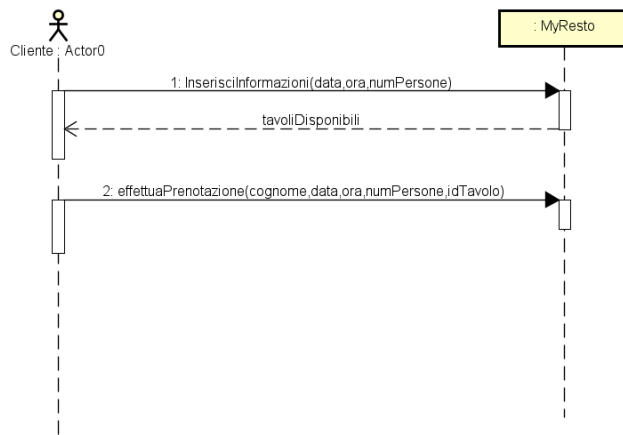
### Diagramma di sequenza di sistema

Procedendo con l'analisi Orientata agli Oggetti, il passo successivo è la creazione del Diagramma di Sequenza di Sistema (SSD) al fine di illustrare il corso degli eventi di input e di output per lo scenario principale di successo nei casi d'uso scelti (UC1, UC4, UC6 e UC10), quindi avremo:

## Iterazione 1

**UC1:**

sdSSD-UC1

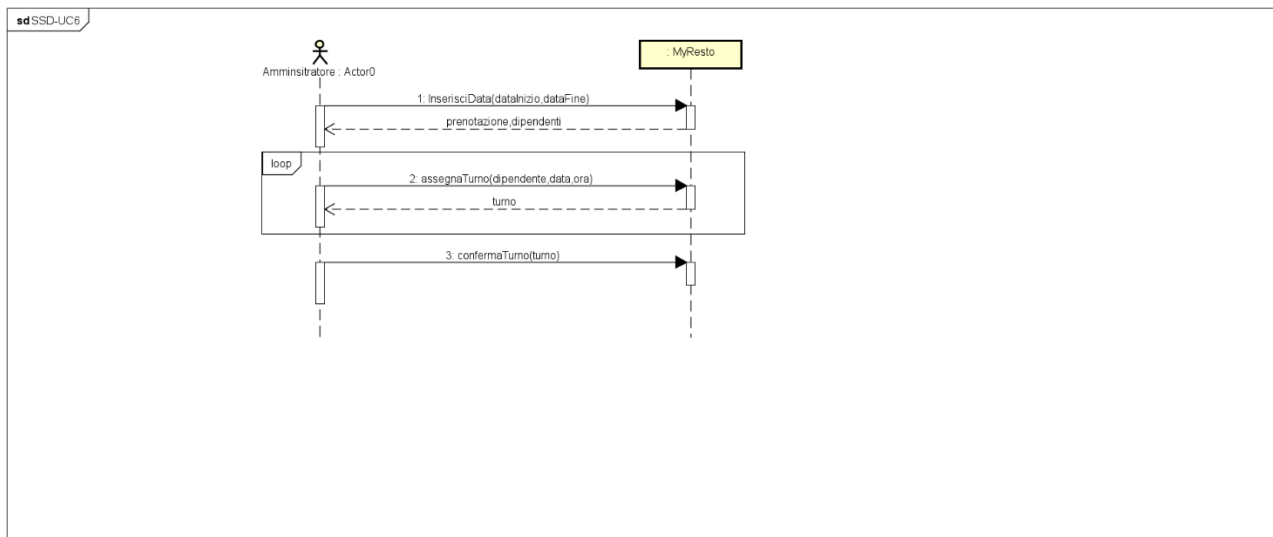


#### UC4:

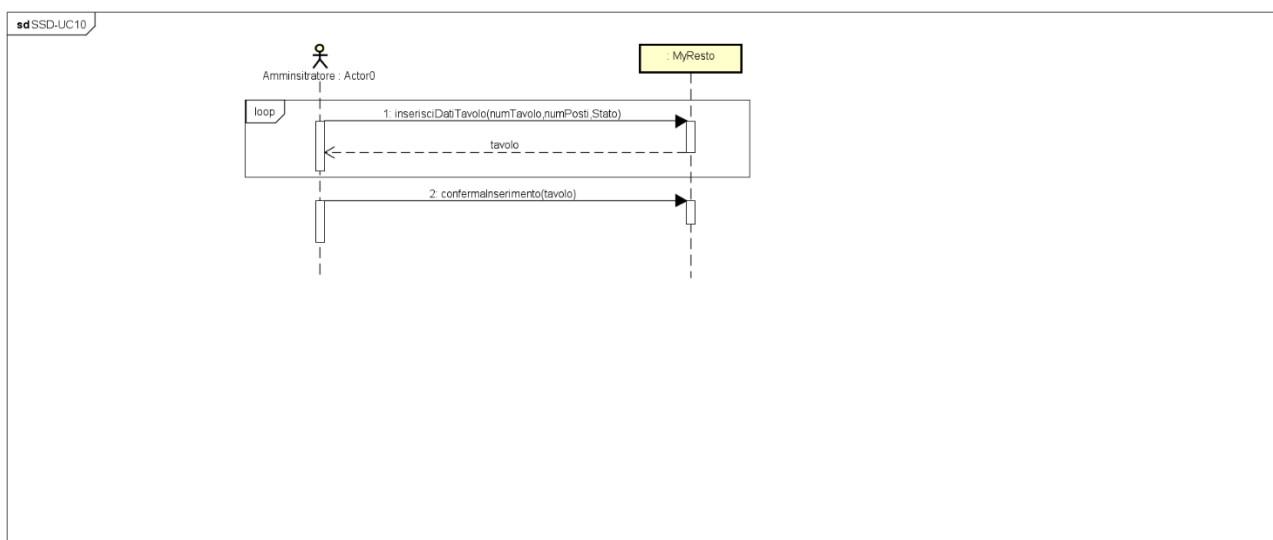
sdSSD-UC4



#### UC6:



### UC10:



## Contratti delle operazioni

Vengono ora descritte attraverso i Contratti le principali operazioni di sistema che si occupano di gestire gli eventi di sistema individuati nell'SSD.

### UC1:

#### Contratto CO1: InserisciInformazioni

**Operazione:** *InserisciInformazioni*(data: Date, ora: Time, numPersone: int).

**Riferimenti:** caso d'uso: Gestione Prenotazione.

**Pre-condizioni:** è in corso l'inserimento di una nuova prenotazione.

#### **Post-condizioni:**

-il sistema ha cercato nella lista dei tavoli (List<Tavolo>) quelli disponibili per la data, ora e numero di persone indicati.

- È stata generata una lista filtrata di tavoli disponibili, compatibili con la richiesta.

#### **Contratto CO2:effettuaPrenotazione**

**Operazione:** *inserisciDatiPrenotazione(cognome: String, data: Date, ora: Time, numPersone: int, idTavolo: int)*

**Riferimenti:** caso d'uso: Gestione Prenotazione

**Pre-condizioni:** Il Tavolo con idTavolo esiste e non è ancora occupato per la data e ora specificate.

**Post-condizioni:**

- È stata creata una nuova istanza preno di Prenotazione.
- L'istanza preno è stata aggiunta a Prenotazioni:List<Prenotazione>.
- L'istanza t di Tavolo ha visto il suo stato aggiornato a "OCCUPATO" per la data e ora della prenotazione.

#### **UC4:**

##### **Contratto CO1: inserisciPiatto**

**Operazione:** *inserisciPiatto(nome: String, descrizione: String, prezzo: Float, ingredienti: List<String>)*

**Riferimenti:** caso d'uso: Gestione Menu.

**Pre-condizioni:** è in corso l'inserimento di un nuovo piatto nel menù

**Post-condizioni:**

- È stata creata una nuova istanza p di Piatto con i dettagli forniti (nome, descrizione, prezzo, ingredienti).
- Il Menu è stato aggiornato.

##### **Contratto CO2: confermaInserimento**

**Operazione:** *confermaInserimento(piatto: Piatto)*

**Riferimenti:** caso d'uso: Gestione Menu.

**Pre-condizioni:** è stato precedentemente creato un nuovo piatto ed è pronto per essere aggiunto al menu.

**Post-condizioni:** il piatto specificato è stato aggiunto alla lista Piatti (List<Piatto>).

#### **UC6:**

##### **Contratto CO1: inserisciData**

**Operazione:** *inserisciData(dataInizio: Date, dataFine: Date)*

**Riferimenti:** caso d'uso: Pianificazione del personale.



**Pre-condizioni:** è in corso la pianificazione dei turni.

**Post-condizioni:**

-sono state recuperate le prenotazioni comprese tra dataInizio e dataFine dalla lista Prenotazioni (List<Prenotazione>).

-sono stati selezionati e poi restituiti al sistema i dipendenti disponibili per il periodo richiesto dalla lista Dipendenti (List<Dipendente>).

### **Contratto CO2: assegnaTurno**

**Operazione:** *assegnaTurno(dipendente: String, data: Date, ora: Time)*

**Riferimenti:** caso d'uso: Pianificazione del personale

**Pre-condizioni:** è in corso l'assegnazione di un turno a un dipendente.

**Post-condizioni:**

-è stata creata un'istanza turno di Turno

-il sistema ha aggiornato la pianificazione corrente con il nuovo turno assegnato.

### **Contratto CO3: confermaTurni**

**Operazione:** *confermaTurni(turno: List<String>)*

**Riferimenti:** caso d'uso: Pianificazione del personale.

**Pre-condizioni:** è in corso la conferma di un turno.

**Post-condizioni:**

-il turno è stato aggiunto alla lista Turni (list<Turno>).

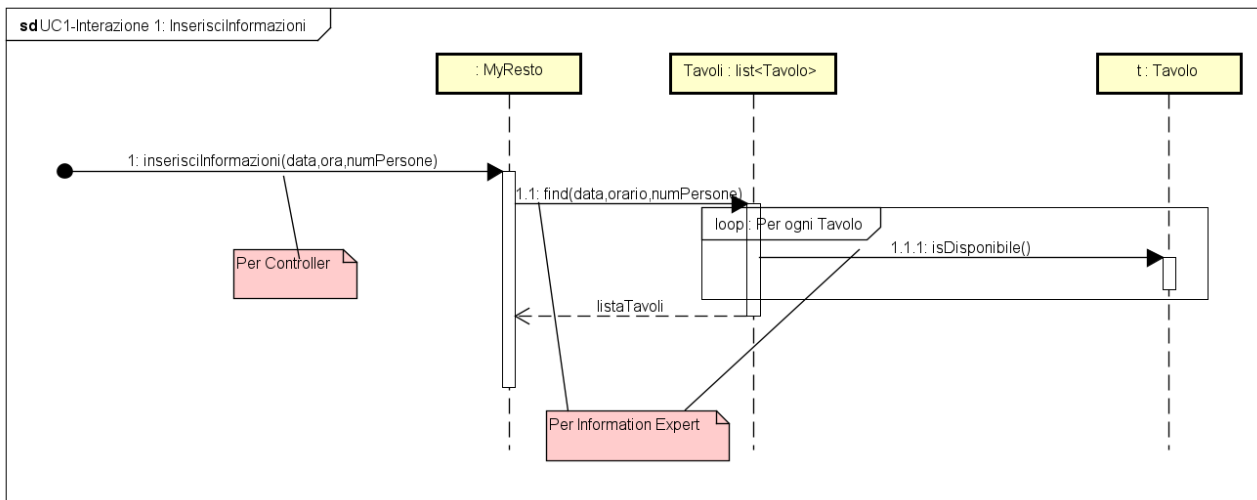
## **Progettazione**

La progettazione orientata agli oggetti è la disciplina di UP interessata alla definizione degli oggetti software, delle loro responsabilità e a come questi collaborano per soddisfare i requisiti individuati nei passi precedenti. L'elaborato principale di questa fase, che è stato preso in considerazione, è il Modello di Progetto, ovvero l'insieme dei diagrammi che descrivono la progettazione logica sia da un punto di vista dinamico (Diagrammi di Interazione) che da un punto di vista statico (Diagrammi delle Classi). Seguono dunque i diagrammi di interazione più significativi e il diagrammi delle classi relativi ai casi d'uso UC1, UC4, UC6 e UC10 determinati a seguito di un attento studio degli elaborati scritti in precedenza.

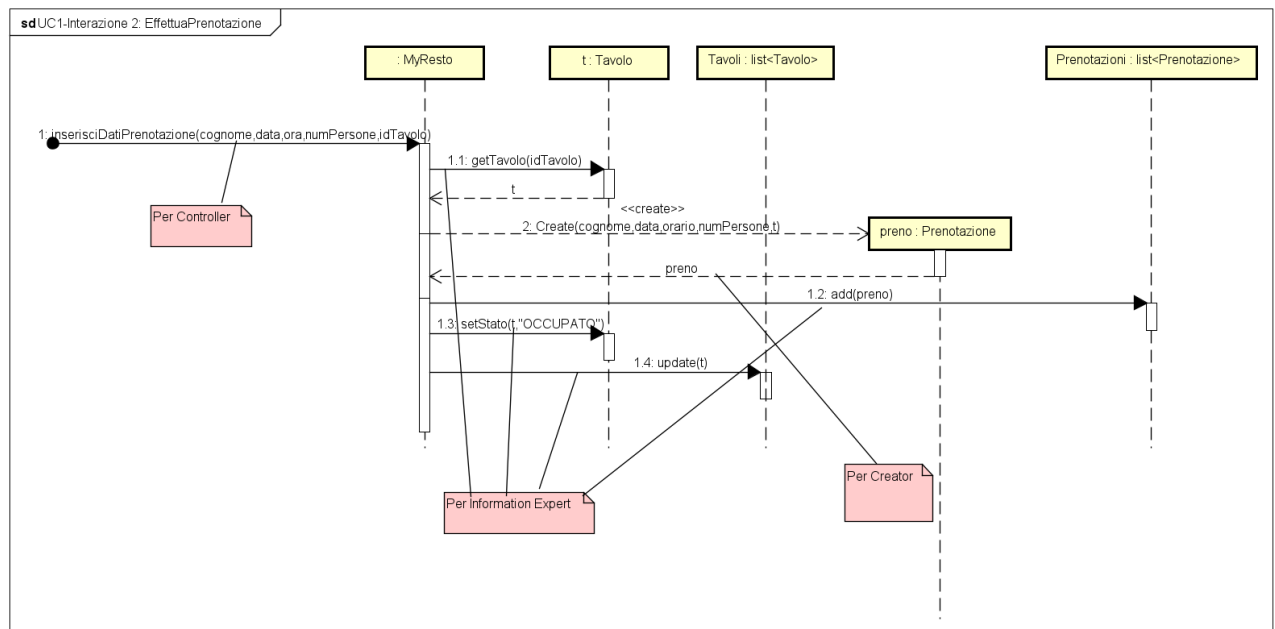
## **Diagrammi di Sequenza**

**UC1:**

- ***inserisciInformazioni***

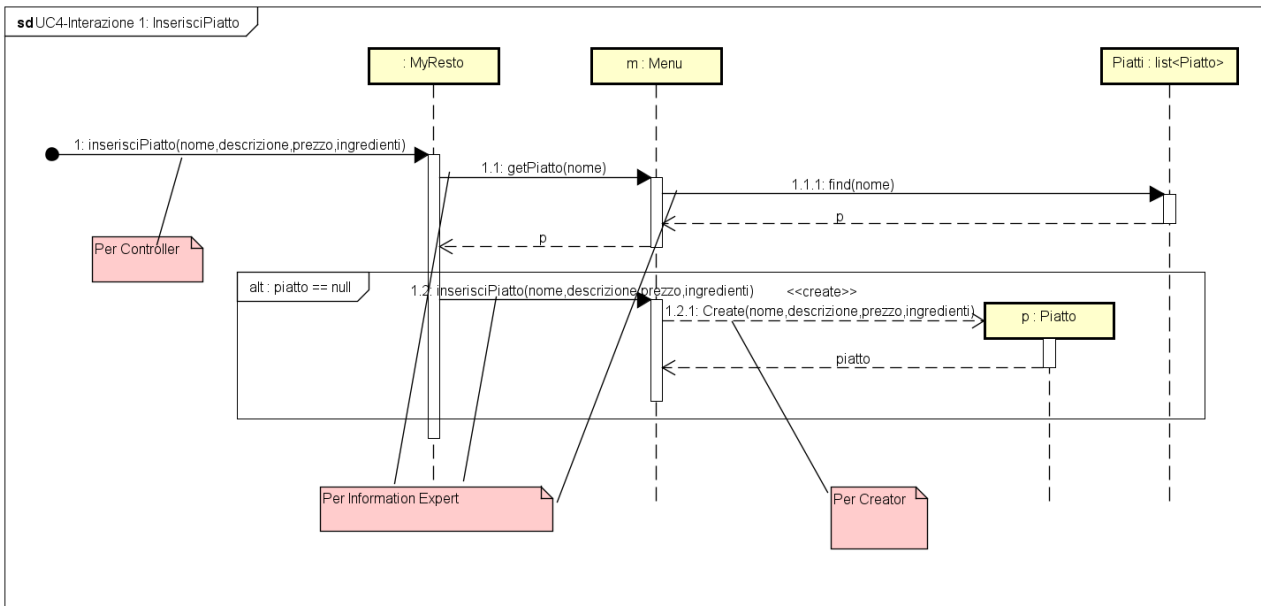


- effettuaPrenotazione**

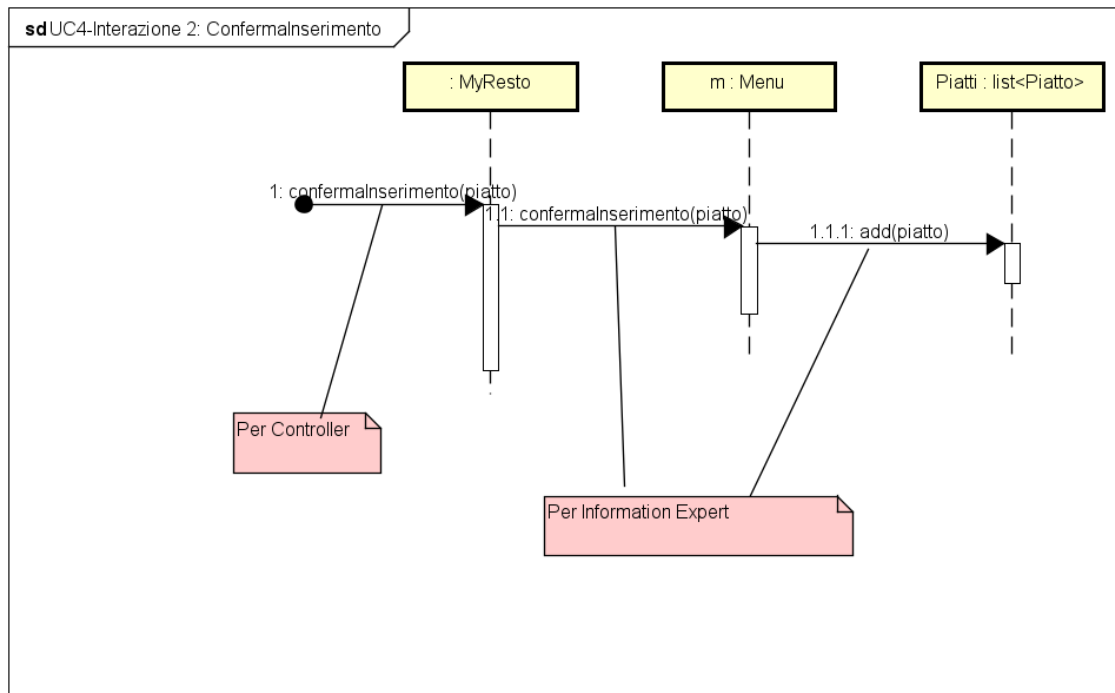


**UC4:**

- inserisciPiatto**

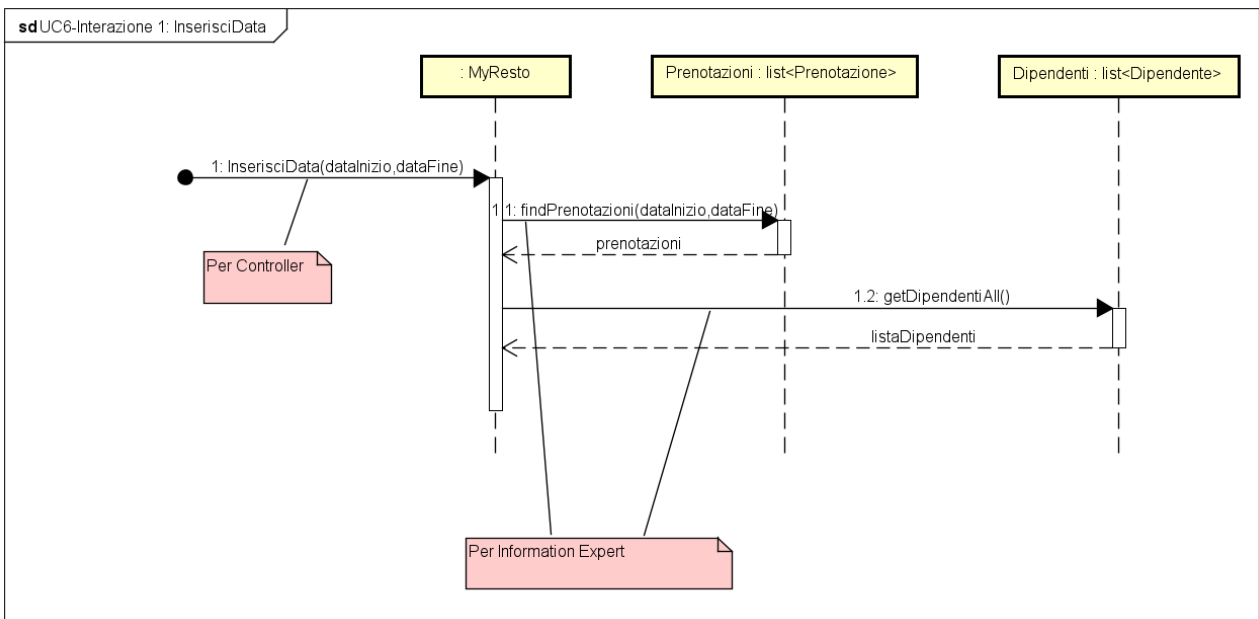


- **confermaInserimento**

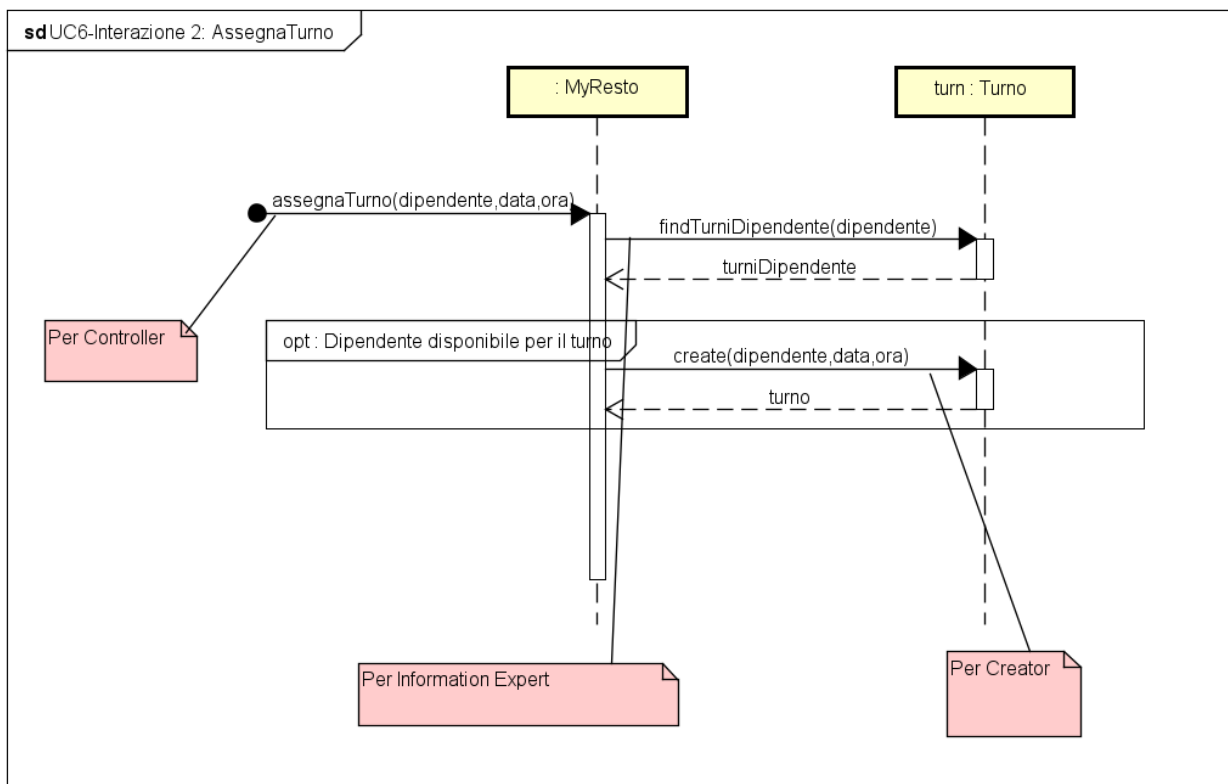


**UC6:**

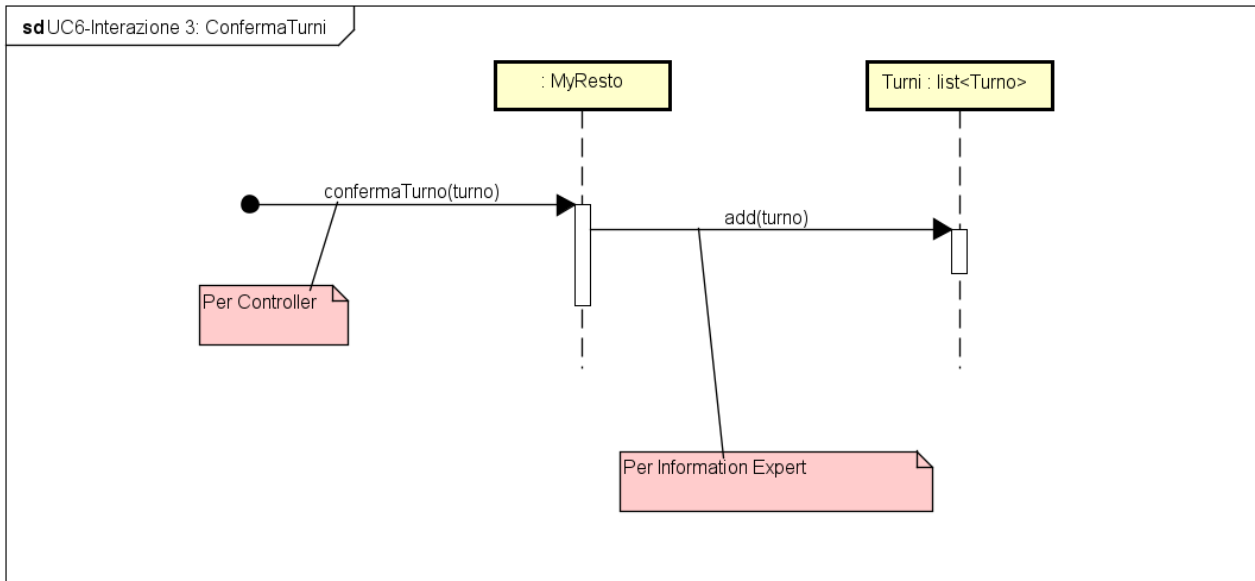
- **inserisciData**



- **assegnaTurno**

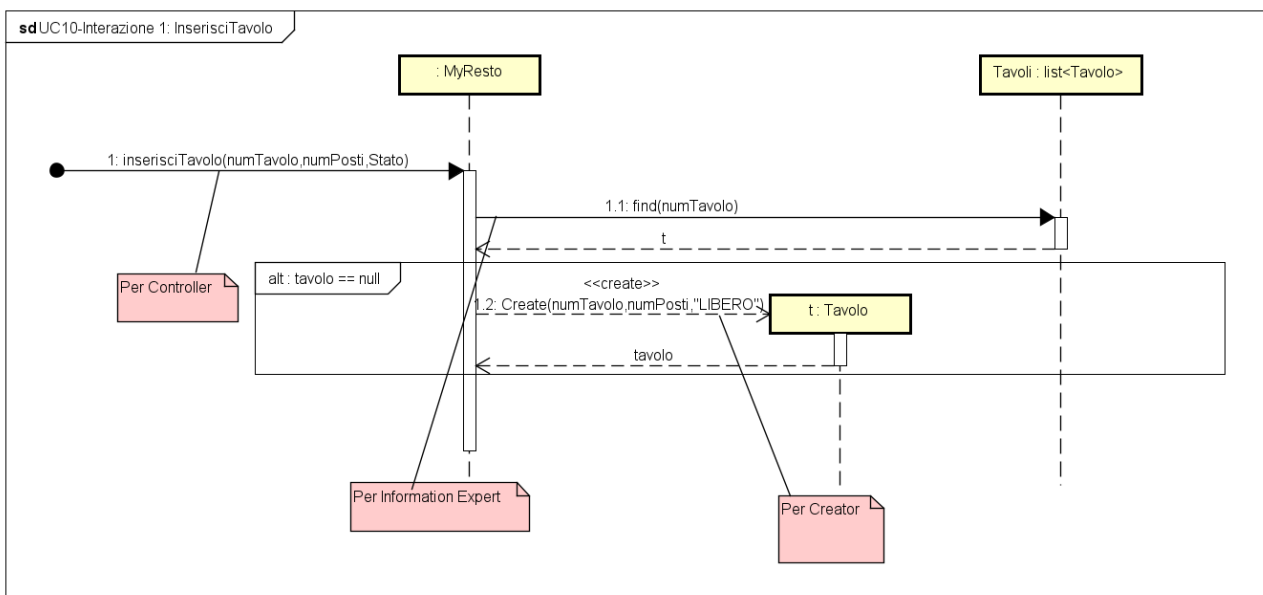


- **confermaTurni**

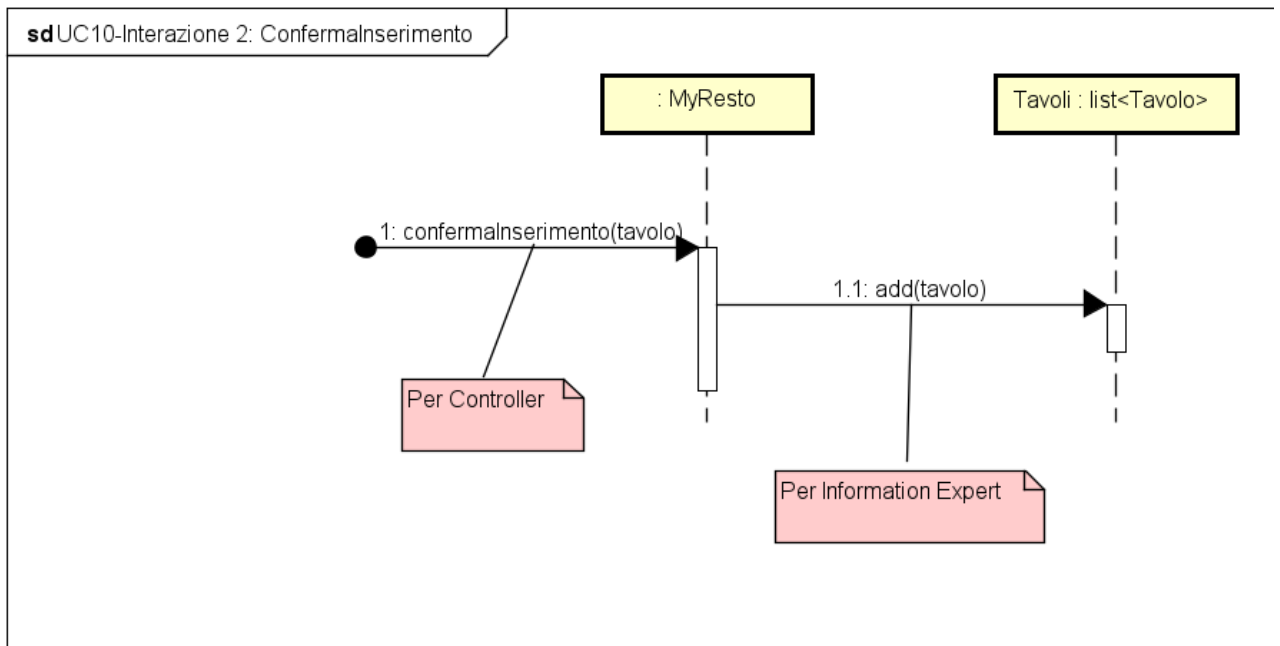


## UC10:

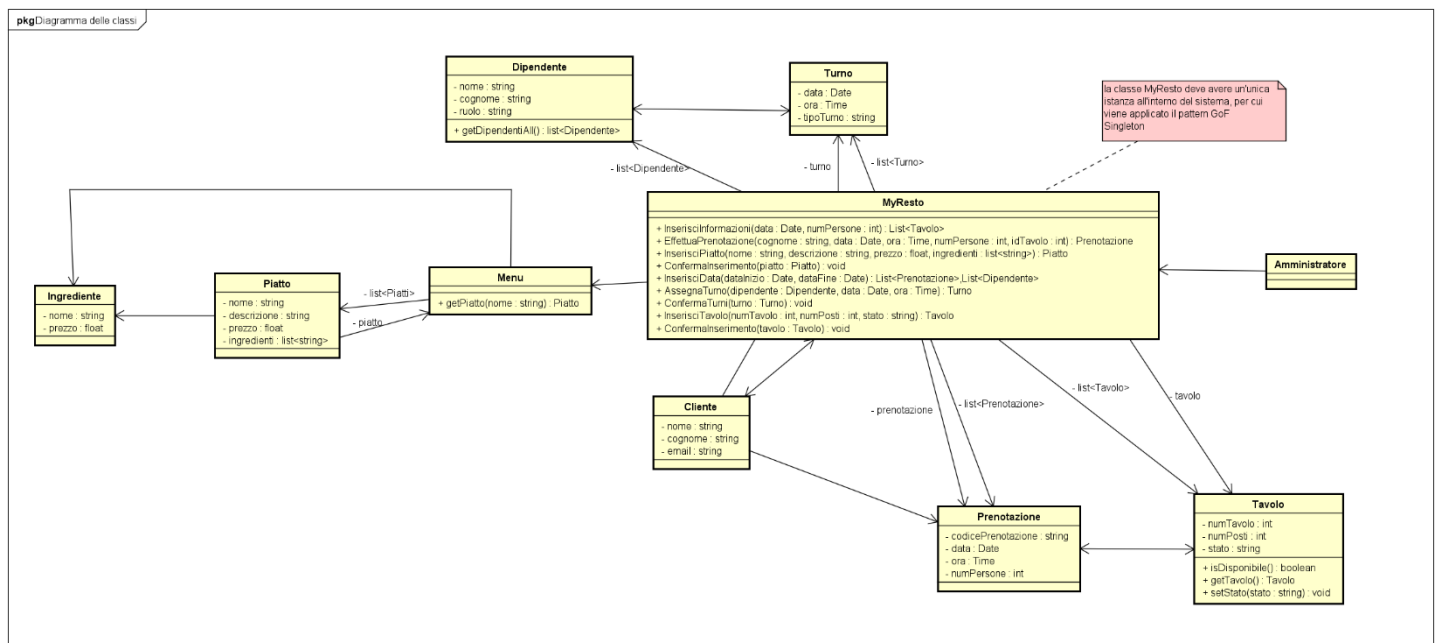
- inserisciTavolo**



- **confermaInserimento**



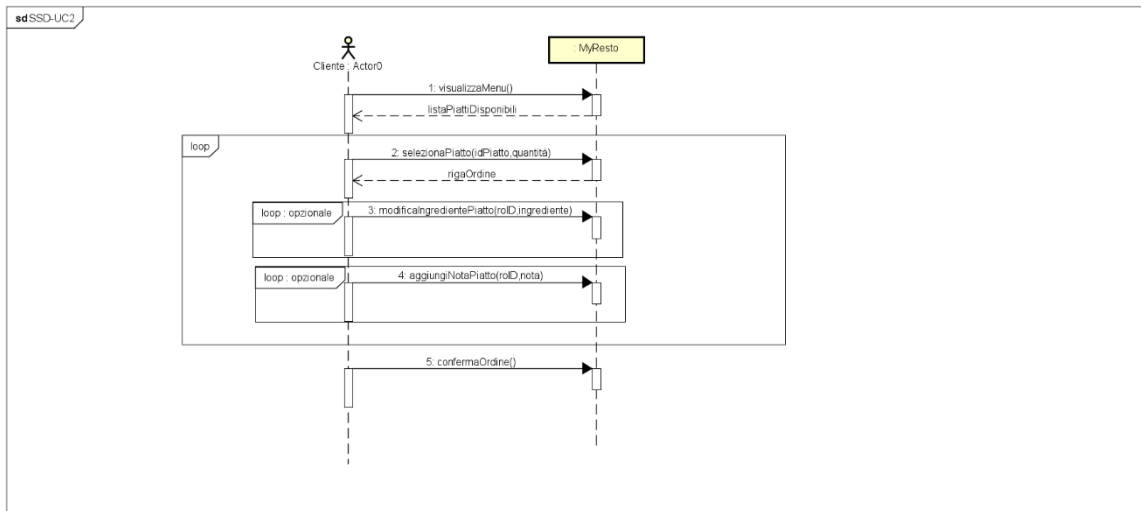
## Diagramma delle classi



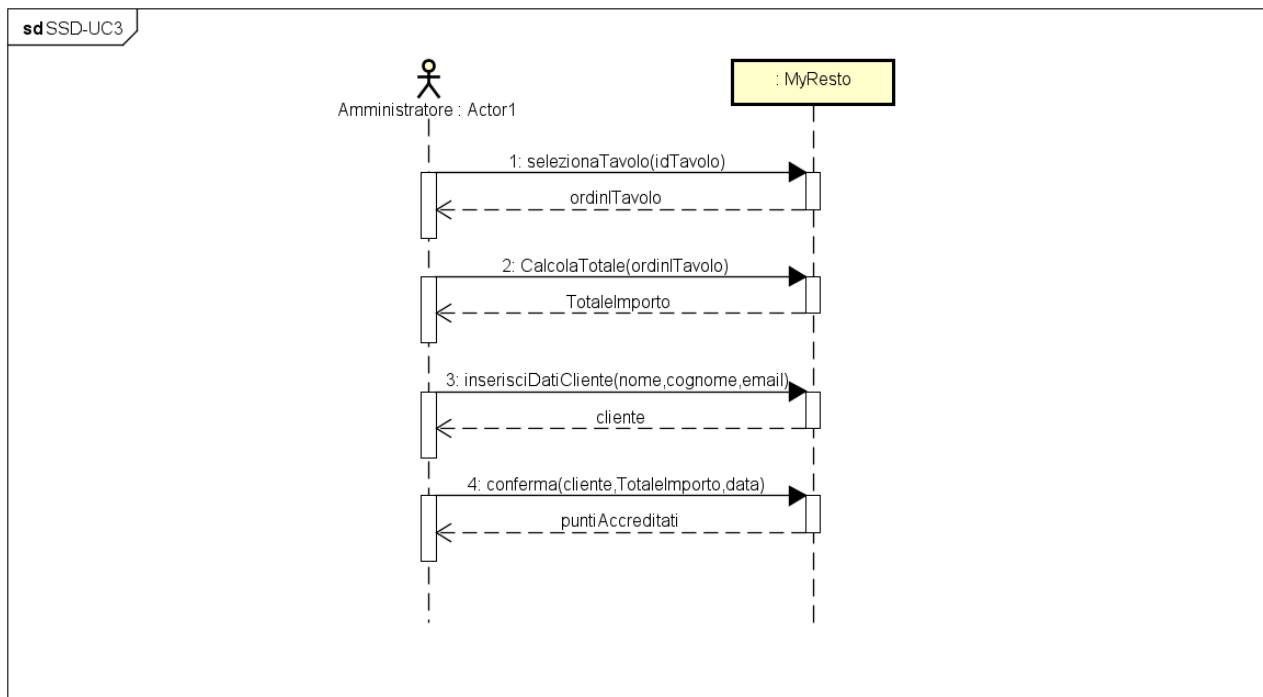
## Iterazione 2

### Diagramma di sequenza di sistema

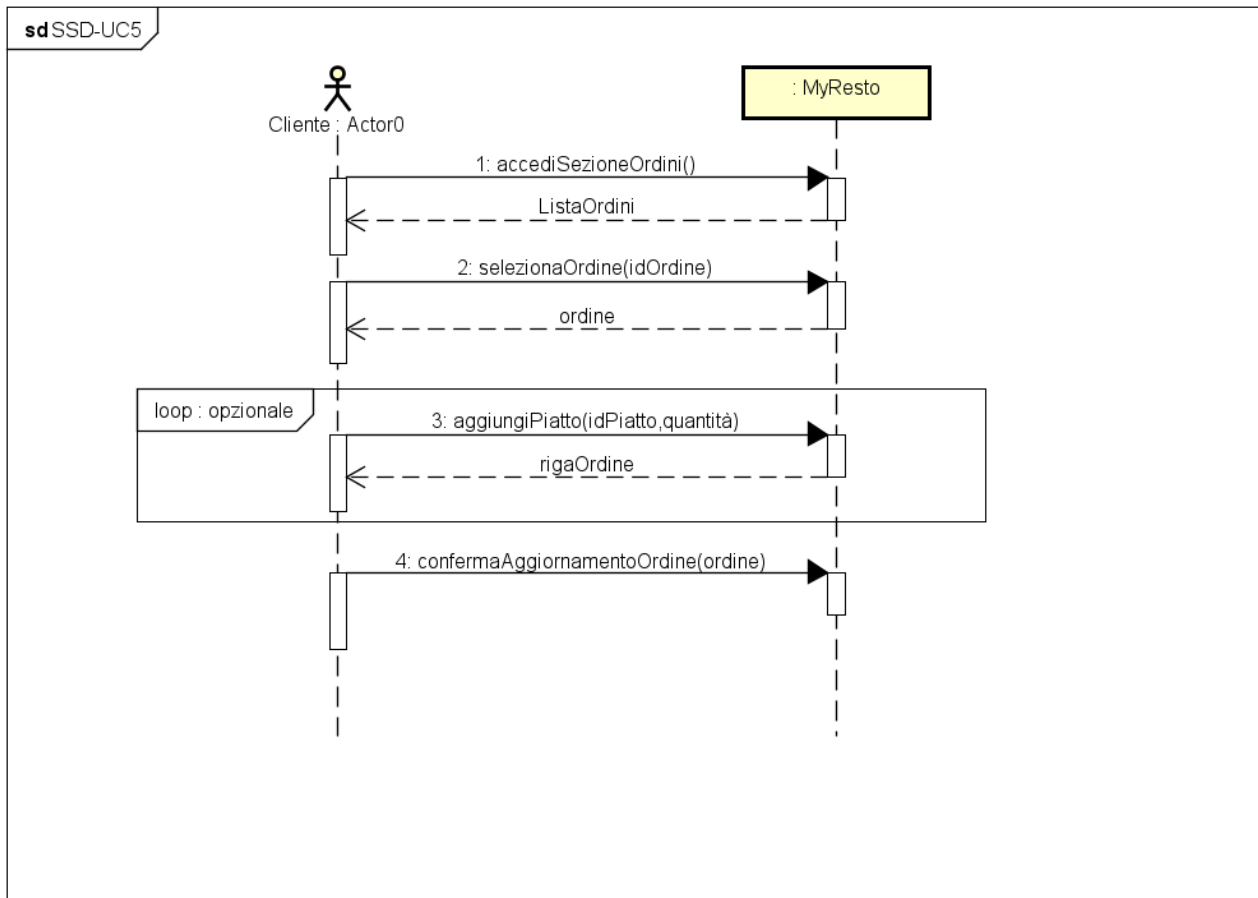
#### UC2:



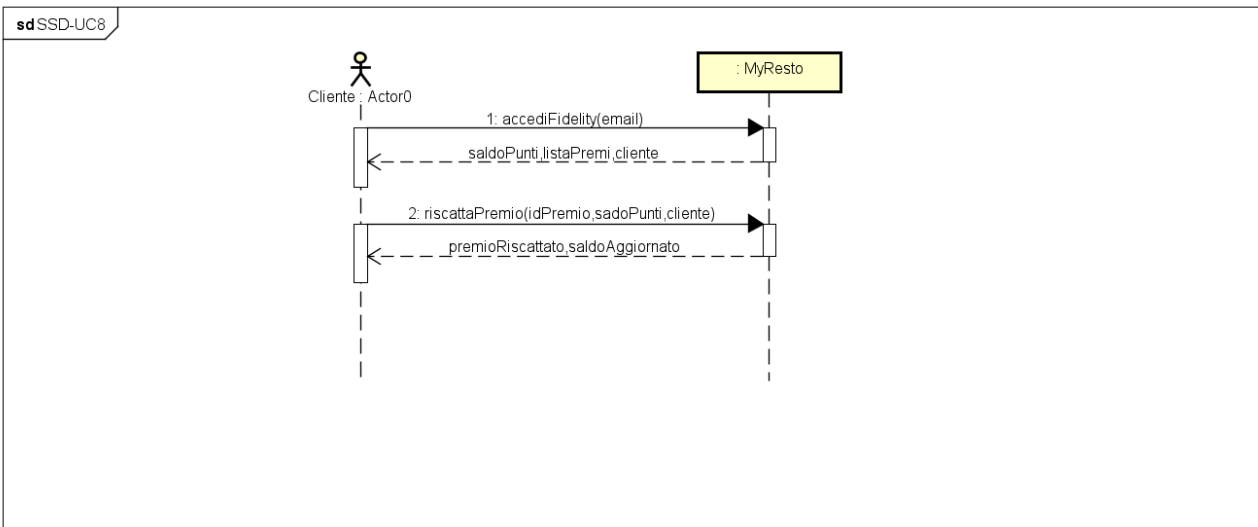
#### UC3:



### UC5:



### UC8:



## Contratti delle operazioni

### UC2:

#### Contratto CO1: visualizzaMenu

Operazione: *visualizzaMenu()*.



**Riferimenti: caso d'uso:** Gestione Ordine.

**Pre-condizioni:**

- Il cliente ha un tavolo assegnato.
- Il sistema è operativo e il menu è configurato e disponibile.

**Post-condizioni:**

- Il sistema accede all'istanza m del Menu.
- Vengono recuperati tutti i piatti dalla collezione Piatti:list<Piatto> associata al Menu.
- Una lista di piatti (listaPiatti) è disponibile per essere mostrata al cliente, permettendogli di selezionare i piatti desiderati.

**Contratto CO2: selezionaPiatto**

**Operazione:** *selezionaPiatto(idPiatto:int, quantità:int).*

**Riferimenti: caso d'uso:** Gestione Ordine.

**Pre-condizioni:**

- Il cliente sta visualizzando il menu.
- L'idPiatto fornito corrisponde a un piatto esistente e disponibile nel Menu.

**Post-condizioni:**

- Se non c'è nessun ordine in corso viene creata una nuova istanza o di Ordine.
- Se l'ordine è già in corso il sistema recupera l'istanza p di Piatto corrispondente a idPiatto dalla collezione Piatti:list<Piatto>.
- Viene creata una nuova istanza ro di RigaOrdine con il Piatto (p) e la quantità specificata.
- La RigaOrdine creata (ro) viene aggiunta alla collezione righeOrdine:list<RigaOrdine> dell'ordine corrente.

**Contratto CO3: modificaIngredientePiatto**

**Operazione:** *modificaIngredientePiatto(roID:int,ingrediente:String,tipomodifica:String)*

**Riferimenti: caso d'uso:** Gestione Ordine.

**Pre-condizioni:** Il cliente ha un ordine in corso e ha precedentemente selezionato un piatto, creando una RigaOrdine.

**Post-condizioni:**

- Il sistema recupera l'istanza ro di RigaOrdine corrispondente a roID dalla collezione righeOrdine:list<RigaOrdine>.
- Il sistema recupera l'istanza p di Piatto associato alla RigaOrdine (ro).

- Il sistema recupera l'istanza i di Ingrediente dalla collezione Ingredienti:list<Ingrediente>.
- Viene aggiunta una nuova Variazione (v) alla RigaOrdine (ro) con il tipoModifica e l'ingrediente specificati.
- L'istanza v di Variazione viene creata e aggiunta alla collezione Variazioni:list<Variazione>.

#### **Contratto CO4: aggiungiNota**

**Operazione:** *aggiungiNota(rolD:int, nota:String).*

**Riferimenti: caso d'uso:** Gestione Ordine.

**Pre-condizioni:** Il cliente ha un ordine in corso e ha precedentemente selezionato un piatto, creando una RigaOrdine.

**Post-condizioni:**

- Il sistema recupera l'istanza ro di RigaOrdine corrispondente a rolD dalla collezione righeOrdine:list<RigaOrdine>.
- La nota fornita viene aggiunta e associata alla RigaOrdine (ro).

#### **Contratto CO5: confermaOrdine**

**Operazione:** *confermaOrdine().*

**Riferimenti: caso d'uso:** Gestione Ordine.

**Pre-condizioni:** Il cliente ha un Ordine in corso e che contiene almeno una RigaOrdine.

**Post-condizioni:** L'istanza dell'ordine corrente viene aggiunta alla collezione ordiniSalvati:list<Ordine>.

**UC3:**

#### **Contratto CO1: selezionaTavolo**

**Operazione:** *selezionaTavolo(idTavolo:int)*

**Riferimenti: caso d'uso:** Gestione Pagamento.

**Pre-condizioni:** L'Amministratore è autenticato nel sistema.

**Post-condizioni:**

- Il sistema cerca l'ordine associato al idTavolo specificato all'interno della collezione ordiniSalvati:list<Ordine>.
- L'ordine del tavolo (ordiniTavolo) viene recuperato.

### **Contratto CO2: calcolaTotale**

**Operazione:** *prezzoTot(ordiniTavolo: list<Ordine>)*

**Riferimenti: caso d'uso:** Gestione Pagamento.

**Pre-condizioni:** Il sistema ha precedentemente recuperato la lista degli ordini (ordiniTavolo) associati al tavolo selezionato.

**Post-condizioni:**

- Il sistema itera attraverso ogni Ordine in ordiniTavolo.
- Per ogni Ordine, il sistema recupera la sua listaRO.
- Per ogni ro di RigaOrdine (ro):
  - Viene recuperato il prezzo base del piatto associato a ro.
  - Viene recuperata la listaV (lista di Variazione) associata a ro.
  - Per ogni Variazione in listaV, viene recuperato il costoV della variazione.
  - Il prezzo della RigaOrdine viene aggiornato per includere i costoV delle variazioni.
- Il sistema calcola il subtotale per ogni Ordine sommando i prezzi di tutte le sue RigaOrdine modificate.
- Il sistema calcola il totaleImporto complessivo sommando i subtotale di tutti gli Ordine in ordiniTavolo.

### **Contratto CO3: inserisciCliente**

**Operazione:** *inserisciDatiCliente(nome:String, cognome:String, email:String).*

**Riferimenti: caso d'uso:** Gestione Pagamento.

**Pre-condizioni:**

- L'Amministratore ha calcolato il totale dell'ordine e sta procedendo con la gestione del pagamento.
- I dati nome, cognome, email del cliente sono stati forniti dall'amministratore.

**Post-condizioni:**

- Il sistema cerca un cliente esistente nella collezione Clienti:list<Cliente> utilizzando l'email fornita.
- Se il cliente non esiste:
  - Una nuova istanza cliente di Cliente viene creata con i nome, cognome e email forniti.
  - L'istanza cliente di Cliente viene aggiunta alla collezione Clienti:list<Cliente>.
- Se il Cliente esiste:
  - Viene recuperata l'istanza c di Cliente dalla collezione Clienti:list<Cliente> con l'email fornita.

#### **Contratto CO4: conferma**

**Operazione:** *conferma(cliente: Cliente, totaleImporto: float, data: Date)*

**Riferimenti: caso d'uso:** Gestione Pagamento.

**Pre-condizioni:**

- Il TotaleImporto è stato precedentemente calcolato.
- I dati del cliente sono stati inseriti o recuperati.

**Post-condizioni:**

- Viene creata una nuova istanza pa di Pagamento con i parametri cliente, TotaleImporto e data.
- L'istanza pa di Pagamento viene aggiunta alla collezione Pagamenti:list<Pagamento>.
- Se il cliente non è nullo:
  - Il sistema recupera la FidelityCard (fc) associata al cliente.
  - Il sistema ottiene i punti correnti dalla FidelityCard (fc).
  - I punti della FidelityCard (fc) vengono aggiornati aggiungendo TotaleImporto al saldo esistente.
  - L'istanza del cliente viene aggiornata nel sistema per riflettere il nuovo saldo punti.

#### **UC5:**

#### **Contratto CO1: accediSezioneOrdini**

**Operazione:** *accediSezioneOrdini()*

**Riferimenti: caso d'uso:** Modifica ordine già effettuato.

**Pre-condizioni:** Il Cliente ha effettuato l'accesso al sistema.

**Post-condizioni:**

- Il sistema recupera tutti gli ordini dalla collezione ordiniSalvati:list<Ordine>.
- Una lista di ordini (listaOrdini) è disponibile per essere visualizzata dal cliente, permettendogli di selezionare l'ordine da modificare.

#### **Contratto CO2: selezioneOrdine**

**Operazione:** *selezionaOrdine(idOrdine: int)*

**Riferimenti: caso d'uso:** Modifica ordine già effettuato.

**Pre-condizioni:** Il cliente ha precedentemente visualizzato una lista di ordini disponibili per la modifica.

**Post-condizioni:**

- Il sistema cerca l'istanza dell'Ordine (ordine) corrispondente a idOrdine all'interno della collezione ordiniSalvati:list<Ordine>.

-L'istanza dell'Ordine selezionato (ordine) viene recuperata e resa disponibile al sistema.

### **Contratto CO3: confermaAggiornamentoOrdine**

**Operazione:** *confermaAggiornamentoOrdine(ordine: Ordine).*

**Riferimenti:** caso d'uso: Modifica ordine già effettuato.

**Pre-condizioni:** Un ordine (ordine) è stato precedentemente selezionato dal cliente per la modifica.

**Post-condizioni:** L'istanza ordine viene aggiornata nella collezione ordiniSalvati:list<Ordine>.

### **UC8:**

### **Contratto CO1: accediFidelity**

**Operazione:** *accediFidelity(email:String)*

**Riferimenti:** caso d'uso: Fidelity Card.

**Pre-condizioni:** Il cliente è registrato nel sistema.

**Post-condizioni:**

-Il sistema cerca l'istanza c del Cliente corrispondente all'email fornita all'interno della collezione Clienti:list<Cliente>.

-Se il cliente (c) viene trovato:

-Il sistema recupera l'istanza fc della FidelityCard associata al cliente.

-Viene recuperato il saldo punti corrente dalla FidelityCard.

-Viene recuperata la listaPremi disponibile dalla collezione Premi:list<Premio>.

### **Contratto CO2: riscattaPremio**

**Operazione:** *riscattaPremio(idPremio:int, saldoPunti:int, cliente:Cliente)*

**Riferimenti:** caso d'uso: Fidelity Card.

**Pre-condizioni:** Il cliente ha effettuato l'accesso alla sezione Fidelity Card.

**Post-condizioni:**

-Il sistema recupera l'istanza prem di Premio corrispondente a idPremio dalla collezione Premi:list<Premio>.

-Il sistema recupera il costoPunti del Premio (prem).

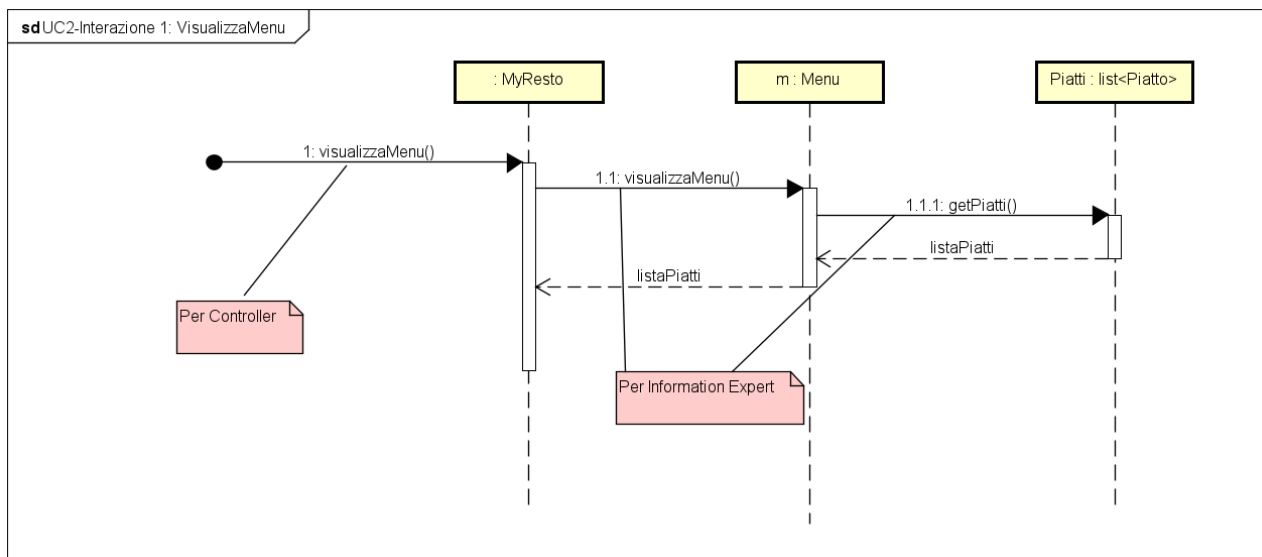
-Viene creata una nuova istanza pr di PremioRiscattato con il cliente, il premio e la dataRiscatto corrente.

-L'istanza pr di PremioRiscattato viene aggiunta alla collezione premiRiscattati:list<PremioRiscattato>.

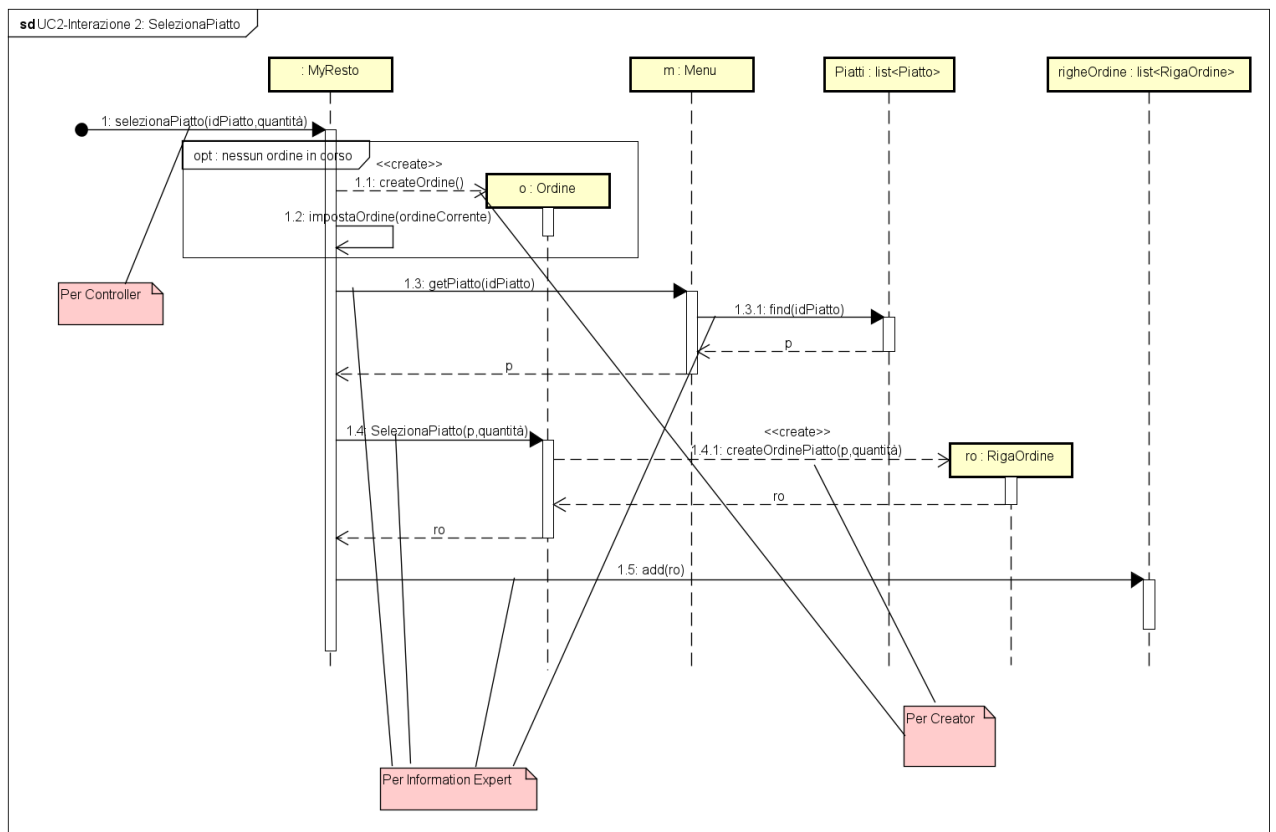
-L'istanza del cliente viene aggiornata nel sistema per riflettere il nuovo saldo punti.



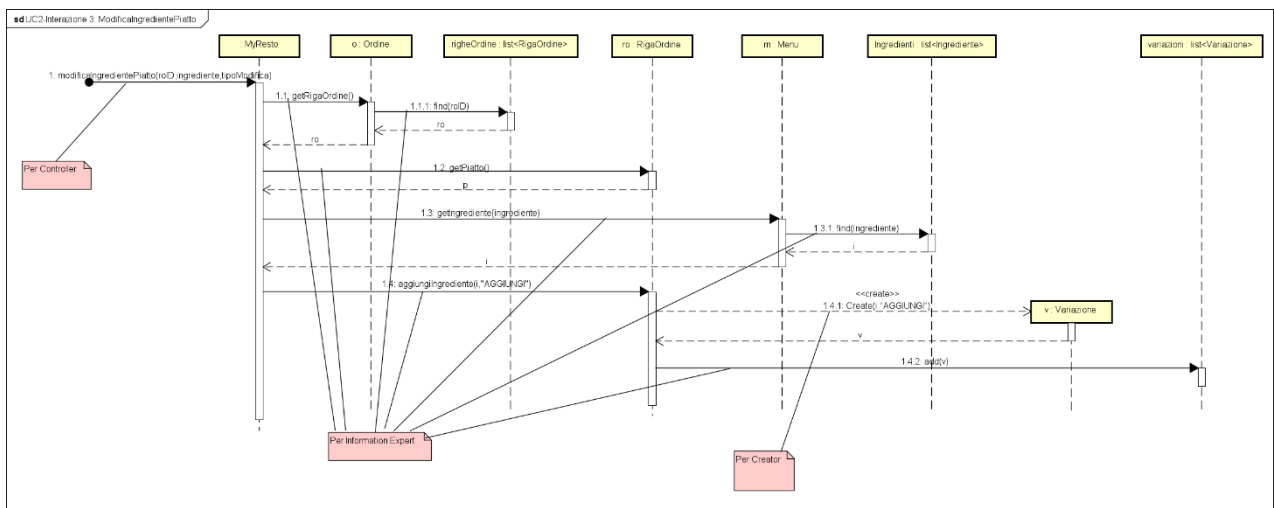
- ***visualizzaMenu***



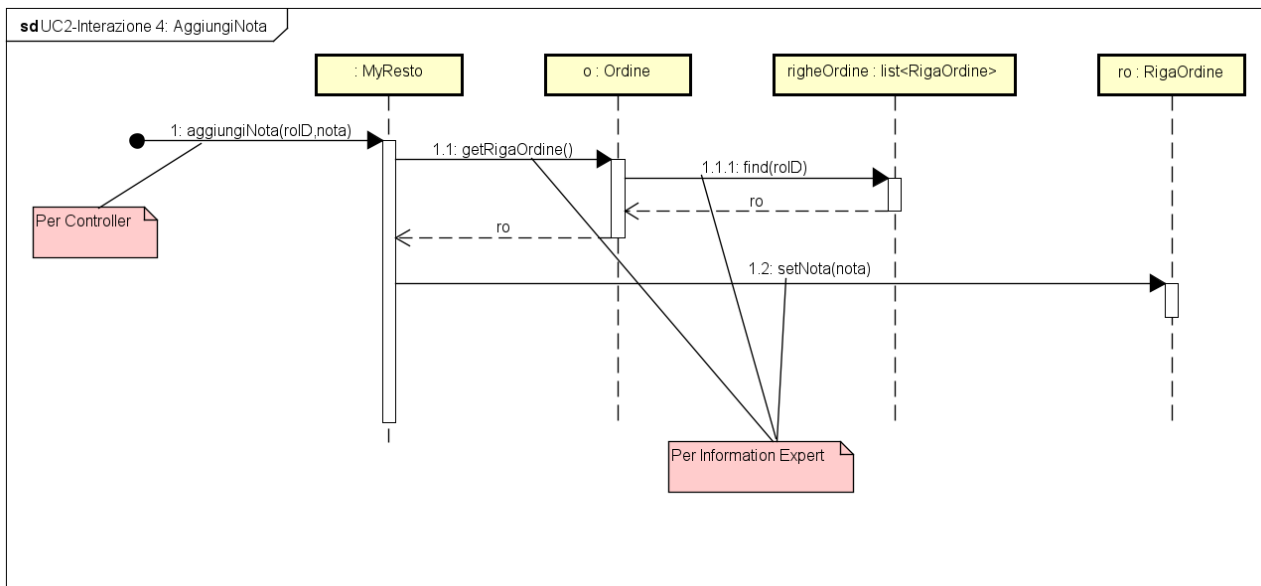
- selezionaPiatto**



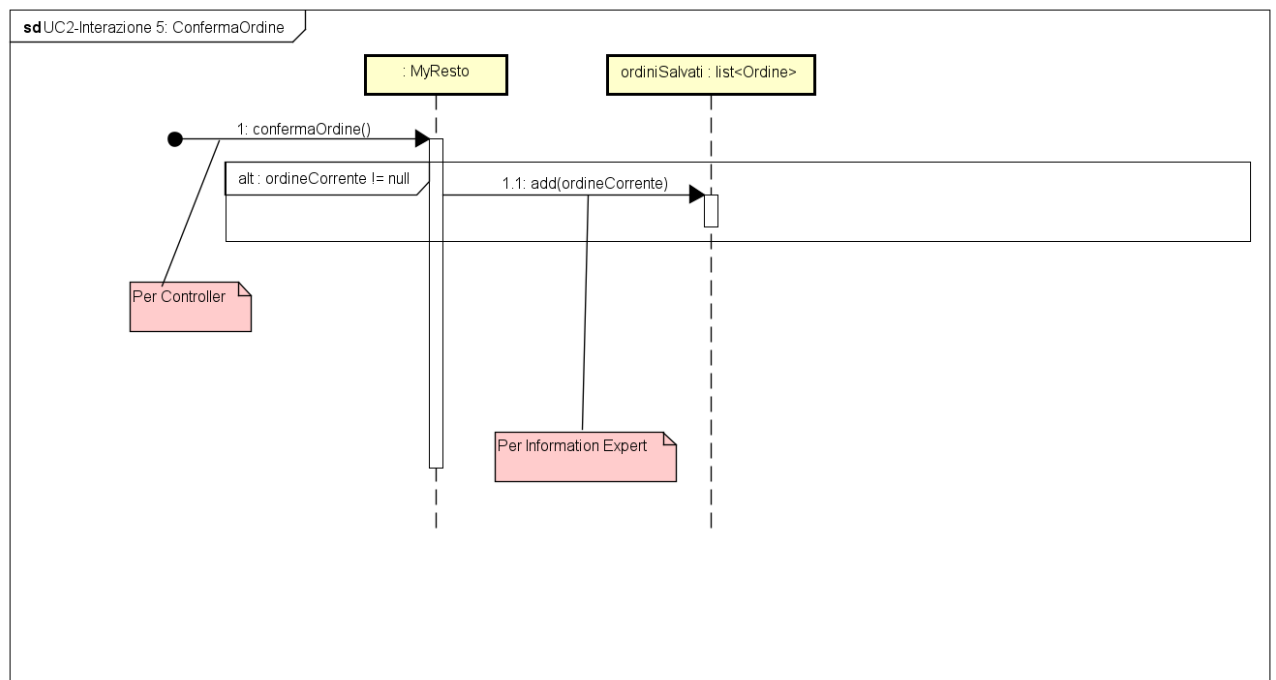
- modificaIngredientePiatto**



- aggiungiNota**



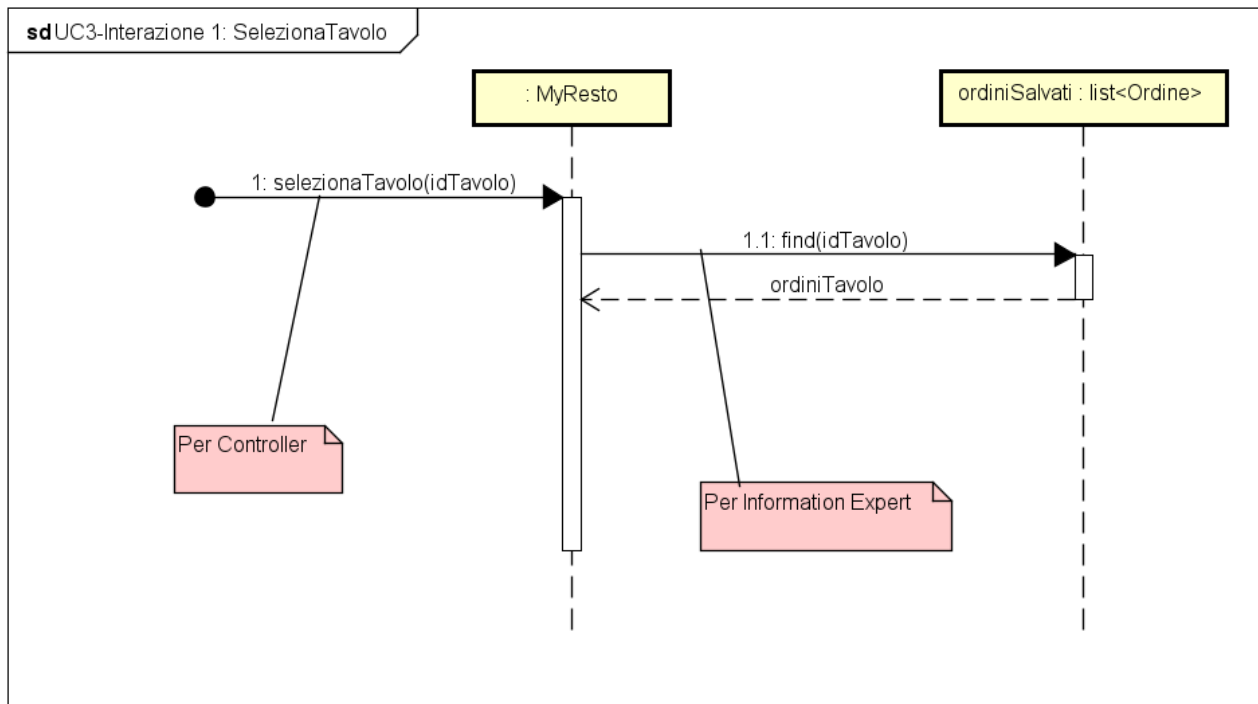
- **confermaOrdine**



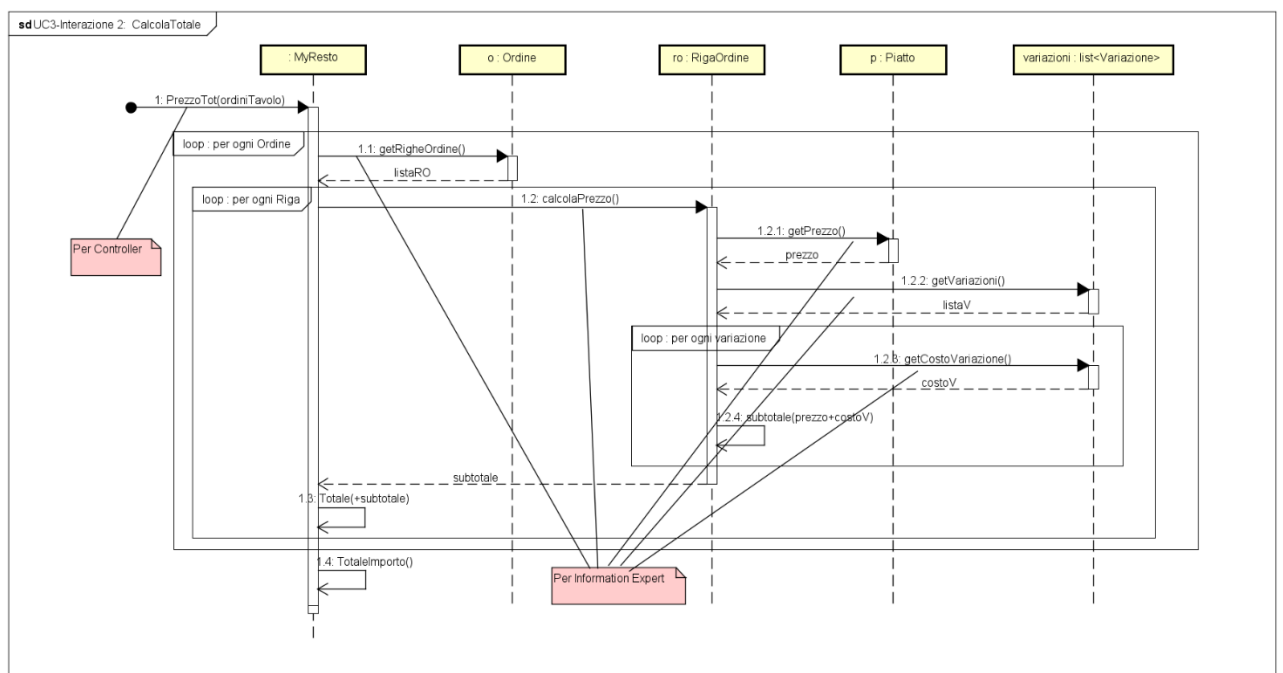
**UC3:**

- **selezionaTavolo**

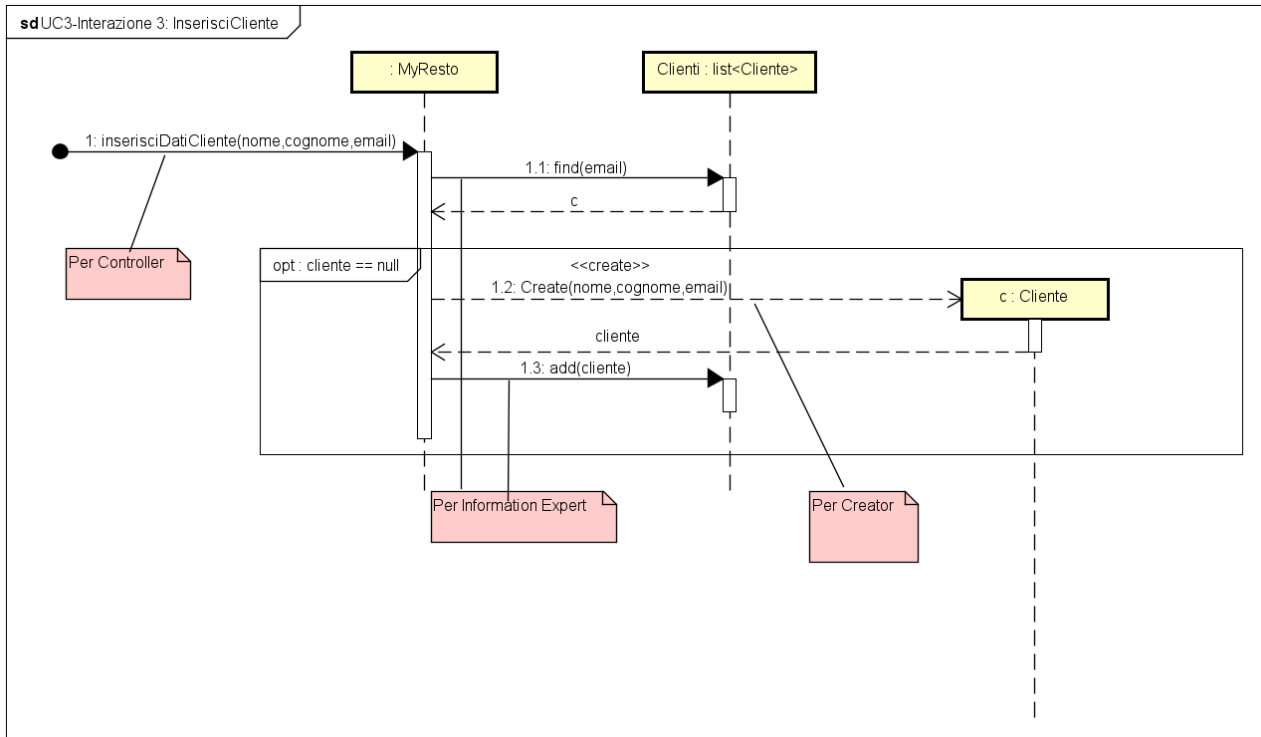




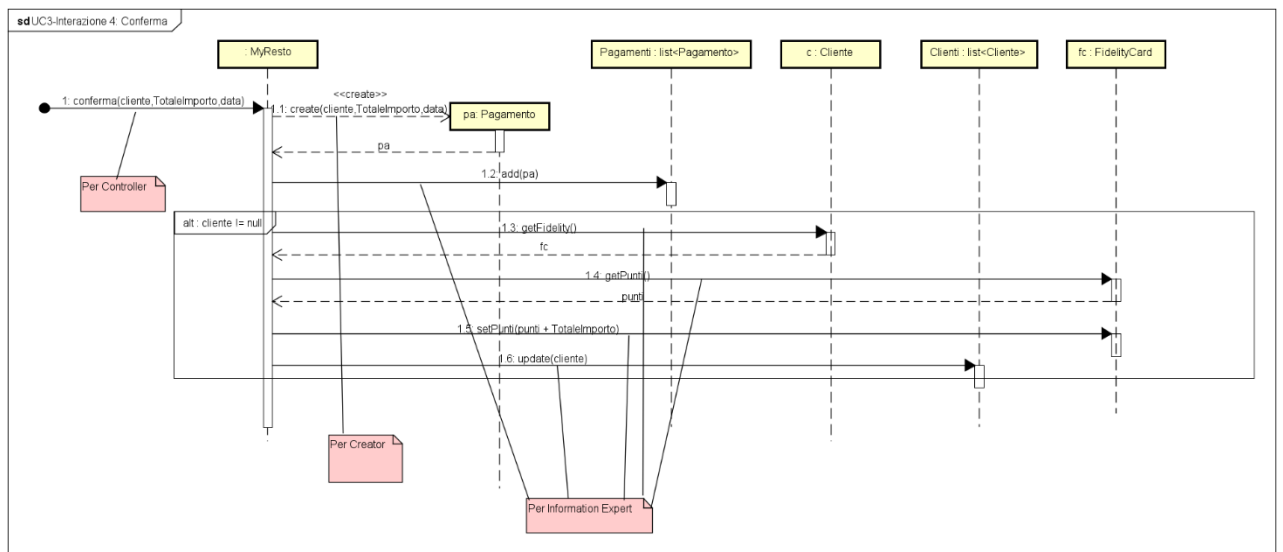
- calcolaTotale**



- inserisciCliente**

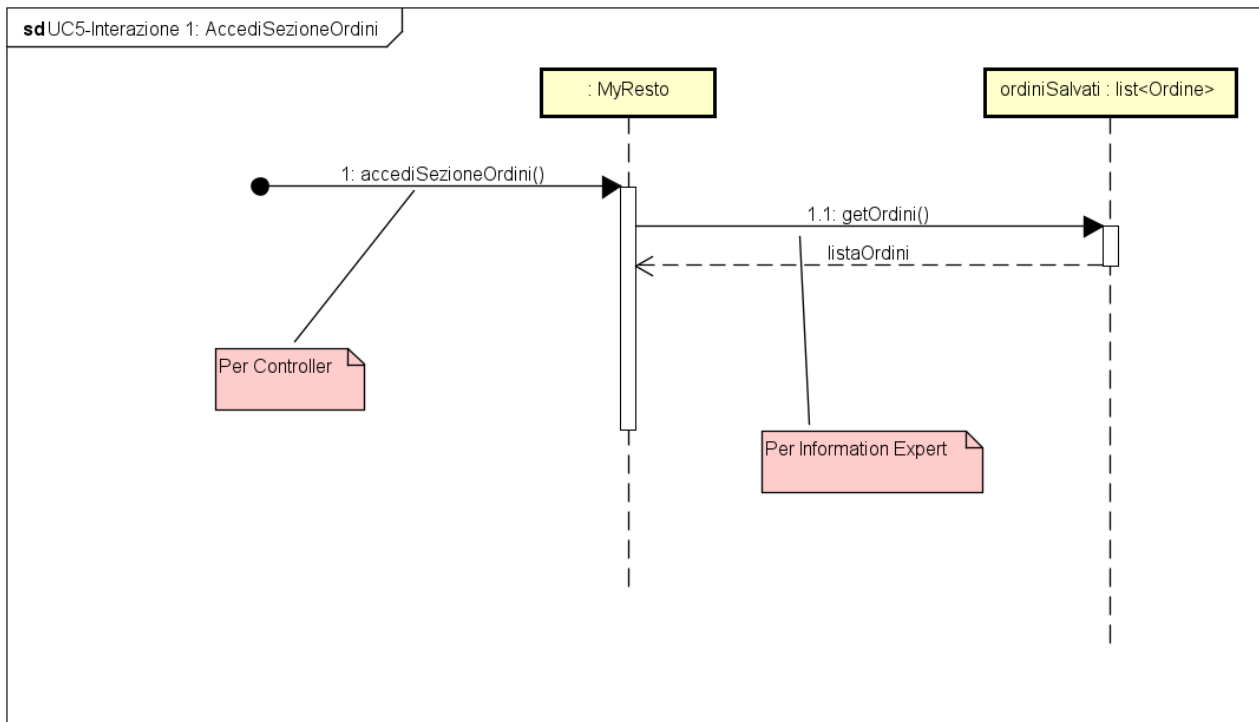


- conferma**

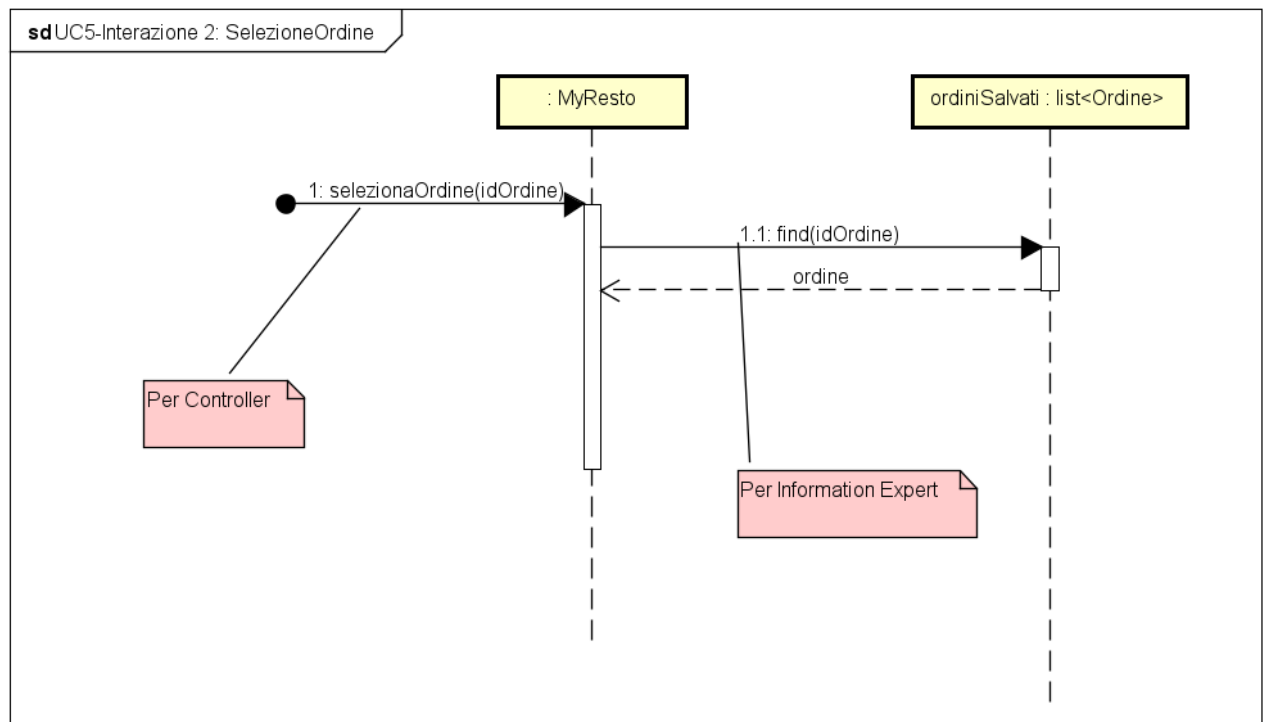


**UC5:**

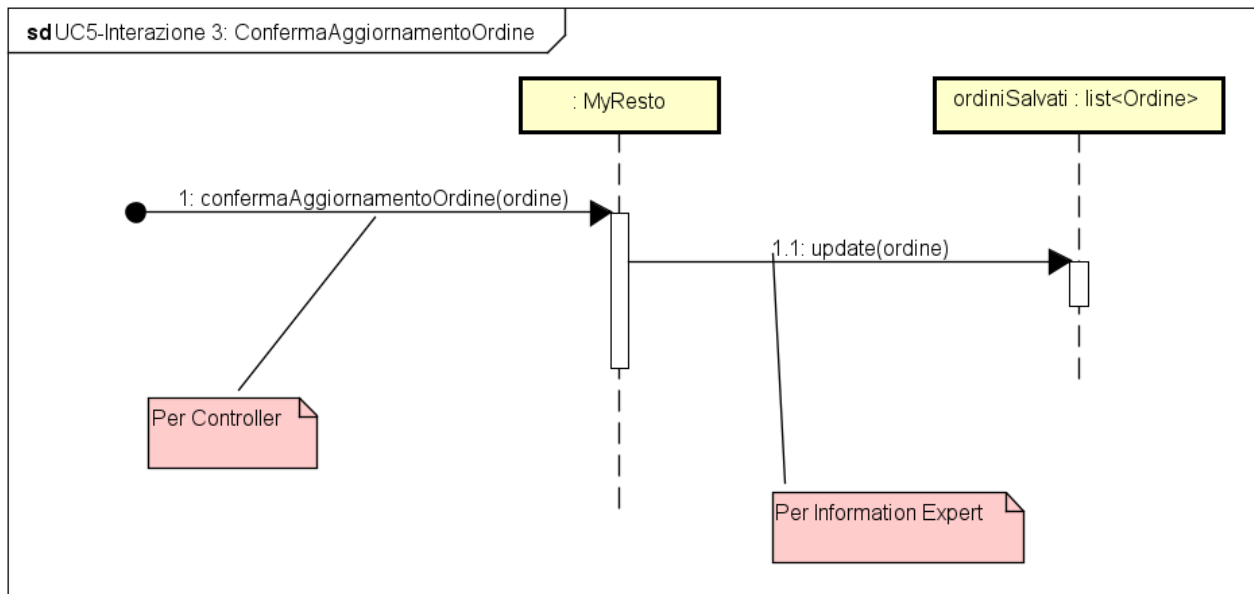
- accediSezioneOrdini**



- **selezioneOrdine**

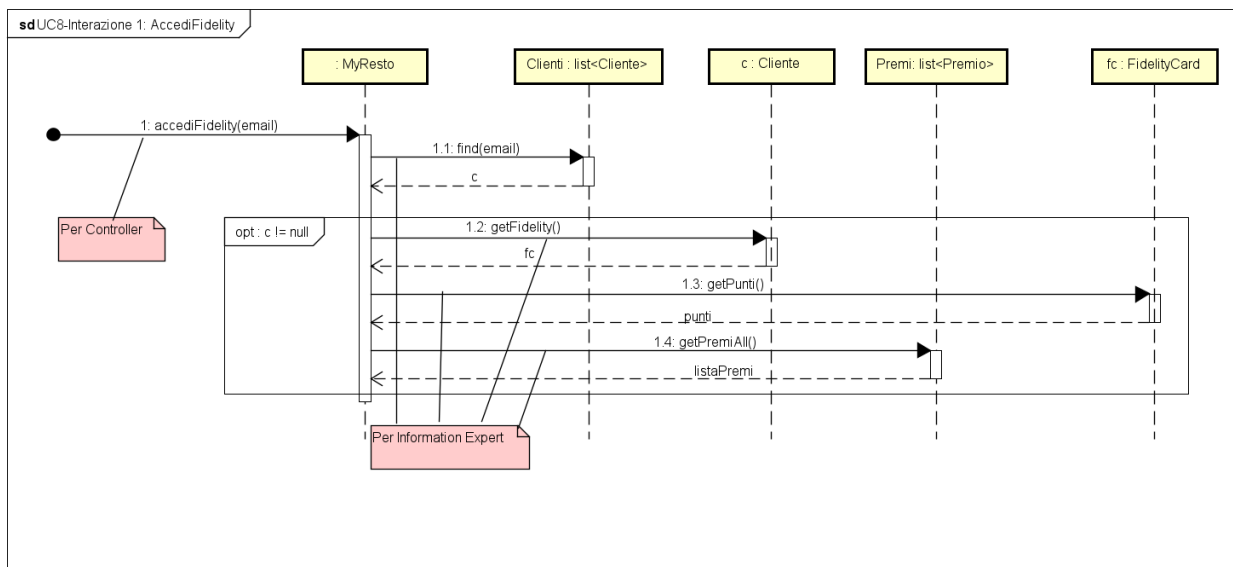


- **confermaAggiornamentoOrdine**



## UC8:

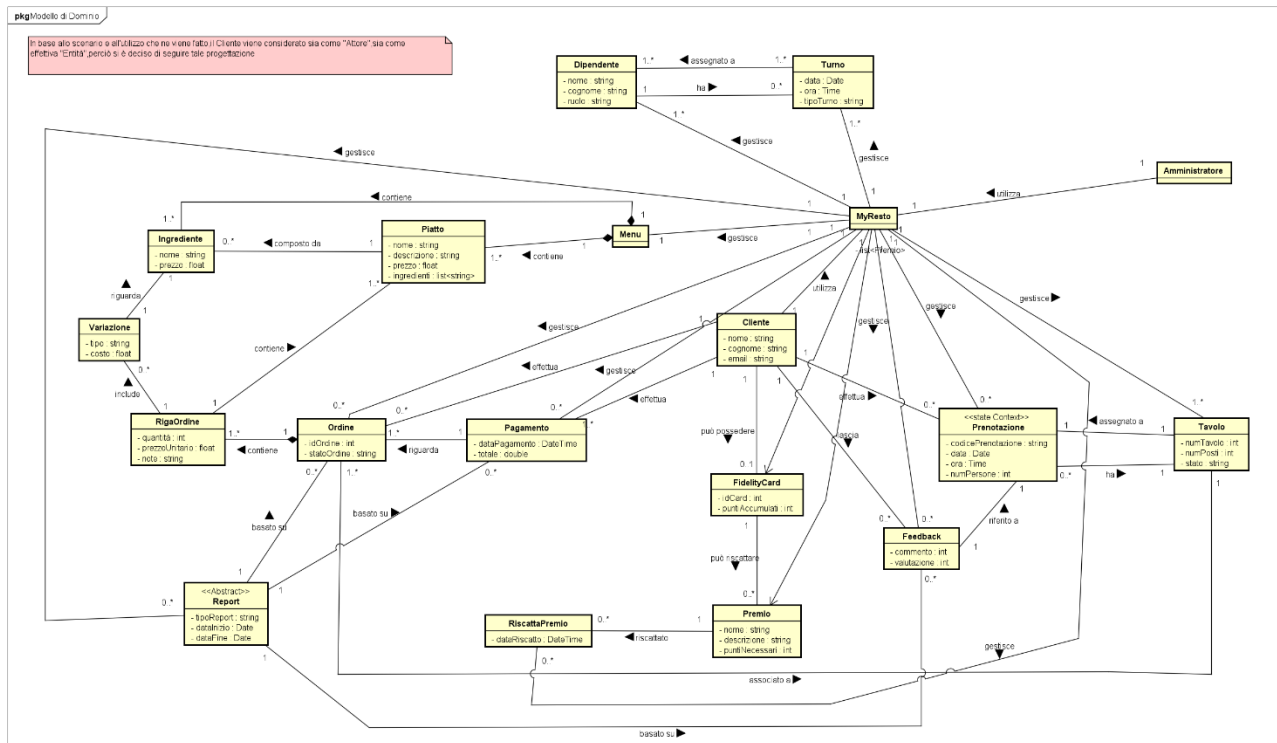
- accediFidelity**



- riscattaPremio**

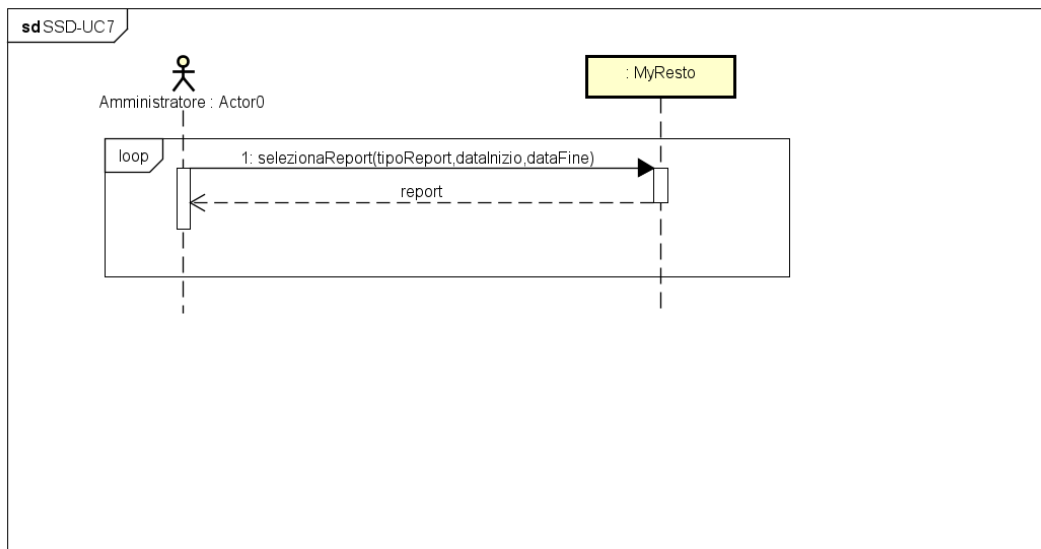


## Iterazione 3:

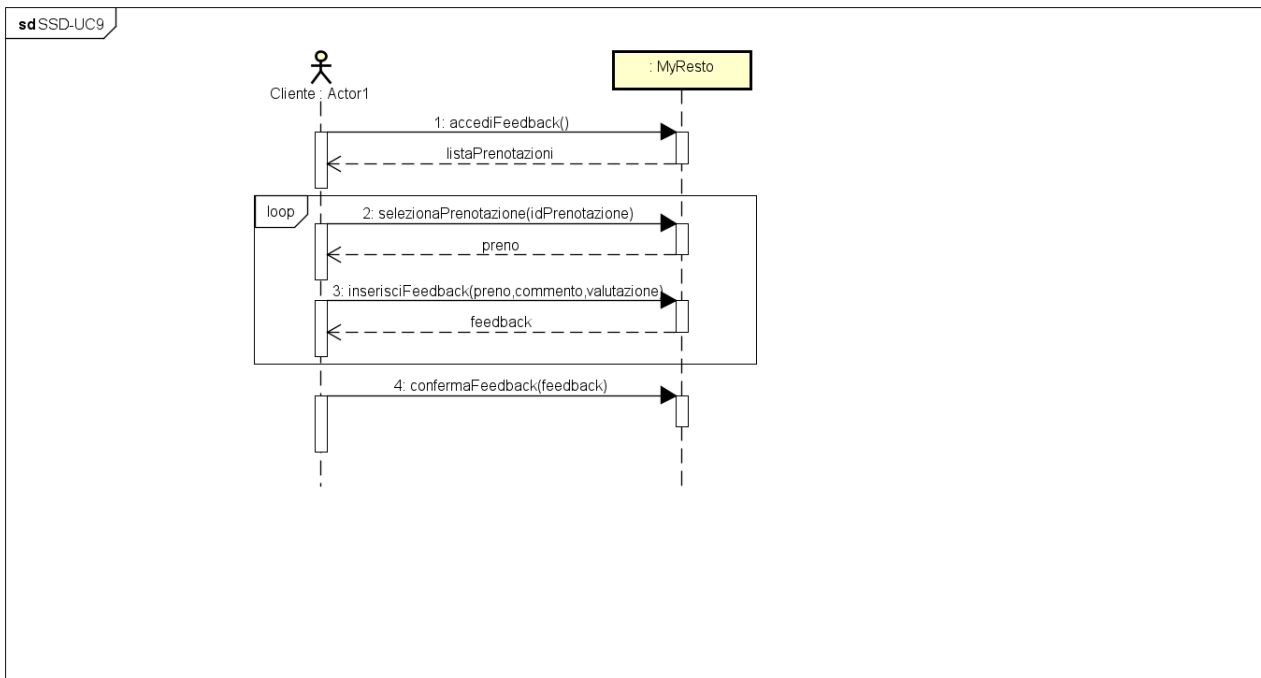


## Diagramma di sequenza di sistema

### UC7:



## UC9:



## Contratti delle operazioni

### UC7:

#### Contratto CO1: selezionaReport

**Operazione:** *selezionaReport(tipoReport: String, dataInizio: Date, dataFine: Date).*

**Riferimenti:** caso d'uso: Generazione Report.

**Pre-condizioni:** l'amministratore ha effettuato l'accesso alla sezione report.

#### **Post-condizioni:**

- Il sistema recupera i dati pertinenti all'intervallo di date e al tipoReport richiesto.
- Se tipoReport è "Feedback Clienti": vengono recuperati i Feedback dalla collezione Feedbacks:list<Feedback> nel periodo specificato. Il sistema calcola la media delle valutazioni dei feedback recuperati.
- Se tipoReport è "Piatti più ordinati": vengono recuperati gli Ordini dalla collezione ordiniSalvati:list<Ordine> nel periodo specificato. Il sistema conta le occorrenze di ogni Piatto all'interno degli ordini recuperati.
- Se tipoReport è "Ricavi": vengono recuperati i Pagamenti dalla collezione Pagamenti:list<Pagamento> nel periodo specificato. Il sistema calcola il totale dei ricavi sommando gli importi dei pagamenti recuperati.
- Una nuova istanza r di Report è stata creata con i dati elaborati.
- L'istanza r di Report è stata aggiunta alla collezione reportSalvati:list<Report>.
- Il file di testo reports.txt contiene le informazioni del report richiesto.

**UC9:**

**Contratto CO1: accediFeedback**

**Operazione:** *accediFeedback()*

**Riferimenti:** caso d'uso: Feedback Clienti.

**Pre-condizioni:** Il Cliente ha effettuato almeno un ordine.

**Post-condizioni:**

-Il sistema recupera e fornisce l'elenco delle prenotazioni recenti associate al cliente (listaPrenotazioni).

**Contratto CO2: selezionaPrenotazione**

**Operazione:** *selezionaPrenotazione(idPrenotazione: int)*

**Riferimenti:** caso d'uso: Feedback Clienti.

**Pre-condizioni:** Il sistema ha precedentemente mostrato al cliente una lista di prenotazioni recenti.

**Post-condizioni:**

-Il sistema cerca la prenotazione specificata da idPrenotazione all'interno della collezione Prenotazioni:list<Prenotazione>.

-L'istanza della prenotazione (preno) corrispondente a idPrenotazione viene recuperata.

**Contratto CO3: inserisciFeedback**

**Operazione:***inserisciFeedback(preno:Prenotazione,commento:String, valutazione:int).*

**Riferimenti:** caso d'uso: Feedback Clienti.

**Pre-condizioni:**

-Una prenotazione valida (preno) è stata selezionata dal cliente e passata all'operazione.

-Il commento e la valutazione sono stati forniti dal cliente e sono validi.

**Post-condizioni:**

-Una nuova istanza f di Feedback viene creata utilizzando i parametri preno, commento e valutazione.

-L'istanza f di Feedback viene restituita al sistema.

**Contratto CO4: confermaFeedback**

**Operazione:** *confermaFeedback(feedback: Feedback )*

**Riferimenti:** caso d'uso: Feedback Clienti.



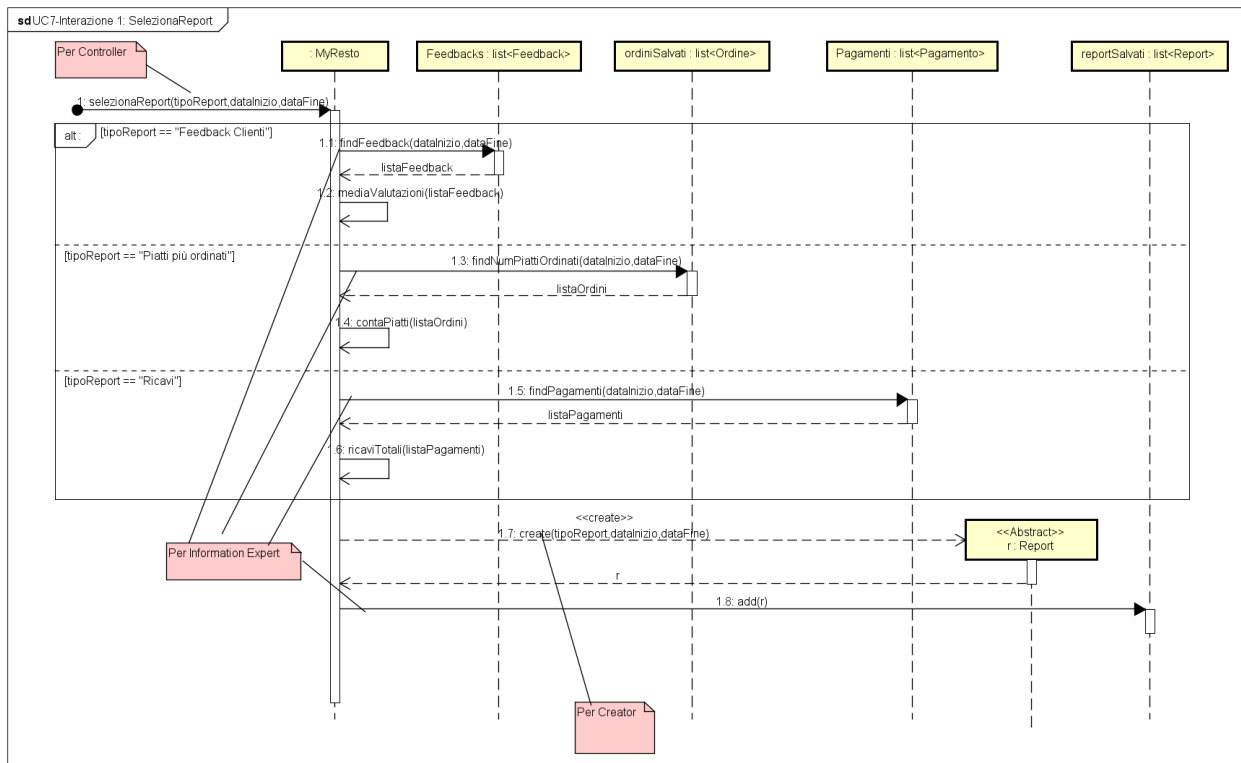
**Pre-condizioni:** Un'istanza di Feedback valida è stata creata e i suoi dati sono stati forniti dal cliente.

**Post-condizioni:** L'istanza feedback viene aggiunta alla collezione Feedbacks:list<Feedback>.

## Diagrammi di Sequenza

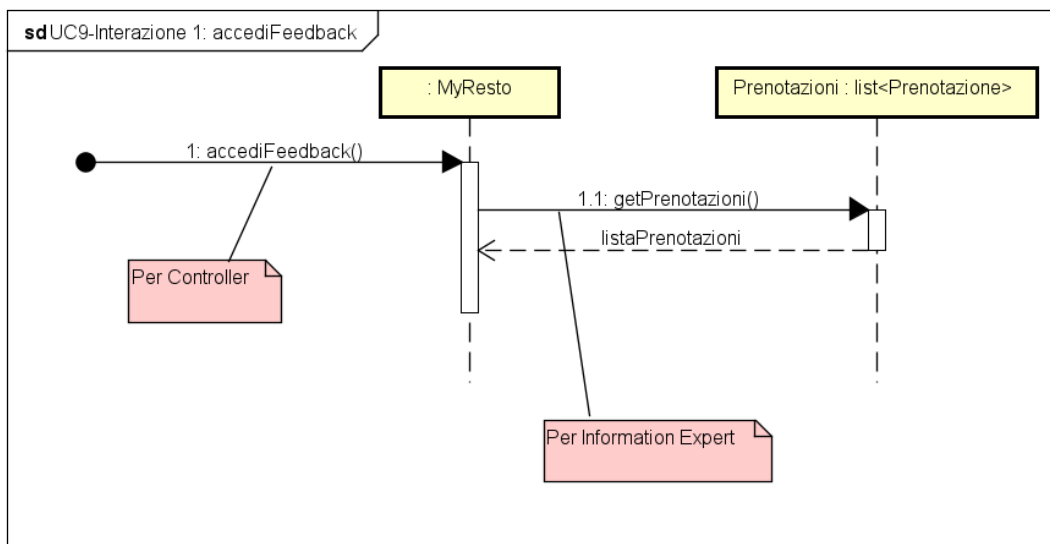
### UC7:

- selezionaReport**

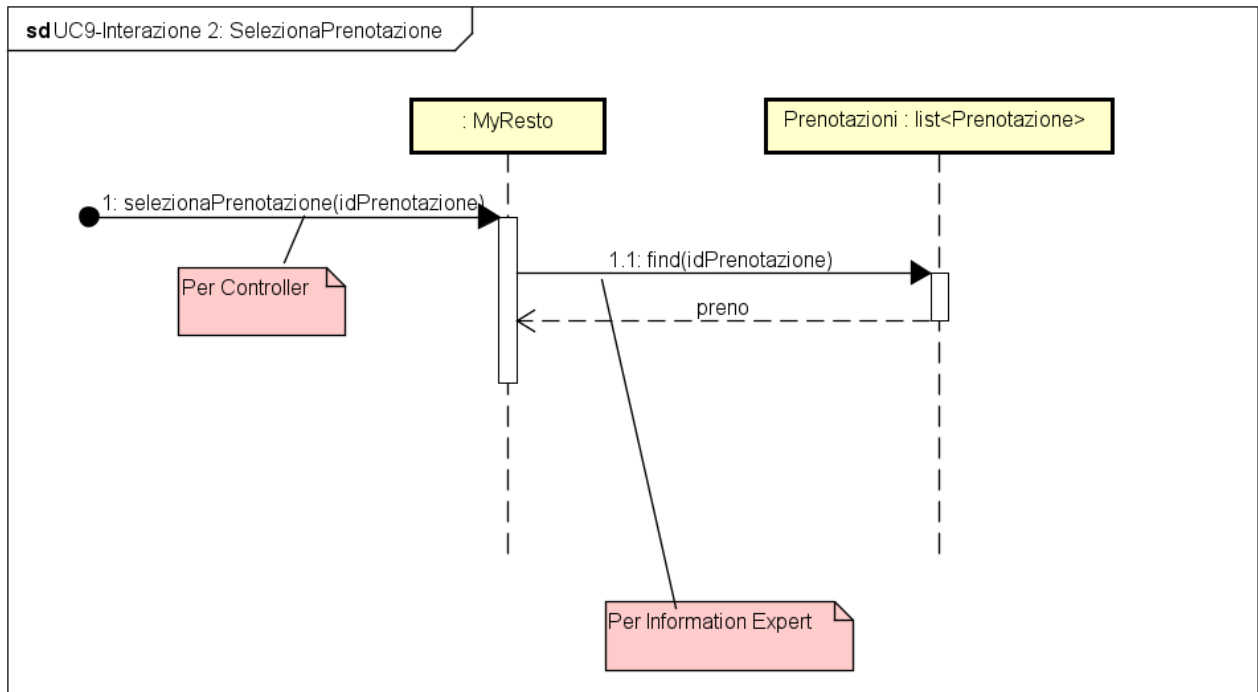


### UC9:

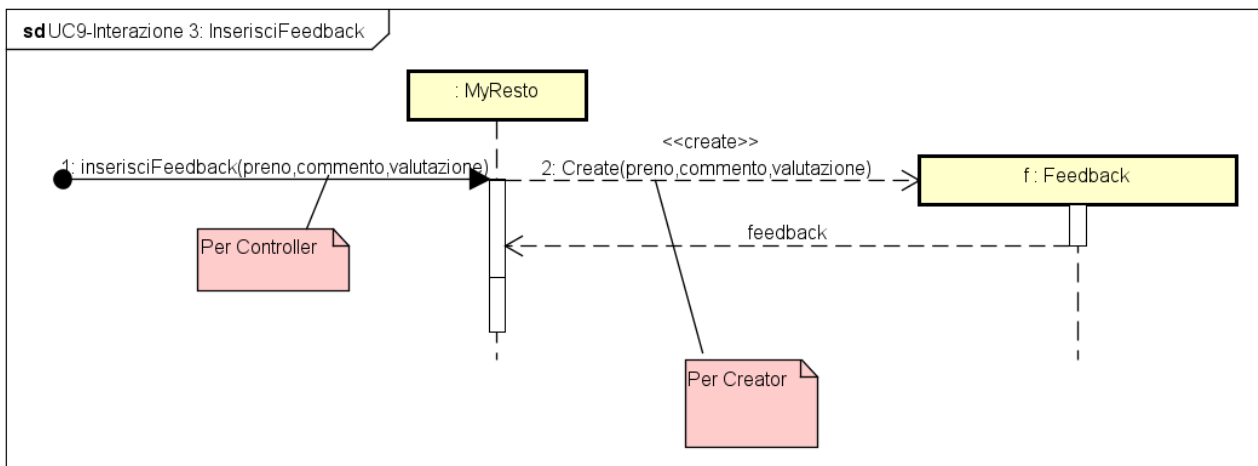
- accediFeedback**



- **selezionaPrenotazione**



- **inserisciFeedback**



- **confermaFeedback**



# TESTING

## Introduzione

Il processo di testing del progetto "MyResto" è stato implementato con un'attenzione particolare alla qualità e affidabilità delle sue componenti software.

Perché questi test siano efficaci, avranno bisogno di:

1. Repository fittizi (Mock Repositories): Invece di usare i FileBasedCrudRepository reali, che interagiscono con il file system, useremo implementazioni semplici in memoria (chiamati "mock"). Questo rende i test:

- 1.1 Isolati: Ogni test verifica solo l'unità di codice (il Service) senza dipendere dallo stato dei file.
- 1.2 Veloci: Non ci sono operazioni di I/O, quindi i test girano rapidamente.
- 1.3 Ripetibili: Ogni test parte da uno stato pulito e prevedibile.

1. Unità di Test: Ogni classe di test si concentrerà su un singolo Service (es. ClienteServiceTest, PiattoServiceTest).

Di seguito, riportamo i test suddivisi per iterazione descrivendo per ciascuno lo scopo e la realizzazione.

## Iterazione 1

### **MyRestoServiceTest.java**

#### **testRegistraNuovoCliente()**

- **Scopo:** Verificare la corretta registrazione di un nuovo cliente.
- **Realizzazione:** Simula la risposta del ClienteService alla richiesta di registrazione e verifica che il metodo registraNuovoCliente del clienteService sia chiamato una volta e che il risultato corrisponda.

#### **testCercaClientePerId()**

- **Scopo:** Verificare la ricerca di un cliente tramite ID.
- **Realizzazione:** Simula il ritrovamento di un cliente da parte del ClienteService e verifica che il metodo getClientById sia chiamato e che l'Optional contenga il cliente atteso.

### **MenuServiceTest.java**

#### **shouldGetAllPiattiFromMenu()**

- **Scopo:** Recuperare solo la lista di tutti i piatti disponibili.
- **Realizzazione:** Simula il ritorno di una lista di piatti dal repository e verifica che il servizio restituisca solo i piatti attesi.

### **shouldGetAllIngredientiFromMenu()**

- **Scopo:** Recuperare solo la lista di tutti gli ingredienti disponibili.
- **Realizzazione:** Simula il ritorno di una lista di ingredienti dal repository e verifica che il servizio restituisca solo gli ingredienti attesi.

### **PiattoServiceTest.java**

#### **testCreaPiatto\_SuccessoConIngredienti()**

- **Scopo:** Creare un piatto con ingredienti validi.
- **Realizzazione:** Simula la ricerca di ingredienti e il salvataggio del piatto, poi verifica che il piatto creato contenga gli ingredienti e che i repository siano stati chiamati correttamente.

#### **testCreaPiatto\_NomeDuplicato\_ThrowsException()**

- **Scopo:** Verificare l'eccezione per la creazione di un piatto con un nome già esistente.
- **Realizzazione:** Simula che findByNome trovi un piatto esistente e asserisce l'eccezione `IllegalArgumentException`.

### **IngredienteServiceTest.java**

#### **testAggiungiNuovoIngrediente\_Successo()**

- **Scopo:** Aggiungere un nuovo ingrediente valido al sistema.
- **Realizzazione:** Simula che l'ingrediente non esista, che venga generato un nuovo ID e salvato. Verifica che l'ingrediente ritornato sia corretto e che le interazioni con il repository siano avvenute.

#### **testAggiungiNuovoIngrediente\_PrezzoNegativo\_ThrowsException()**

- **Scopo:** Verificare l'eccezione quando si tenta di aggiungere un ingrediente con prezzo negativo.
- **Realizzazione:** Tenta l'operazione con prezzo negativo e asserisce il lancio di `IllegalArgumentException` senza interazioni con il repository.

### **PrenotazioneServiceTest.java**

#### **testCreaPrenotazione\_Successo\_TavoloAssegnato()**

- **Scopo:** Creare con successo una prenotazione assegnando un tavolo specifico.

- **Realizzazione:** Simula la disponibilità del cliente e del tavolo, il non-conflitto di orario e il salvataggio. Verifica che la prenotazione sia creata correttamente e il tavolo riservato.

#### **testCreaPrenotazione\_ClienteNonTrovato\_ThrowsException()**

- **Scopo:** Verificare l'eccezione se il cliente per la prenotazione non esiste.
- **Realizzazione:** Simula la non-trovabilità del cliente e asserisce `IllegalArgumentException`.

#### **TavoloServiceTest.java**

##### **testCreaTavolo\_Successo()**

- **Scopo:** Creare con successo un nuovo tavolo.
- **Realizzazione:** Simula che non esista già un tavolo con lo stesso numero e che il tavolo venga salvato. Verifica che il tavolo creato abbia le proprietà corrette e che i metodi del repository siano chiamati.

##### **testGetAllTavoli\_ListaNonVuota()**

- **Scopo:** Recuperare una lista di tutti i tavoli quando la lista non è vuota.
- **Realizzazione:** Simula il ritorno di una lista predefinita di tavoli dal repository e verifica che la lista restituita non sia nulla, non sia vuota e contenga i tavoli attesi.

#### **DipendenteServiceTest.java**

##### **testAggiungiNuovoDipendente\_Successo()**

- **Scopo:** Verificare l'aggiunta di un nuovo dipendente con successo.
- **Realizzazione:** Simula la generazione di un nuovo ID e il salvataggio del dipendente nel repository, poi verifica che il dipendente restituito sia corretto.

##### **testGetDipendenteById\_NonEsistente()**

- **Scopo:** Verificare che nessun dipendente venga trovato per un ID inesistente.
- **Realizzazione:** Simula il non-ritrovamento di un dipendente dal repository per un ID dato e verifica che l'`Optional` sia vuoto.

#### **TurnoServiceTest.java**

##### **testAssegnaTurno\_Successo()**

- **Scopo:** Assegnare un nuovo turno a un dipendente con successo.

- **Realizzazione:** Simula la disponibilità del dipendente e l'assenza di turni sovrapposti per quella data. Verifica che il turno sia creato correttamente, con ID, dipendente, data, ora e tipo turno attesi.

#### **testAssegnaTurno\_DipendenteNonTrovato\_ThrowsException()**

- **Scopo:** Verificare l'eccezione quando il dipendente a cui assegnare il turno non esiste.
- **Realizzazione:** Simula che il DipendenteRepository non trovi il dipendente specificato e asserisce il lancio di IllegalArgumentException.

### Iterazione 2

#### **OrdineServiceTest.java**

##### **testCreaNuovoOrdine\_Successo()**

- **Scopo:** Verificare la creazione di un nuovo ordine con successo.
- **Realizzazione:** Simula che il cliente e il tavolo siano trovati, che il tavolo sia libero e che l'ordine venga salvato. Verifica che l'ordine creato abbia le proprietà corrette e che lo stato del tavolo cambi a "OCCUPATO".

##### **testCreaNuovoOrdine\_TavoloGiaOccupato\_ThrowsException()**

- **Scopo:** Assicurare che un'eccezione venga lanciata se il tavolo è già occupato.
- **Realizzazione:** Simula che il tavolo sia già in stato "OCCUPATO" e asserisce il lancio di IllegalStateException.

#### **RigaOrdineServiceTest.java**

##### **testCreaRigaOrdine\_Successo\_ConVariazioni()**

- **Scopo:** Verificare la creazione di una riga d'ordine con successo, includendo variazioni.
- **Realizzazione:** Simula il recupero di ordine, piatto e variazioni dal repository. Verifica che la riga d'ordine venga creata con i dati corretti, incluse le variazioni, e che venga aggiunta all'ordine.

##### **testCreaRigaOrdine\_OrdineNonTrovato\_ThrowsException()**

- **Scopo:** Assicurare che un'eccezione venga lanciata se l'ordine specificato non viene trovato.
- **Realizzazione:** Simula la non-trovabilità dell'ordine e asserisce il lancio di IllegalArgumentException.

#### **VariazioneServiceTest.java**

##### **testCreaVariazione\_Successo\_CostoPositivo()**

- **Scopo:** Creare con successo una nuova variazione con un costo positivo.
- **Realizzazione:** Simula l'ottenimento di un nuovo ID dal repository e il salvataggio della variazione. Verifica che l'oggetto creato abbia l'ID, il tipo e il costo attesi e che non sia associato a una riga d'ordine.

#### **testCreaVariazione\_Successo\_CostoNegativo()**

- **Scopo:** Creare con successo una nuova variazione con un costo negativo (es. per uno sconto).
- **Realizzazione:** Simula l'ottenimento di un nuovo ID e il salvataggio. Verifica che la variazione sia creata correttamente con il costo negativo.
- **testGetAllVariazioni\_ListaNonVuota()**
- **Scopo:** Recuperare una lista non vuota di tutte le variazioni.
- **Realizzazione:** Simula il ritorno di una lista predefinita di variazioni dal repository e verifica che la lista risultante sia corretta e non vuota.

#### **PagamentoServiceTest.java**

##### **testRegistraPagamento\_SuccessoConFidelityCard()**

- **Scopo:** Registrare con successo un pagamento associato a un ordine e una Fidelity Card, assicurando l'assegnazione dei punti.
- **Realizzazione:** Simula il ritrovamento dell'ordine e della Fidelity Card. Verifica che il pagamento venga creato correttamente, che i punti vengano aggiunti alla Fidelity Card e che lo stato dell'ordine sia aggiornato a "COMPLETATO".

##### **testRegistraPagamento\_SuccessoSenzaFidelityCard()**

- **Scopo:** Registrare con successo un pagamento senza l'utilizzo di una Fidelity Card.
- **Realizzazione:** Simula il ritrovamento dell'ordine. Verifica che il pagamento sia creato correttamente senza Fidelity Card e che lo stato dell'ordine sia aggiornato a "COMPLETATO".

#### **ClienteServiceTest.java**

##### **shouldThrowExceptionIfEmailsInvalidOnRegistration()**

- **Scopo:** Assicurare che venga lanciata un'eccezione se l'email fornita non è valida.
- **Realizzazione:** Esegue due test, uno con email nulla e uno con email senza il simbolo "@", e asserisce il lancio di IllegalArgumentException con il messaggio atteso.

##### **shouldFindClienteById()**

- **Scopo:** Trovare un cliente esistente tramite il suo ID.



- **Realizzazione:** Registra un cliente, poi cerca il cliente tramite il suo ID e verifica che l'Optional contenga il cliente corretto.

### **FidelityCardServiceTest.java**

#### **testCreaFidelityCardPerCliente\_Successo()**

- **Scopo:** Verificare la creazione di una nuova Fidelity Card per un cliente esistente.
- **Realizzazione:** Simula il ritrovamento del cliente e la non esistenza di una card precedente. Asserisce che la card venga creata con l'ID corretto, associata al cliente e con 0 punti.

#### **testCreaFidelityCardPerCliente\_ClienteNonTrovato\_ThrowsException()**

- **Scopo:** Assicurare che venga lanciata un'eccezione se il cliente per cui si vuole creare la card non viene trovato.
- **Realizzazione:** Simula la non-trovabilità del cliente e asserisce il lancio di IllegalArgumentException.

### **PremioServiceTest.java**

#### **testCreaNuovoPremio\_Successo()**

- **Scopo:** Verificare la corretta creazione di un nuovo premio.
- **Realizzazione:** Simula la non esistenza di un premio con lo stesso nome e l'ottenimento di un nuovo ID. Asserisce che il premio venga creato con l'ID, nome, descrizione e punti attesi.

#### **testCreaNuovoPremio\_NomeNonValido\_ThrowsException()**

- **Scopo:** Assicurare che venga lanciata un'eccezione se il nome del premio è nullo, vuoto o fatto di soli spazi.
- **Realizzazione:** Esegue tre test con nomi non validi e asserisce il lancio di IllegalArgumentException con il messaggio atteso.

### **PremioRiscattatoServiceTest.java**

#### **testRegistraRiscossionePremio\_Successo()**

- **Scopo:** Verificare la corretta registrazione di un premio riscattato.
- **Realizzazione:** Simula il ritrovamento del cliente e del premio, e l'ottenimento di un nuovo ID. Asserisce che l'oggetto PremioRiscattato venga creato con gli attributi corretti (ID, cliente, premio, data di riscatto).

#### **testRegistraRiscossionePremio\_ClienteNonTrovato\_ThrowsException()**

- **Scopo:** Assicurare che un'eccezione venga lanciata se il cliente che tenta di riscattare il premio non viene trovato.
- **Realizzazione:** Simula la non-trovabilità del cliente e asserisce il lancio di `IllegalArgumentException` con il messaggio atteso.

### Iterazione 3

#### **FeedbackServiceTest.java**

##### **testAggiungiFeedback\_ConPiatto\_Successo()**

- **Scopo:** Aggiungere con successo un feedback associato a un cliente e a un piatto.
- **Realizzazione:** Simula il ritrovamento del cliente e del piatto. Asserisce che il feedback venga creato con gli attributi corretti (commento, valutazione, ID cliente e ID piatto).

##### **testAggiungiFeedback\_CommentoVuoto\_ThrowsException()**

- **Scopo:** Assicurare che venga lanciata un'eccezione se il commento del feedback è vuoto.
- **Realizzazione:** Tenta di aggiungere un feedback con un commento vuoto e asserisce il lancio di `IllegalArgumentException`.

#### **ReportServiceTest.java**

##### **testGeneraReportIncassi\_Successo\_ConPagamenti()**

- **Scopo:** Generare un report degli incassi con pagamenti registrati nel periodo.
- **Realizzazione:** Simula l'esistenza di pagamenti nel periodo specificato. Asserisce che il report venga creato con l'ID, tipo, date e un contenuto che riflette l'incasso totale e il numero di pagamenti.

##### **testGeneraReportIncassi\_Successo\_SenzaPagamenti()**

- **Scopo:** Generare un report degli incassi quando non ci sono pagamenti nel periodo.
- **Realizzazione:** Simula l'assenza di pagamenti nel periodo. Asserisce che il report venga creato con incasso totale pari a zero e zero pagamenti.

## **Istruzioni di Configurazione ed Esecuzione di MyResto**

Per avviare e testare l'applicativo MyResto, ecco i seguenti passaggi.

### **1. Prerequisiti**

È necessaria l'installazione di:

- Java Development Kit (JDK) 21
- Apache Maven 3.x
- IntelliJ IDEA.

### **2. Configurazione del Progetto**

1. Apri il progetto in IntelliJ IDEA.

### **3. Compilazione ed Installazione delle Dipendenze**

Apri il terminale integrato di IntelliJ IDEA ed esegui il seguente comando: *mvn clean install*.

#### 4. Esecuzione dell'Applicativo

Una volta completato *mvn clean install*, puoi eseguire l'applicativo:

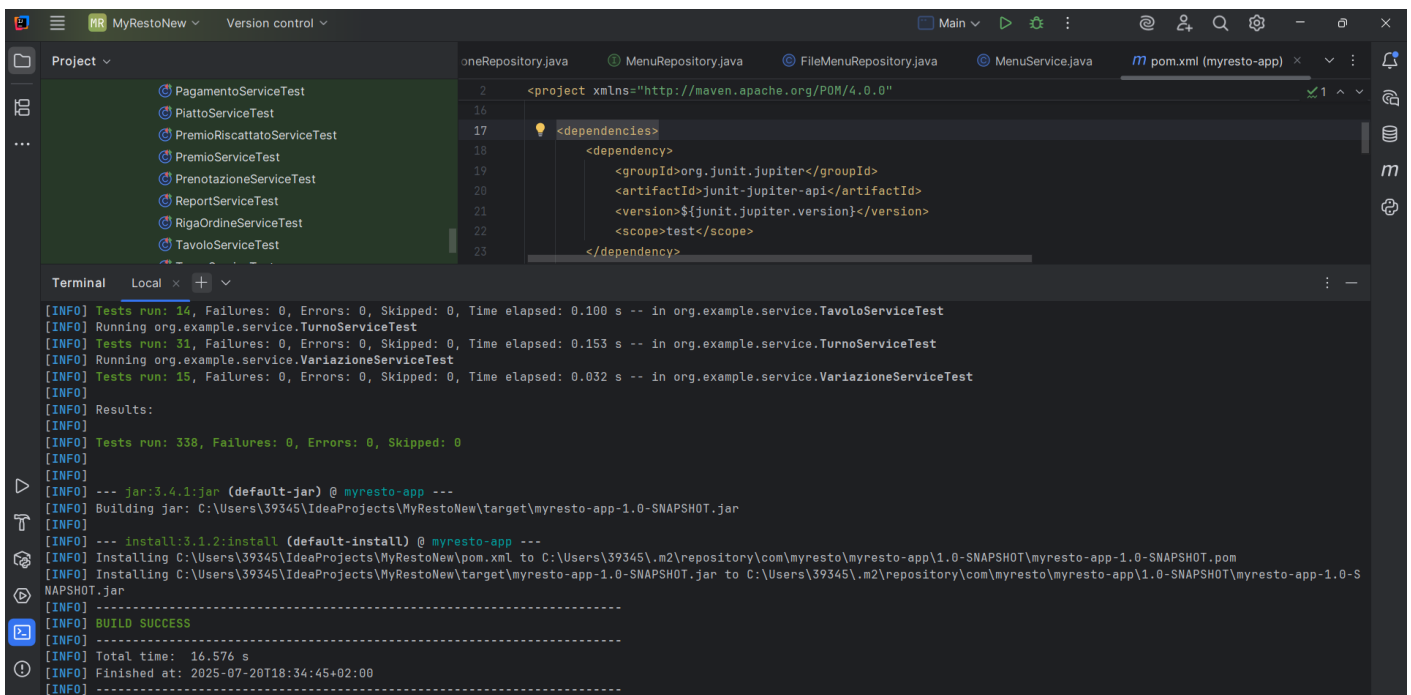
1. In IntelliJ IDEA, naviga al file: `src/main/java/org/example/Main.java`.
2. Utilizza il pulsante di esecuzione verde e seleziona “Run 'Main'”.

#### 5. Credenziali Amministratore

All'inizio dell'esecuzione verrà richiesta l'autenticazione come amministratore per sbloccare determinate funzionalità (come la creazione di piatti, l'aggiunta di dipendenti, ecc.).

Le credenziali per accedere come amministratore sono:

- Username: admin



The screenshot shows the IntelliJ IDEA interface with a Maven project named 'myresto-app'. The 'pom.xml' file is open, showing a dependency on 'junit-jupiter-api'. The terminal window displays the output of the 'mvn clean install' command, including test results and the final 'BUILD SUCCESS' message. The test results show that all tests passed successfully.

```
[INFO] Tests run: 14, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.100 s -- in org.example.service.TavoloServiceTest
[INFO] Running org.example.service.TurnoServiceTest
[INFO] Tests run: 31, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.153 s -- in org.example.service.TurnoServiceTest
[INFO] Running org.example.service.VariazioneServiceTest
[INFO] Tests run: 15, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.032 s -- in org.example.service.VariazioneServiceTest
[INFO] Results:
[INFO] Tests run: 338, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] --- jar:3.4.1:jar (default-jar) @ myresto-app ---
[INFO] Building jar: C:\Users\39345\IdeaProjects\MyRestoNew\target\myresto-app-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- install:3.1.2:install (default-install) @ myresto-app ---
[INFO] Installing C:\Users\39345\IdeaProjects\MyRestoNew\pom.xml to C:\Users\39345\.m2\repository\com\myresto\myresto-app\1.0-SNAPSHOT\myresto-app-1.0-SNAPSHOT.pom
[INFO] Installing C:\Users\39345\IdeaProjects\MyRestoNew\target\myresto-app-1.0-SNAPSHOT.jar to C:\Users\39345\.m2\repository\com\myresto\myresto-app\1.0-SNAPSHOT\myresto-app-1.0-SNAPSHOT.jar
[INFO] BUILD SUCCESS
[INFO] Total time: 16.576 s
[INFO] Finished at: 2025-07-20T18:34:45+02:00
[INFO]
```

- Password: adminpass

L'applicazione procederà poi a mostrare le varie funzionalità, salvando i dati su file di testo nella directory `src/main/resources/data/`.

Benvenuto in MyResto!

--- Accesso Amministratore ---

Username: *admin*

Password: *adminpass*

Autenticazione amministratore riuscita! Benvenuto, admin.

--- Registrazione Cliente ---

Cliente registrato: Alice Bianchi (ID: 0)

--- Creazione Fidelity Card per Cliente ---

Fidelity Card creata per Alice (ID Card: 0, Punti: 0)

--- Creazione Tavoli ---

Tavoli creati: 101, 102

Tavoli disponibili per 3 persone: 1

--- Funzionalità Amministrative (Richiede Autenticazione) ---

--- Aggiunta Dipendente ---

Dipendente aggiunto: Luca Neri (ID: 0)

--- Assegnazione Turno ---

Turno assegnato: Turno{id=0, dipendente=Luca, dataInizio=2025-07-20, dataFine=2025-07-27, tipoTurno='Mattina'}

--- Creazione Piatti e Ingredienti ---

Piatto creato: Pizza Margherita (ID: 0)

Piatto creato: Spaghetti alla Carbonara (ID: 1)

```
--- Creazione Piatti e Ingredienti ---
Piatto creato: Pizza Margherita (ID: 0)
Piatto creato: Spaghetti alla Carbonara (ID: 1)
Piatto creato: Risotto ai Funghi Porcini (ID: 2)
Piatto creato: Insalata di Salmone e Avocado (ID: 3)
Piatto creato: Pollo al Forno con Patate (ID: 4)
Piatto creato: Tiramisù (ID: 5)
```

```
--- Visualizzazione Menu Completo ---
```

```
Numero piatti nel menu: 6
```

- Pizza Margherita (€8.5)
- Spaghetti alla Carbonara (€12.0)
- Risotto ai Funghi Porcini (€15.0)
- Insalata di Salmone e Avocado (€14.5)
- Pollo al Forno con Patate (€13.0)
- Tiramisù (€6.0)

```
Numero ingredienti disponibili: 23
```

- Pomodoro
- Mozzarella
- Basilico
- Pasta
- Uova
- Pancetta
- Pecorino Romano
- Pepe Nero
- Riso Arborio
- Funghi Porcini

```
--- Creazione Premi ---
```

```
Premi creati: Caffè Omaggio (2 punti), Sconto 10 Euro (500 punti)
```

```
--- Funzionalità Generali (Non Richiede Autenticazione Admin) ---
```

```
--- Creazione Ordine ---
```

```
Ordine creato (ID: 0) per Tavolo: 101
```

```
--- Aggiunta Righe Ordine ---
```

```
Pizze aggiunte all'ordine.
```

```
Aggiunto Tiramisù all'ordine.
```

```
--- Aggiornamento Stato Ordine ---
```

```
Stato ordine 0 aggiornato a 'Servito'.
```

```
--- Totale Ordine ---
```

```
Totale Ordine 0: 34,50 EUR
```

```
--- Registrazione Pagamento ---
```

```
Pagamento registrato (ID: 0) per Ordine 0
```

```
Stato Ordine dopo pagamento: Completato
```

```
--- Punti Fidelity Card dopo Pagamento ---
```

```
Punti Fidelity Card per Alice: 3 punti.
```

```
--- Lascia Feedback per i piatto consumati ---
Feedback aggiunto da Alice per 'Pizza Margherita': 5/5 - "Ottima pizza, impasto leggero!"

--- Riscatto Premi ---
Tentativo di riscatto: Caffè Omaggio (2 punti)
Premio riscattato con successo! Caffè Omaggio per Alice.
Nuovi punti Fidelity Card per Alice: 1 punti.
--- Report Incassi ---
Periodo: dal 2025-07-13 al 2025-07-20
Numero di Pagamenti: 1
Incasso Totale: 34.50 EUR
-----
--- Report Piatti Più Venduti ---
Periodo: dal 2025-07-13 al 2025-07-20
-----
Pizza Margherita: 3 unità
Tiramisù: 1 unità
-----

--- Fine ---

Process finished with exit code 0
```