JourneyHouse

Indice

1. Ideazione ed analisi dei requisiti

Idea di base e requisiti:

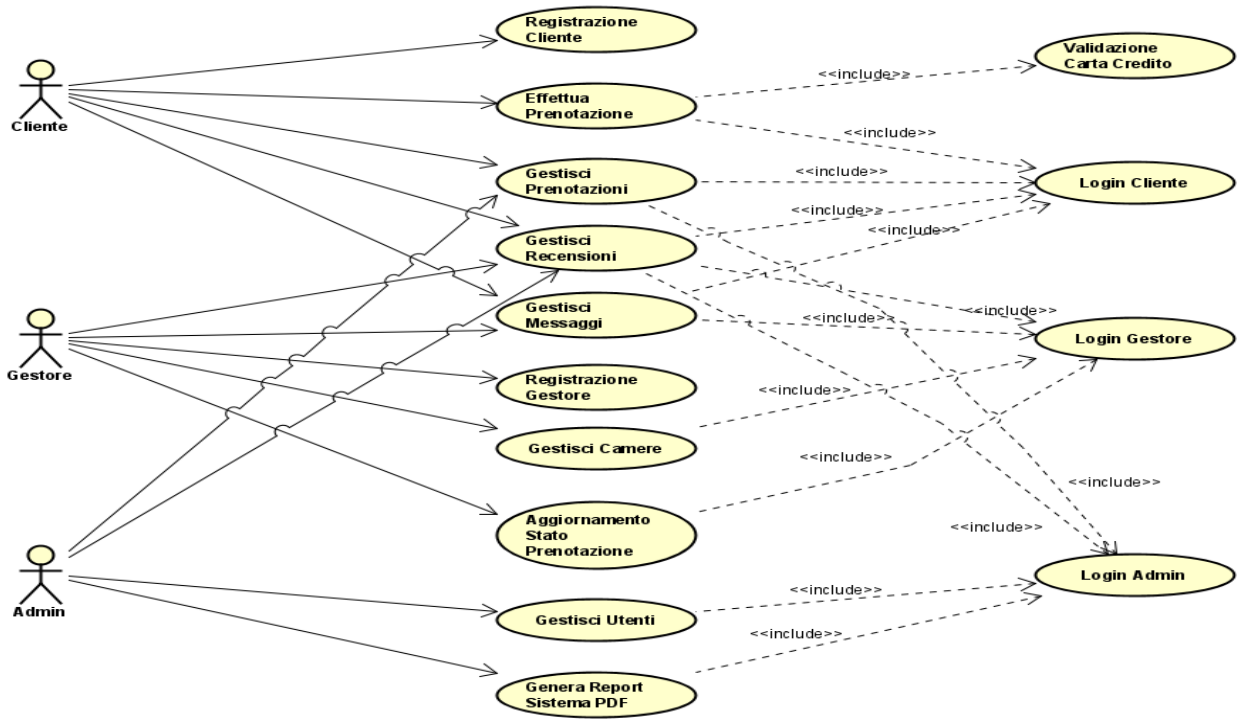
A partire dai primi anni ’90 il notevole incremento dei traffici aerei e lo sviluppo di internet hanno portato ad una crescita delle attività turistiche senza precedenti. Il numero sempre crescente di persone che si spostano ha portato alla necessità di poter usufruire di soluzioni abitative temporanee come ad esempio hotel, case vacanze e B&B, si è pensato, allora, ad un sistema che potesse gestire in maniera semi-automatizzata questo tipo di abitazioni. Il sistema deve esporre le seguenti funzionalità:

- I nuovi clienti effettuano una registrazione in corrispondenza della quale avranno la possibilità di scegliere il proprio tipo di utenza.

    - Modalità ‘Gestore Struttura’: mette a disposizione la sua struttura inserendola sul sistema, si cura di inserire disponibilità, prezzi per la struttura appena creata ed eventuali sconti.
    - modalità ‘Cliente’: usufruisce del servizio, prenota la stanza nel luogo da lui scelto.

  Inoltre, all’interno del sistema sarà presente un ‘Admin’ che potrà gestire l’intero sistema (es. eliminare prenotazione), non ci sarà la possibilità di registrarsi come ‘Admin’. I clienti già registrati, invece, effettueranno l’accesso con le proprie credenziali.
- Le camere inserite dai vari gestori possono essere di vari tipi (es. singole, doppie, triple).
- Ad ogni tipo di camera corrisponde un prezzo in base al periodo.
- In base alle scelte del Cliente (città, periodo, date e tipo camera) il sistema si occupa di mostrare le varie strutture e camere disponibili con il relativo prezzo.
- Il prezzo finale della camera viene calcolato dal sistema sommando al prezzo per notte, inserito dal gestore, le tasse e la percentuale spettante al sistema stesso.
- Il cliente può decidere di eliminare la prenotazione.
- Il cliente può lasciare una recensione.
- All’interno della sezione ‘Notifiche’ il cliente potrà visualizzare messaggi relativi alla sua prenotazione (es. annullamento).

Modello dei casi d’uso



I casi d'uso che analizzerò in modo dettagliato saranno: 'Effettua Prenotazione' e 'Genera PDF Prenotazione'.

Nome del caso d'uso	**UC1: Effettua Prenotazione**
Portata	Applicazione *JourneyHouse*
Livello	Obiettivo utente
Attore primario	Cliente
Parti interessate e interessi	**Cliente**: desidera poter prenotare la camera da lui scelta per il periodo selezionato, in modo autonomo e semplice. **Gestore Struttura:** auspica che i clienti non abbiamo difficoltà nella prenotazione e possano usufruire in modo agevole del servizio di pagamento. Desidera ricevere in modo sicuro la somma di denaro e che la disponibilità delle camere venga aggiornata correttamente.
Precondizioni	Il cliente ha effettuato l'accesso.
Garanzia di successo	Il cliente sceglie la modalità di pagamento e conferma la prenotazione.
Scenario di principale di successo	1. Il cliente inserisce luogo e date del soggiorno desiderato. 2.Il sistema mostra le camere disponibili seguendo le preferenze del cliente. 3.Il cliente sceglie una fra le camere disponibili. 4. Il cliente seleziona la modalità di pagamento con carta. 5. Il sistema procede alla validazione della Carta di Credito [vedi caso d'uso Validazione Carta Credito] 6.Il pagamento va a buon fine. Il sistema aggiorna la disponibilità delle camere di quella determinata struttura per il periodo selezionato e mostra l'id della prenotazione insieme ad un messaggio di successo. 7. Il sistema genera un file PDF riguardante le informazioni sulla sua prenotazione.
Estensioni	4a. Il cliente seleziona la modalità di pagamento in contanti. 1. Il sistema aggiunge allora anche un costo di interesse. 2. Si procede al passo 6. 6a. Il pagamento non va a buon fine. 1. Il sistema mostra un messaggio di errore e chiede all'utente di ricominciare con l'azione di effettuazione della prenotazione.
Requisiti speciali	Non specificato.
Elenco delle variabili tecnologiche e dei dati	Il sistema accetta qualunque tipo di carta di pagamento.
Frequenze di ripetizioni	Probabilmente alta, dipende dal numero di clienti che utilizzano *JourneyHouse*.
Varie	Non specificato.

Nome del caso d'uso	**UC2: Validazione Carta Credito**
Portata	Applicazione *JourneyHouse*
Livello	Obiettivo utente
Attore primario	Cliente
Parti interessate e interessi	**Cliente**: desidera poter prenotare il suo soggiorno mediante la sua carta di credito. **Gestore Struttura:** desidera poter essere certo che la carta di credito inserita dall'utente sia valida.
Precondizioni	Il cliente ha effettuato l'accesso al sistema, scelto la struttura di suo gradimento e confermato di voler pagare con Carta di Credito.
Garanzia di successo	Il sistema verifica correttamente se la carta di credito è valida o meno.

Scenario di principale di successo	1.Il cliente sceglie la modalità "Effettua Prenotazione". 2.Il cliente sceglie la modalità di pagamento con carta di Credito. 3.Il sistema verifica che la carta di credito inserita sia valida. 4.Viene fornito il benestare e la prenotazione può concludersi.
Estensioni	4a. La carta di credito inserita non è valida. 1. Il sistema mostra un messaggio d'errore. 2. Viene chiesto al cliente di reinserire il numero della carta di credito.
Requisiti speciali	Non specificato.
Elenco delle variabili tecnologiche e dei dati	Non specificato.
Frequenze di ripetizioni	Probabilmente alta, dipende dal numero di clienti che utilizzano *JourneyHouse*.
Varie	Non specificato.

- **UC3: Registra Cliente**

1. Il cliente fornisce al sistema il proprio nome, cognome, data di nascita, e-mail ed una password.
2. Il sistema preleva i dati forniti dall'utente e restituisce un riepilogo.
3. Il cliente conferma il riepilogo.
4. Il sistema termina la registrazione.

Scenari alternativi:

4a. L'operazione non va a buon fine.
1. Il sistema mostra un messaggio d'errore.
2. Si procede col passo 1.

- **UC4: Registra Gestore Struttura**

1. Il gestore struttura fornisce al sistema i propri dati personali ovvero nome, cognome, data di nascita, e-mail. Il gestore della struttura oltre ai suoi dati personali inserisce anche i dati relativi alla struttura stessa: nome, città, indirizzo, numero di telefono.
2. Il sistema preleva i dati forniti dal gestore e restituisce un riepilogo.
3. Il gestore conferma il riepilogo.
4. Il sistema termina la registrazione.

Scenari alternativi:

4a. L'operazione non va a buon fine.
1. Il sistema mostra un messaggio d'errore.
2. Si procede col passo 1.

- **UC5: Login Cliente**

1. Il cliente fornisce la propria e-mail e password.
2. Il sistema verifica la correttezza dei dati immessi e dà l'accesso al cliente.

Scenario alternativo:
2a. Il sistema non trova una corrispondenza coi dati immessi e chiede al cliente di reinserirli.

- **UC6: Login Gestore Struttura**

1. Il gestore fornisce la propria e-mail e password.
2. Il sistema verifica la correttezza dei dati immessi e dà l'accesso al cliente.

Scenario alternativo:
2a. Il sistema non trova una corrispondenza coi dati immessi e chiede al gestore di reinserirli.

- **UC7: Login Admin**

1. L'admin fornisce la propria e-mail e password.
2. Il sistema verifica la correttezza dei dati immessi e dà l'accesso all'admin.

Scenario alternativo:
2a. Il sistema non trova una corrispondenza coi dati immessi e richiede all'admin di reinserirli.

- **UC8: Gestisci Camere, CRUD**

1. Il gestore della struttura sceglie la modalità inserisci camera.
2. Il gestore inserisce tutti i dati necessari per l'inserimento della camera.
3. Il gestore conferma il riepilogo.
4. Il sistema crea la camera e invia un messaggio di riuscita dell'operazione.

Scenari alternativi:

1a. Il gestore sceglie di modificare una camera già inserita.
   1. Il sistema richiede il numero della camera.
   2. Il gestore inserisce il campo che intende modificare.
   3. Il sistema mostra un messaggio di successo.

1b. Il gestore o l'admin intende eliminare una camera.
   1. Il sistema richiede il numero della camera.
   2. Il sistema mostra un messaggio di successo.

2a. L'operazione non va a buon fine.
   1. Il sistema mostra un messaggio d'errore.
   2. Si procede col passo 1.

- **UC9: Gestisci Prenotazioni, CRUD**

1. Il Cliente o il gestore intendono visualizzare le prenotazioni.
2. Il sistema mostra le prenotazioni relative all'utente o al gestore che hanno effettuato la richiesta.

Scenari alternativi:

1a. Il cliente intende eliminare una prenotazione.
   1. Il sistema richiede l'id della prenotazione.
   2. Il sistema mostra un messaggio di successo.

2a. L'operazione non va a buon fine.
   1. Il sistema mostra un messaggio d'errore.
   2. Si procede col passo 1.

- **UC10: Gestisci Recensioni, CRUD**

1. Il cliente scrive una recensione assegnando un voto da 1 a 5.
2. Il sistema fornisce un riepilogo della recensione.
3. Il cliente conferma il riepilogo.
4. Il sistema crea la recensione, assegna alla stessa un id e mostra al cliente un messaggio di successo con il relativo id della recensione.

Scenari alternativi:

1a. Il cliente o l'admin intendono eliminare una recensione.
   1. Il sistema richiede l'id della recensione.
   2. Il sistema elimina la recensione.
   3. Il sistema mostra un messaggio di successo o errore.

1b. Il cliente intende modificare la recensione.
   1. Il sistema richiede l'id della recensione.
   2. Il cliente modifica la recensione.
   3. Il sistema mostra un messaggio di successo o errore.

1c. Il cliente, il gestore o l'admin intende leggere le recensioni.
   1. Il sistema mostra l'elenco di recensioni.

- **UC11: Gestisci Utenti, CRUD**

1. L'admin intende modificare l'e-mail di un utente.
2. Il sistema richiede l'id dell'utente a cui si intende apportare la modifica.
3. Il sistema richiede la nuova e-mail dell'utente.
4. Il sistema mostra un messaggio di successo.

Scenari alternativi:

1a. L'admin intende modificare la password di un utente.
   1. Il sistema richiede l'id dell'utente a cui si intende apportare la modifica.
   2. Il sistema richiede la nuova password dell'utente.
   3. Il sistema mostra un messaggio di successo.

1b. L'admin intende visualizzare tutti gli utenti registrati.
   1. Il sistema mostra l'elenco degli utenti registrati.

4a. L'operazione non va a buon fine.
   1. Il sistema mostra un messaggio d'errore.
   2. Si procede col passo 1.

Gli scenari di modifica e-mail e modifica password portano alla pubblicazione della notifica da parte del sistema sulla pagina notifiche dell'utente interessato.

- **UC12: Gestisci Messaggi, CRUD**

1. Si decide di inviare un messaggio ad un determinato utente.
2. Viene scelto il messaggio e l'utente (cliente o gestore) a cui inviarlo.

Scenari alternativi:

1b. Il cliente o il gestore decidono di visualizzare i messaggi ricevuti.
   1. Il sistema mostra l'elenco dei messaggi ricevuti.

- **UC13: Aggiornamento Stato Prenotazione**

1. Il gestore modifica lo stato della prenotazione una volta che il cliente è arrivato passando da “Check-in NON EFFETTUATO” a “Check-in EFFETTUATO”.

Scenari alternativi:

1a. Il gestore modifica lo stato della prenotazione una volta che il cliente è partito passando da “Check-in EFFETTUATO” a “Check-out EFFETTUATO”.

- **UC14: Genera Report Sistema PDF**

1. L’admin intende generare un file di report del sistema in formato PDF.
2. Viene mostrato un messaggio di successo e di relativa avvenuta creazione.

Scenari alternativi:

2a. L’operazione non va a buon fine.
1. Il sistema mostra un messaggio d’errore.
2. Si procede col passo 1.

Documento di visione

- **Introduzione**

L’obiettivo del progetto è la realizzazione di un sistema che permetta ai gestori delle varie strutture ricettive di metterle a disposizione dei clienti sparsi per il mondo e che permetta ai clienti un semplice, intuitivo e sicuro metodo di prenotazione. Il sistema si occuperà, in base alle prenotazioni, di gestire le disponibilità delle camere delle strutture.

- **Scopo**

Lo scopo di questo documento è quello di mostrare tutte le informazioni relative ad interessi e parti interessate del sistema, inoltre si occuperà anche di analizzare i vantaggi nell’utilizzo del sistema da parte del cliente.

- **Portata**

La portata di questo documento di visione è il software *JourneyHouse*.

- **Opportunità di business**

Il sistema verrà utilizzato per facilitare la prenotazione di una camera per la propria vacanza comodamente da casa, si potrà usufruire anche del parere degli altri viaggiatori mediante l’utilizzo delle recensioni. Inoltre, il sistema permetterà al cliente di valutare i prezzi delle camere delle varie strutture in modo da scegliere in maniera appropriata. Esistono già sul mercato sistemi simili, dai quali si distingue per la possibilità di aver il medesimo servizio a costi più bassi.

- **Formulazione del problema**

Descrizione del problema	La gestione di un’agenzia di viaggi fisica richiederebbe costi aggiuntivi dovuti al pagamento del personale di vendita e affitto dei locali. Recarsi in un’agenzia di viaggi comporta un dispendio di tempo maggiore ed inoltre la clientela risulta limitata ai luoghi in cui le agenzie stesse sono presenti. Oltre a ciò, risulta impossibile mostrare ai clienti un elevato numero di strutture ed i relativi prezzi. Risulta allora complicato ai gestori delle strutture far conoscere le proprie strutture ricettive.

Attori coinvolti	Gestori delle strutture, clienti.
Impatto	Le prenotazioni risultano essere limitate, dato il numero ristretto di clienti. Costi di affitto dei locali di vendita. Inoltre, il processo di prenotazione risulta essere rallentato.
Benefici di una soluzione di successo	Riduzione della complessità e dei tempi previsti per le vendite, che comporta un aumento dei guadagni, dovuto anche ad un aumento della clientela. Infatti, qualunque cliente potrà comodamente prenotare attraverso il sistema senza necessità di recarsi in agenzia.

- **Formulazione della posizione del prodotto**

Destinatari	Il prodotto è rivolto ad un'attività di gestione di strutture ricettive.
Obiettivi	I gestori delle varie strutture hanno la necessità di ampliare la propria clientela e di poter gestire le prenotazioni attraverso un sistema centralizzato semplice e facilmente accessibile.
Tipologia	Piattaforma di prenotazione.
Funzione	*JourneyHouse* fornisce un supporto sia lato gestore struttura che lato cliente per gestione e prenotazione delle camere.
Soluzioni alternative attuali	Le odierne agenzie di viaggio online hanno delle commissioni molto alte.
Caratteristiche prodotto	Permette il servizio di prenotazione contemporaneo a quello di gestione della struttura.

Parti interessate e descrizioni utente

- **Riepilogo delle parti interessate**

Nome	**Descrizione**	**Responsabilità**
Sviluppatore del software	Responsabile dello sviluppo della piattaforma.	Colui che si occupa della progettazione del software e dell'implementazione delle funzionalità richieste.
Committente	Società di gestione viaggi e soluzioni abitative temporanee.	Fornire i requisiti corretti necessari per lo sviluppo del software.

- **Riepilogo dell'utente**

Nome	**Descrizione**	**Responsabilità**	**Parte interessata**
Gestore Struttura	Gestisce le varie strutture.	Inserire, modificare o eliminare le	Committente

		camere ed i relativi prezzi.	
Cliente	Prenota una stanza.	Portare a compimento la prenotazione mediante il pagamento.	Committente
Admin	Funzionalità privilegiate.	Modificare e/o eliminare alcune voci.	Committente

- **Ambiente dell'utente**

  Il sistema può essere utilizzato dai Gestori Struttura, dai clienti e dall'Admin.

- **Profili delle parti interessate e degli utenti**

1) Committente

Rappresentante	*JourneyTravel*
Descrizione	Società di gestione viaggi.
Competenze	Alcuni membri del CDA hanno competenze informatiche che gli permettono di poter seguire lo svolgimento dei lavori ed infine utilizzare la piattaforma senza problemi.
Responsabilità	Ha la responsabilità di fornire i requisiti necessari per lo sviluppo del software, nelle fasi successive al rilascio, dovrà notificare l'eventuale presenza di problematiche.
Criteri di successo	Aumento delle vendite data la semplificazione del processo.
Coinvolgimento	I membri del CDA con competenze informatiche verranno coinvolti durante lo sviluppo e nella fase successiva al rilascio per evidenziare eventuali problematiche ed eventuali modifiche necessarie.
Elaborati aggiuntivi	Non previsti.
Commenti/problemi	Non previsti.

2) Sviluppatore

Rappresentante	Calogero Lentini

Descrizione	Sviluppatore e progettista del software.
Competenze	Buone competenze di sviluppo software.
Responsabilità	Ha la responsabilità di sviluppare il software al meglio, implementando tutte le funzionalità richieste dal negoziante attraverso la stesura dei requisiti.
Criteri di successo	Applicazione funzionante e soddisfacente tutte le richieste.
Coinvolgimento	Coinvolto durante tutte le fasi di progettazione, implementazione e rilascio.
Elaborati aggiuntivi	Non previsti.
Commenti/problemi	Non previsti.

3) Admin

Rappresentante	Mario Rossi
Descrizione	Membro del CDA.
Competenze	Buone competenze informatiche.
Responsabilità	Ha la possibilità di effettuare modifiche ed eliminazioni a prenotazioni, recensioni e credenziali qualora necessario.
Criteri di successo	Corretto funzionamento della piattaforma.
Coinvolgimento	Coinvolto durante tutte le fasi di progettazione, implementazione e rilascio.
Elaborati aggiuntivi	Non previsti.
Commenti/problemi	Non previsti.

- **Alternative e concorrenza**
  1. Booking www.booking.com
  2. Airbnb www.airbnb.it

Descrizione generale del prodotto

- **Punto di vista del prodotto**

Il software verrà implementato su un server del committente che esporrà il servizio ai vari utenti.

- **Riepilogo vantaggi**

Vantaggi per le parti interessate	**Caratteristica prodotto**
Aumento delle prenotazioni.	Prenotazioni multiple e rapide.
Possibilità di effettuare un rapido confronto fra i prezzi delle camere.	L'utilizzo del sistema consente l'inserimento di una vasta gamma di strutture.

- **Ipotesi e dipendenze**

1. Accesso alla rete da parte dei clienti per poter effettuare le prenotazioni.
2. Il cliente deve possedere una e-mail per poter effettuare la registrazione.

- **Licenze ed installazione**

JourneyHouse sarà un software Opensource.

Caratteristiche del sistema

- **Caratteristiche del sistema**

    1. Applicazione multipiattaforma

- **Acquisto e prenotazione**

    1. Prenotazioni della camera.
    2. Possibilità di eliminare la prenotazione.

Precedenza e priorità

Priorità	**Caratteristica**
Alta	Ricerca camera ed effettua prenotazione.
Media	Gestione camere e prenotazioni.
Bassa	Modalità admin, Login e Registrazioni.

Vincoli e requisiti del prodotto

- **Requisiti di sistema**

    1. Accesso alla rete internet.
    2. Disponibilità di Java Virtual Machine.

Regole di business

Di seguito sono elencate le regole di dominio:

ID	Regola	Modificabilità	Sorgente
R1	È possibile annullare una prenotazione entro tre (3) giorni prima dall'inizio del soggiorno.	Nessuna.	Politica interna della società di gestione viaggi.
R2	Nel caso in cui si scelga il pagamento in struttura verrà aggiunto un interesse di 10€ sul totale.	Nessuna.	Politica interna della società di gestione viaggi.
R3	Per tutti i clienti verrà aggiunta una percentuale del 2% di tasse da sommarsi al prezzo a notte inserito dal gestore della struttura.	Nessuna.	Legge italiana.
R4	Per tutti i clienti verrà aggiunta una percentuale del 3% spettante alla società di gestione viaggi.	Nessuna.	Politica interna della società di gestione viaggi.
R5	Il gestore potrà decidere di applicare uno sconto percentuale in un determinato periodo.	Alta, il gestore potrà decidere se, quando e in che percentuale applicare lo sconto.	Politica interna della società di gestione viaggi.

Specifiche supplementari

- **Usabilità**

1. L'interfaccia utente deve essere semplice ed intuitiva.
2. La ricerca e la visualizzazione delle varie camere deve essere anch'essa molto semplice ed intuitiva.
3. L'utente deve essere guidato tra le varie fasi dell'utilizzo del sistema.

Glossario

1. **Gestore struttura:** utilizzatore generico del sistema (Admin, Gestore Struttura o Cliente).
2. **Gestore struttura:** proprietario di una struttura o delegato della stessa che ha il compito di occuparsi di tutte le prenotazioni e di cosa ne consegue.
3. **Cliente:** utente che usufruisce del software per effettuare una prenotazione e gestire intuitivamente la sua vacanza.
4. **Admin:** dipendente della società di gestione viaggi che può svolgere alcune operazioni preferenziali.
5. **Notifica:** messaggio recapitato nella bacheca 'Notifiche' dell'utente nel caso in cui sia avvenuta una qualche modifica alla sua prenotazione.

2. Analisi orientata agli oggetti

La realizzazione dell'applicazione è stata articolata su quattro iterazioni, è stata dunque affrontata un'analisi dei requisiti graduale in modo da limitare le problematiche causate da eventuali errori di progettazione e implementazione.

Per ciascuna iterazione si è proceduto nel seguente modo:

Iterazione 1:

-Implementazione dello scenario principale di successo del caso d'uso *UC1: Effettua Prenotazione*.

- Implementazione del caso d'uso *UC2: Validazione Carta Credito*.

-Implementazione del caso d'uso di Start-Up necessario per gestire le esigenze di inizializzazione per questa iterazione.

Iterazione 2:

-Implementazione dello scenario alternativo del caso d'uso *UC1: Effettua Prenotazione*.

- Implementazione del caso d'uso *UC8: Gestisci Camere*.

- Implementazione del caso d'uso *UC9: Gestisci Prenotazioni*.

Iterazione 3:

- Implementazione del caso d'uso *UC10: Gestisci Recensioni*.

- Implementazione del caso d'uso *UC9: Gestisci Messaggi*.

Iterazione 4:

- Implementazione del caso d'uso *UC11: Gestisci Utenti*.

- Implementazione del caso d'uso *UC13: Aggiornamento Stato Prenotazione*.

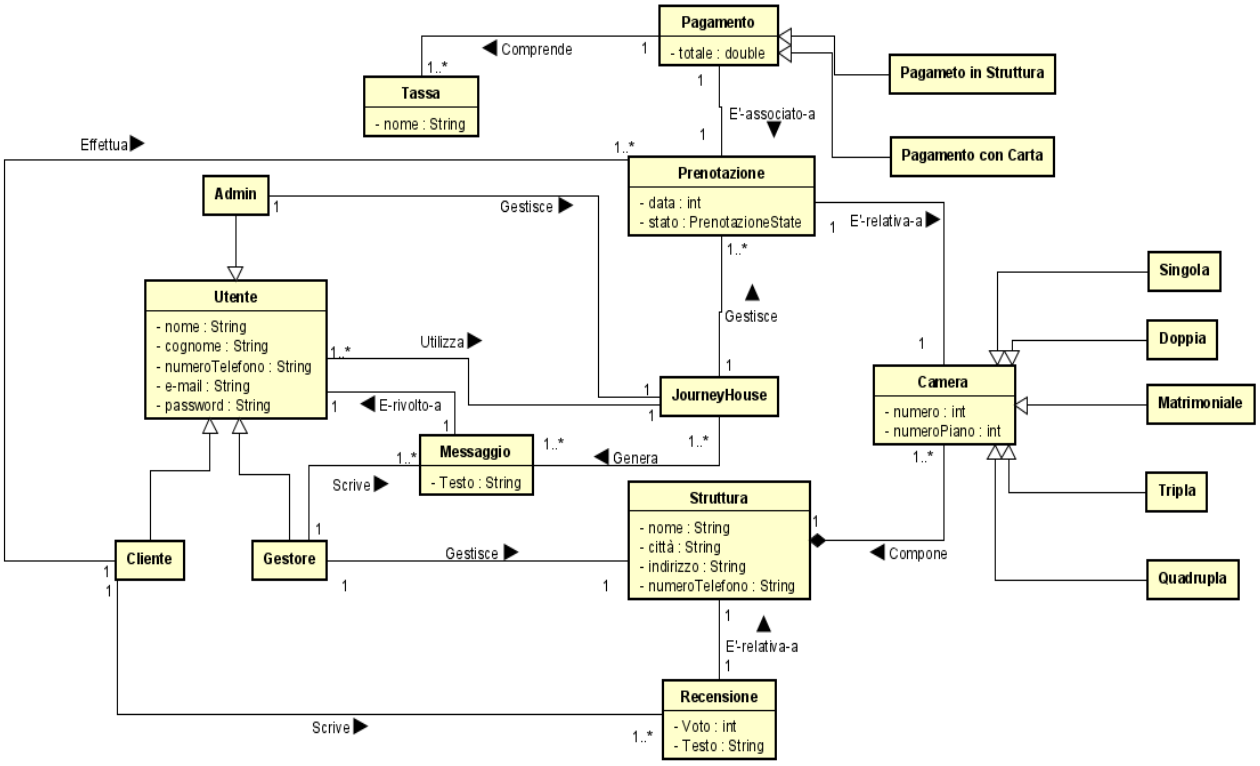
- Implementazione del caso d'uso *UC14: Genera Report Sistema PDF*.

- Implementazione dei vari casi d'uso di login e registrazione.

Il passo iniziale di ciascuna iterazione è stato quello di effettuare un'analisi dei requisiti Orientata agli oggetti. Per fornire tale descrizione sono stati utilizzati diversi strumenti: Modello di Dominio, SSD (Sequence System Diagram) e contratti delle operazioni.

2.1 Modello di Dominio

Il modello di dominio è un elaborato grafico in cui vengono identificati i concetti, gli attributi e le associazioni considerati significativi. Tenendo conto del contributo dato da ciascun'iterazione, il modello di dominio finale è il seguente:



Sono state individuate le seguenti classi concettuali:

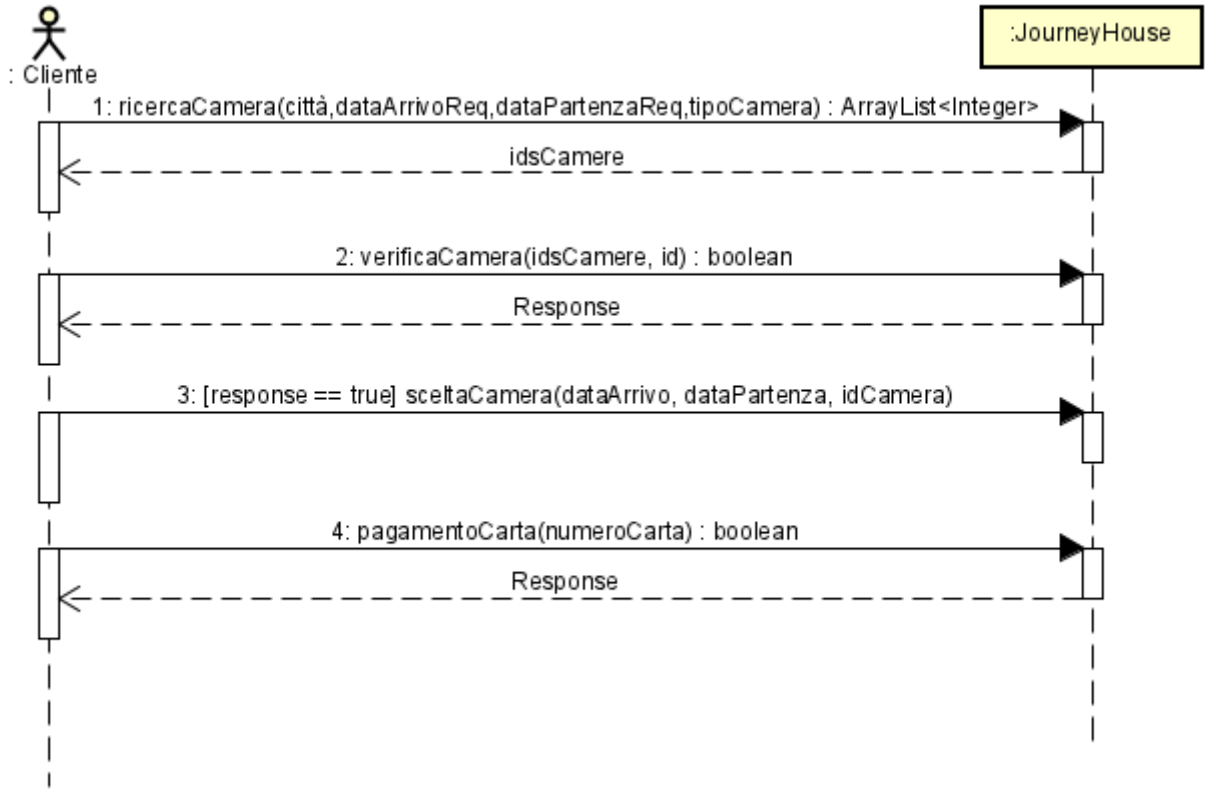
- **JourneyHouse**: sistema centrale.
- **Gestore**: gestore della struttura.
- **Cliente**: cliente del sistema che effettua la prenotazione.
- **Admin**: gestore del sistema.
- **Struttura**: è la struttura creata dal Gestore che ospiterà i vari clienti.
- **Camera:** rappresenta il prodotto che il Cliente va a prenotare.
- **Pagamento:** tipologia di pagamento che viene scelta dal cliente.
- **Prenotazione:** dati relativi alla prenotazione.
- **Tassa:** tipologia di tassa che viene aggiunta al totale da pagare.
- **Recensione:** Giudizio dato dal cliente rispettivamente ad una struttura.
- **Messaggio:** messaggio inviato dal gestore ad un cliente per inviare delle informazioni utili per il soggiorno o messaggio generato dal sistema per notificare un determinato avvenimento a clienti o gestori.

2.2 SSD e Contratti

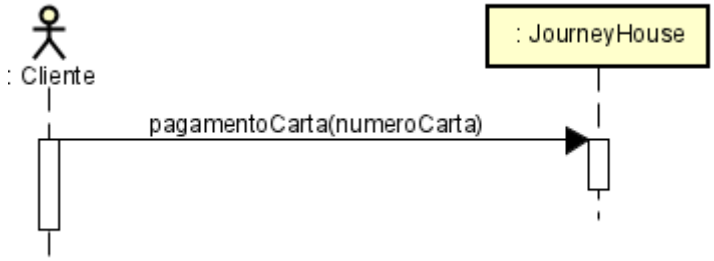
Il passo successivo è la creazione dei Diagrammi di Sequenza di Sistema (SSD) al fine di illustrare il corso degli eventi di input e di output per i vari casi d'uso esaminati in ciascuna iterazione. Inoltre, le principali operazioni di sistema individuate negli SSD verranno descritte attraverso i Contratti, quindi avremo:

Iterazione 1:

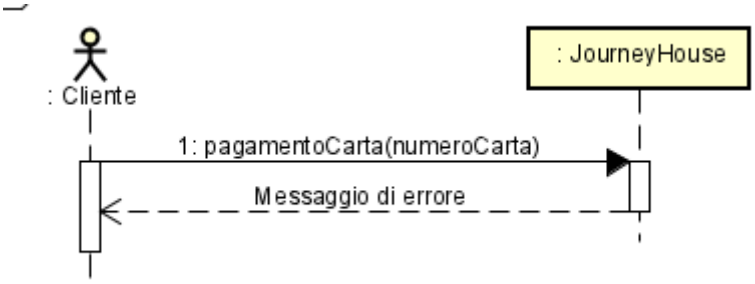
SSD UC1:



SSD UC2:



SSD UC2 alternativo:



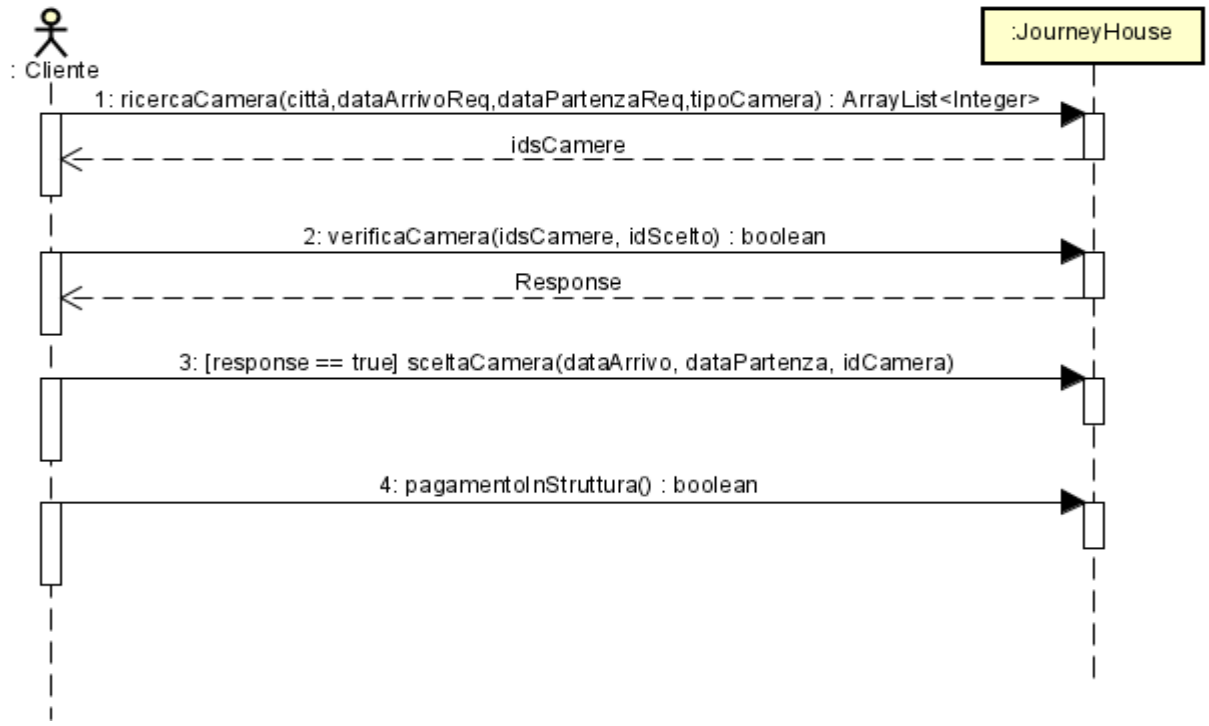
- **CO1: ricercaCamera (città, dataArrivo, dataPartenza, tipoStanza)**
  Operazione: ricercaCamera (città: String, dataArrivo: Date, dataPartenza: Date, tipoCamera: String).
  Riferimenti: UC1: *Effettua Prenotazione*.
  Precondizioni: È in corso la ricerca di una camera in base alle preferenze del cliente e la conseguente prenotazione. La date di arrivo e partenza devono essere date valide nel formato specificato. Il tipo di stanza deve essere una stringa valida corrispondente a uno dei tipi di camera disponibili.
  Post-Condizioni: - Sono state create tre ArrayList (idsStrutture, idsCamere, idsCamereNonDisponibili).

- **CO2: sceltaCamera (dataArrivo, dataPartenza, idCamera)**
  Operazione: sceltaCamera (dataArrivo: Date, dataPartenza: Date, idCamera: int).
  Riferimenti: UC1: *Effettua Prenotazione*.
  Precondizioni: È in corso la ricerca di una camera in base alle preferenze del cliente e la conseguente prenotazione.
  Post-Condizioni: - c*InCorso* viene aggiornato con la camera scelta dal cliente.
  - Viene avviata una Prenotazione *pInCorso.*
  - Gli attributi di *pInCorso* sono stati aggiornati.

- **CO3: pagamentoCarta(numeroCarta)**
  Operazione: pagamentoCarta (numeroCarta: int).
  Riferimenti: UC1: *Effettua Prenotazione*, UC2: *Validazione Carta Credito*.
  Precondizioni: È in corso una Prenotazione *pInCorso*, il cliente ha già selezionato la camera di suo gradimento.
  Post-Condizioni: - È stata creata una nuova istanza di Pagamento *pagInCorso*.
  - È stata creata una nuova istanza di Prenotazione *nuovaPrenotazione*.
  - Gli attributi di *nuovaPrenotazione* sono stati aggiornati con quelli di *pInCorso*.
  - *nuovaPrenotazione* è stata aggiunta all'archivio delle prenotazioni.
  -*nuovaPrenotazione* è stata aggiunta all'archivio delle prenotazioni della camera cInCorso.
  - È stato creato un file PDF riguardante le informazioni sulla prenotazione appena effettuata.

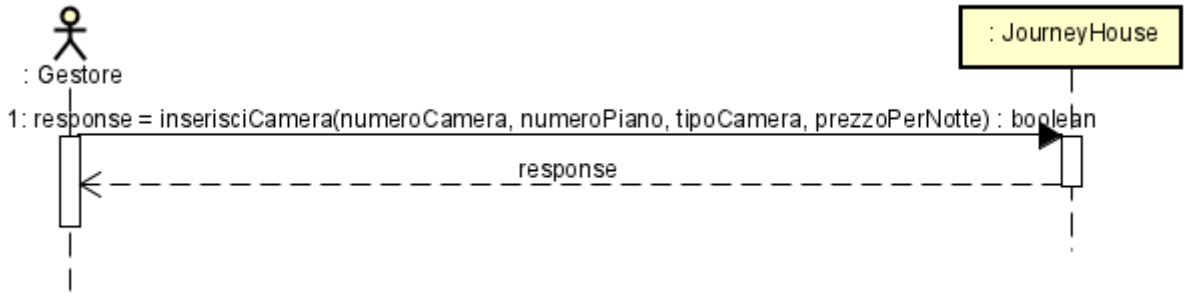
Iterazione 2:

SSD UC1 alternativo:

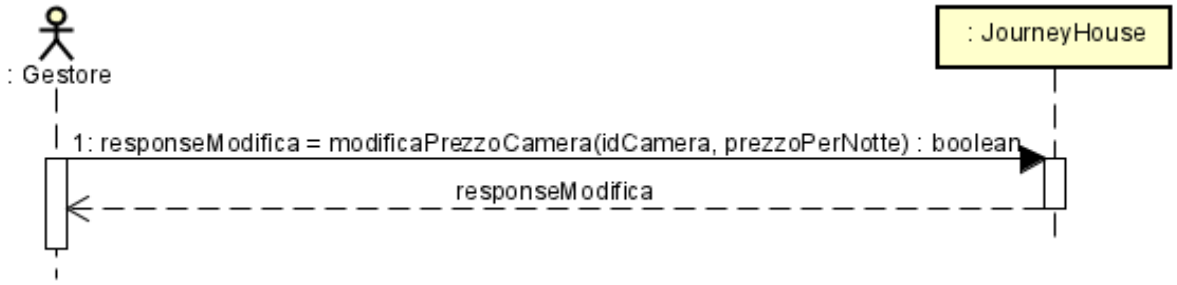


Come si nota l'unica differenza tra i due SSD è costituita dal fatto che nello scenario alternativo è previsto il pagamento in contanti in struttura e cioè l'operazione pagamentoInStruttura ().

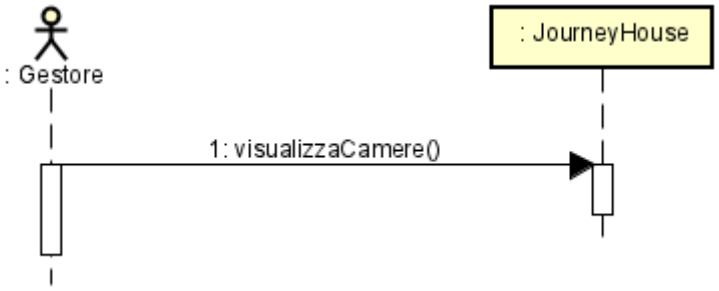
SSD UC8 (inserimento):



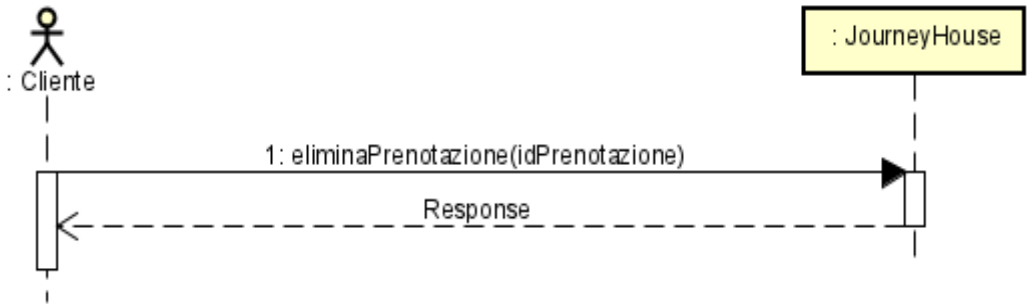
SSD UC8 (modifica):



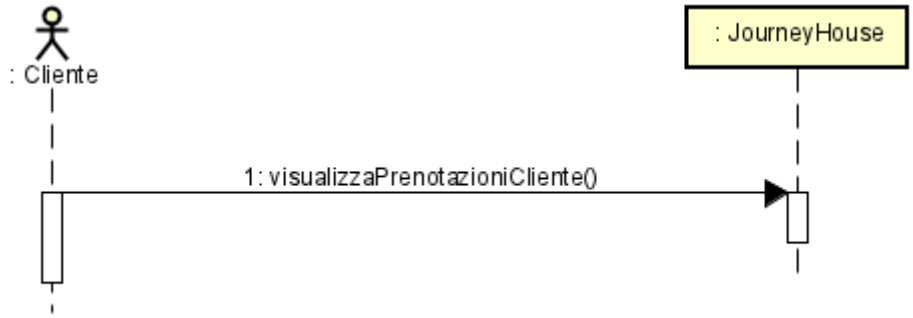
SSD UC8 (visualizza):



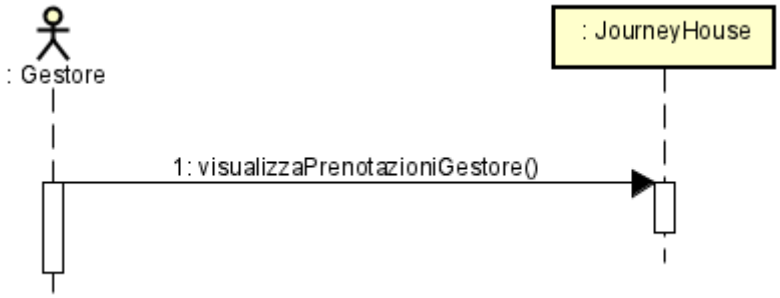
SSD UC9 (elimina):



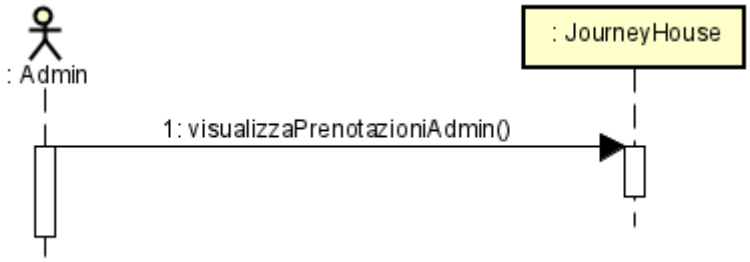
SSD UC9 (visualizzaPrenotazioniCliente):



SSD UC9 (visualizzaPrenotazioniGestore):



SSD UC9 (visualizzaPrenotazioniAdmin):



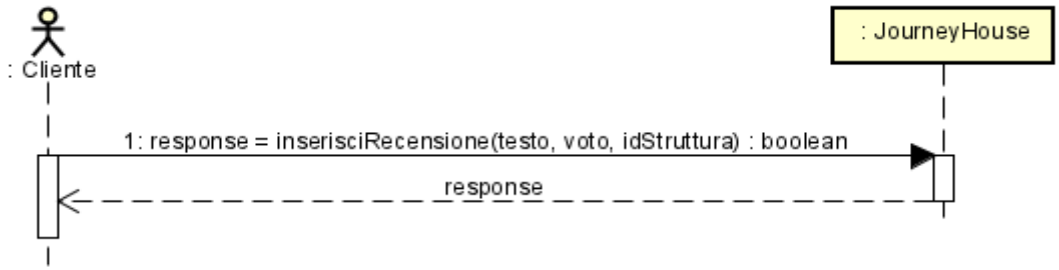
- **CO1: inserisciCamera (numeroCamera, numeroPiano, tipoCamera, prezzoPerNotte)**
  Operazione: inserisciCamera (numeroCamera: int, numeroPiano: int, tipoCamera: String, prezzoPerNotte: double).
  Riferimenti: UC8: *Gestisci camera*.
  Precondizioni: è in corso la creazione di una camera da parte del gestore.
  Post-Condizioni: - è stata creata una nuova istanza di Camera *cameraCorrente.* Il tipoCamera deve essere una stringa valida corrispondente a uno dei tipi di camera disponibili e il prezzoPerNotte deve essere un double valido.
  - Gli attributi di *cameraCorrente* vengono aggiornati con i parametri inseriti dall'utente
  - *cameraCorrente* è stata inserita all'interno delle camere della struttura corrispondente.
  - *cameraCorrente* è stata aggiunta all'archivio delle camere disponibili.

- **CO2: modificaPrezzoCamera (idCamera, prezzoPerNotte)**
  Operazione: inserisciCamera (idCamera: int, prezzoPerNotte: double).
  Riferimenti: UC8: *Gestisci camera*.
  Precondizioni: è in corso la modifica del prezzo da parte del gestore. Il prezzoPerNotte deve essere un double valido.
  Post-Condizioni: - è stata creata una nuova istanza di Camera *cameraCorrente*

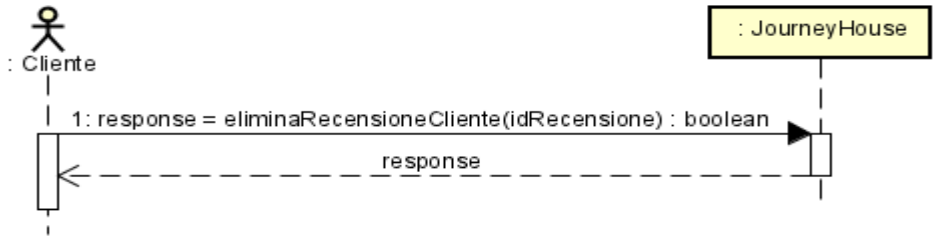
- L’attributo prezzoPerNotte della camera selezionata dall’utente è stato aggiornato.

Iterazione 3:

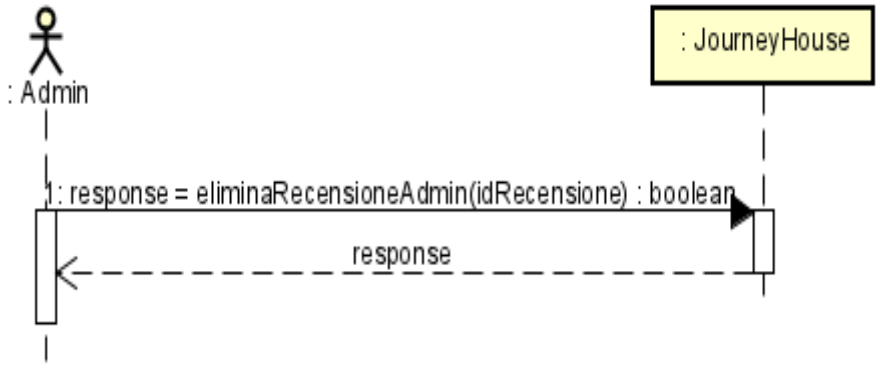
SSD UC10 (inserimento):



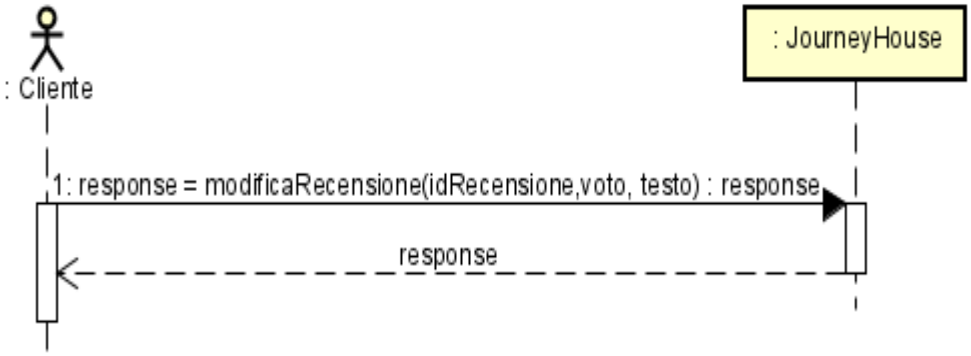
SSD UC10 (eliminaRecensioneCliente):



SSD UC10 (eliminaRecensioneAdmin):

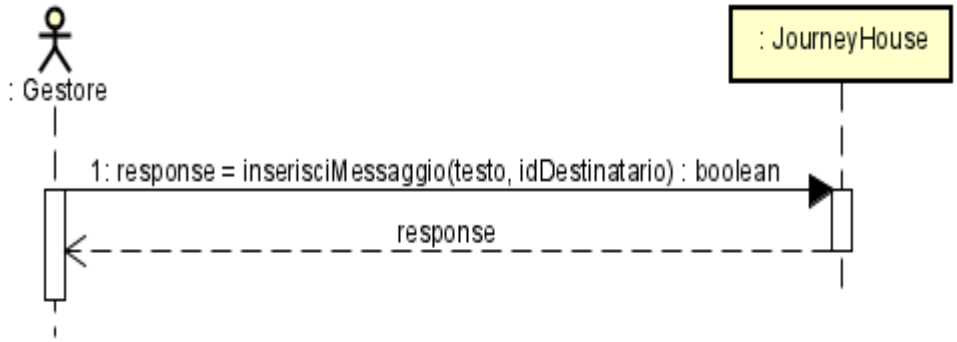


SSD UC10 (modifica):



SSD UC10 (visualizza):

SSD UC12 (inserimentoMessaggio):



SSD UC12 (visualizzaMessaggio):

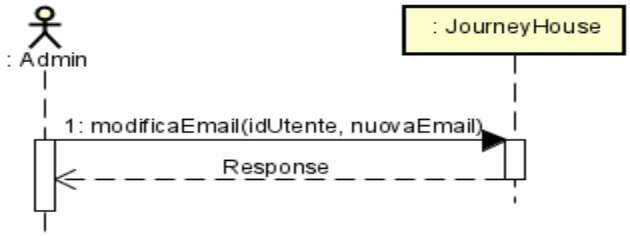
- **CO1: inserisciRecensione (testo, voto, idStruttura)**
  Operazione: inserisciRecensione (testo: String, voto: int, idStruttura: int).
  Riferimenti: UC10: *Gestisci recensione*.
  Precondizioni: è in corso la creazione di una recensione da parte del cliente.
  Post-Condizioni: - è stata creata una nuova istanza di Recensione *nuovaRecensione.*
  - Gli attributi di *nuovaRecensione* vengono aggiornati con i parametri inseriti dall'utente.
  - *nuovaRecensione* è stata inserita all'interno delle recensioni della struttura corrispondente.
  - *nuovaRecensione* è stata aggiunta all'archivio delle recensioni.

- **CO2: inserisciMessaggio (testo, idDestinatario)**
  Operazione: inserisciMessaggio (testo: String, idDestinatario: int).
  Riferimenti: UC12: *Gestisci messaggio*.
  Precondizioni: è in corso la creazione di un messaggio da parte del gestore.
  Post-Condizioni: - è stata creata una nuova istanza di Messaggio *nuovoMessaggio.*
  - Gli attributi di *nuovoMessaggio* vengono aggiornati con i parametri inseriti dall'utente.
  -*nuovoMessaggio* è stata inserito all'interno dei messaggi ricevuti dall'utente destinatario.

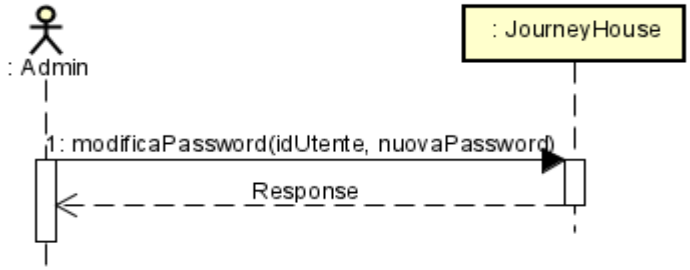
- *nuovoMessaggio* è stata aggiunto all'archivio dei messaggi.

Iterazione 4:

SSD UC11 (modificaEmail):

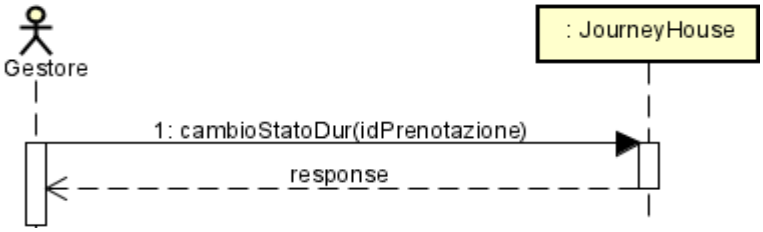


SSD UC11 (modificaPassword):

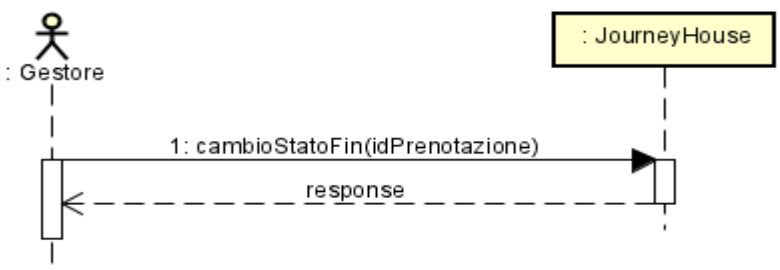


SSD UC11 (visualizzaUtenti):

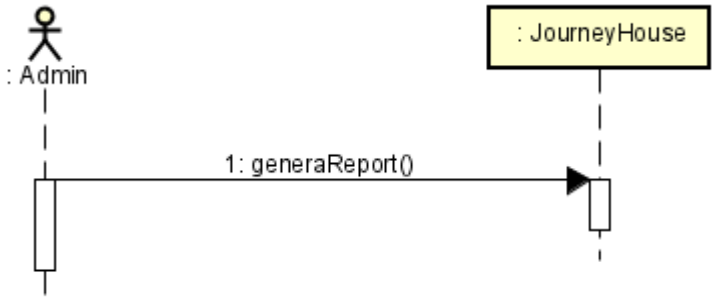
SSD UC13 (aggiornamentoStato1):



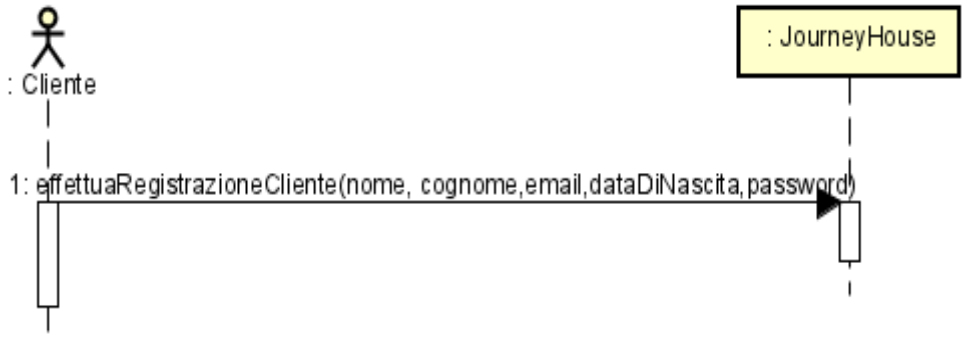
SSD UC13 (aggiornamentoStato2):



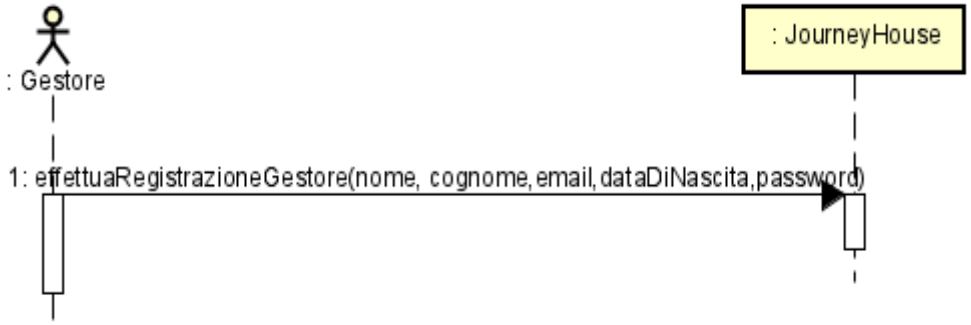
SSD UC14:



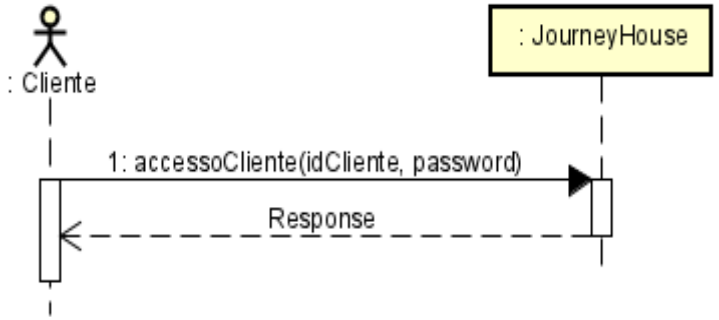
SSD UC3:



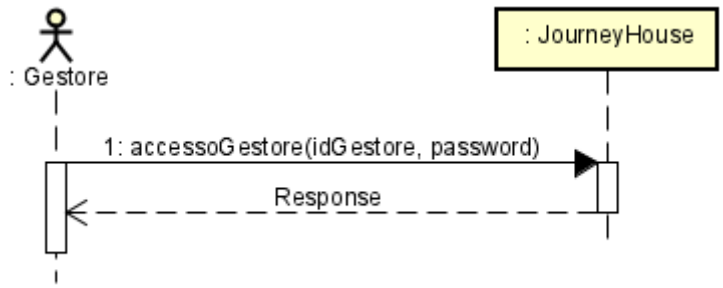
SSD UC4:



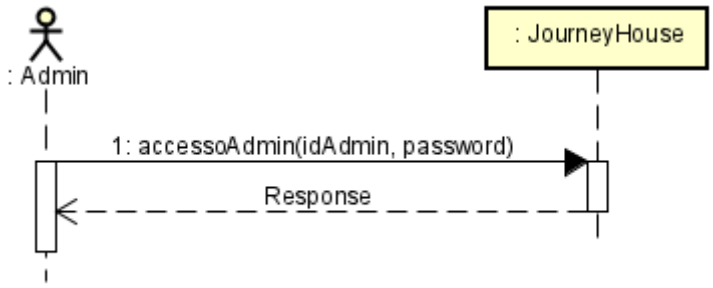
SSD UC5:



SSD UC6:

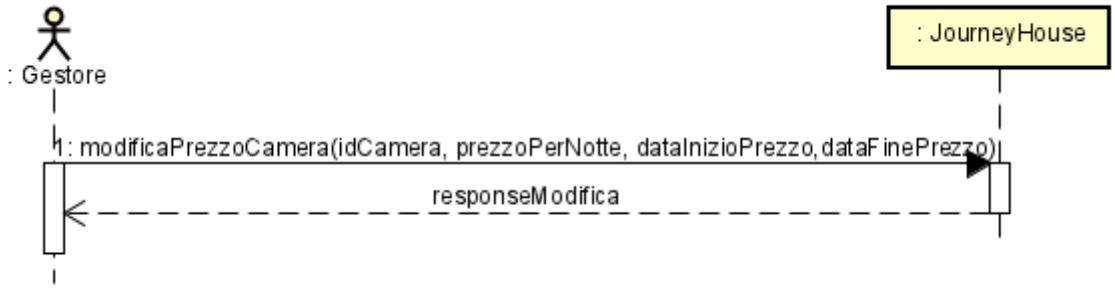


SSD UC7:



Durante lo sviluppo di JourneyHouse è stato rilevato un errore logico nel sistema di modifica dei prezzi delle camere. Inizialmente non era possibile specificare le date di inizio e di fine per la variazione del prezzo, il che limitava la flessibilità nella gestione delle tariffe. Per correggere questa problematica, è stata introdotta la possibilità di specificare le date di inizio e fine per ogni variazione di prezzo desiderata. Inoltre, nel caso in cui il prezzo per una data specifica non fosse stato inserito viene utilizzato automaticamente il prezzo di default come valore di riferimento. Queste correzioni hanno permesso di migliorare la gestione dei prezzi delle camere, offrendo maggiore flessibilità nella creazione di scontistiche e promozioni e garantendo un funzionamento corretto anche nel caso in cui i prezzi specifici per determinate date non siano stati inseriti dal gestore.

Si riporta dunque l'SSD del caso d'uso UC8 ed in seguito l'aggiornamento del SD.



- **CO1: cambiaStatoPrenotazioneDur (idPrenotazione)**
  Operazione: inserisciRecensione (idPrenotazione:int).
  Riferimenti: UC13: *Aggiornamento Stato Prenotazione*.
  Precondizioni: è stata creata una prenotazione, è in corso il cambiamento dello stato da PRECHECKIN a

DURANTESOGGIORNO.

Post-Condizioni: - è stata creata una nuova istanza di Messaggio *msg* con all'interno un messaggio per l'utente relativo alla prenotazione.

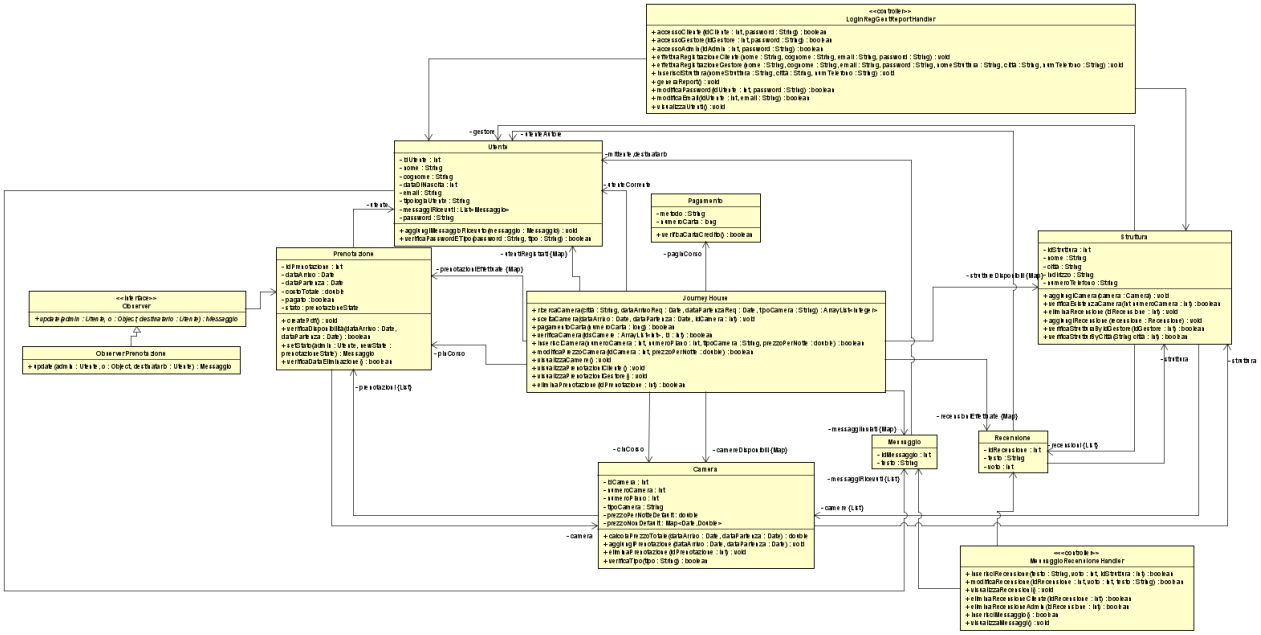
- Gli attributi di *nuovaRecensione* vengono aggiornati con i parametri inseriti dall'utente.
- *msg* è stata inserito all'interno dei messaggi ricevuti dall'utente destinatario.
- *msg* è stata aggiunto all'archivio dei messaggi.

3. Progettazione

3.1 Diagramma delle classi

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 (Diagramma delle Classi).

Il diagramma delle classi dunque sarà:



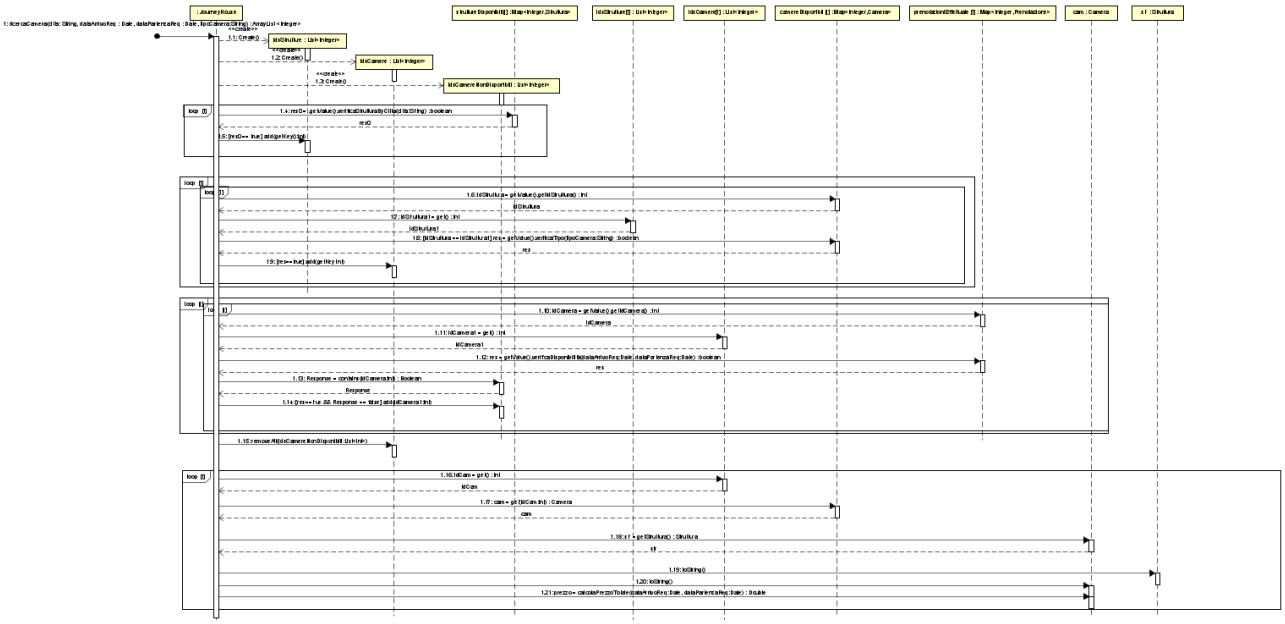
Per un maggiore dettaglio grafico si consulti l'allegato **A**.

3.2 Diagrammi di sequenza

Vengono ora presentati i diagrammi di sequenza realizzati nelle varie iterazioni.

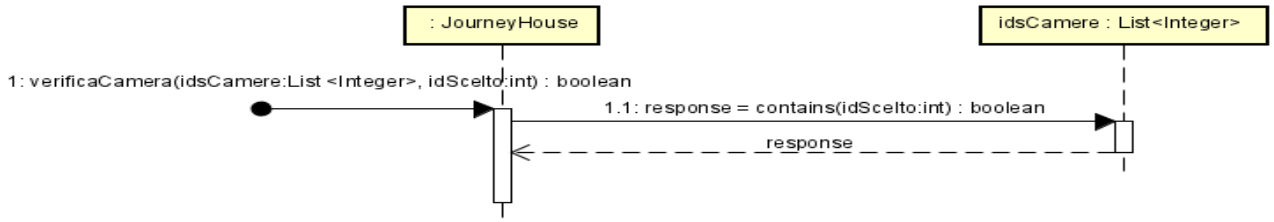
Iterazione 1:

UC1 – ricercaCamera

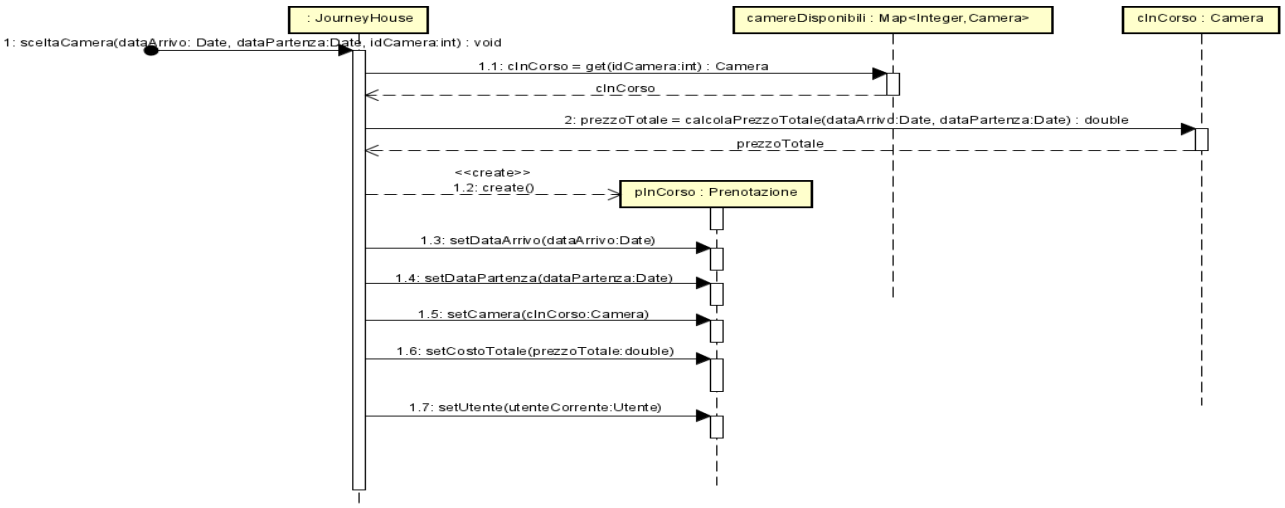


Per un maggiore dettaglio grafico si consulti l'allegato **A1**.

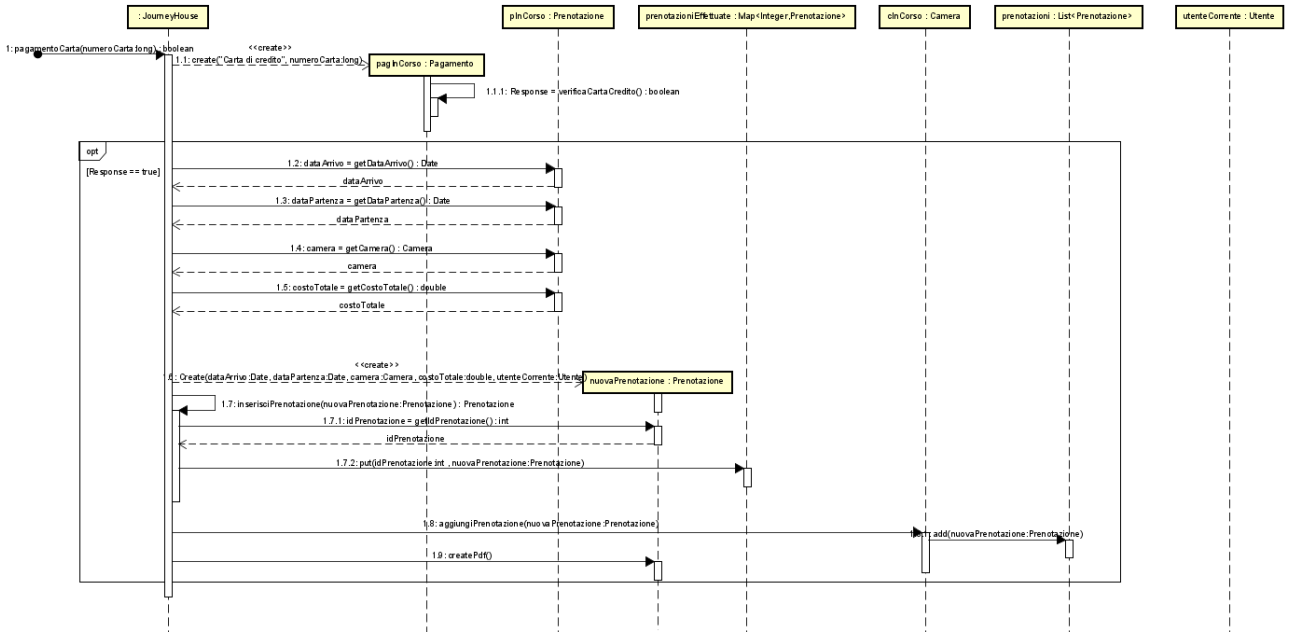
UC1 – verificaCamera



UC1 – sceltaCamera

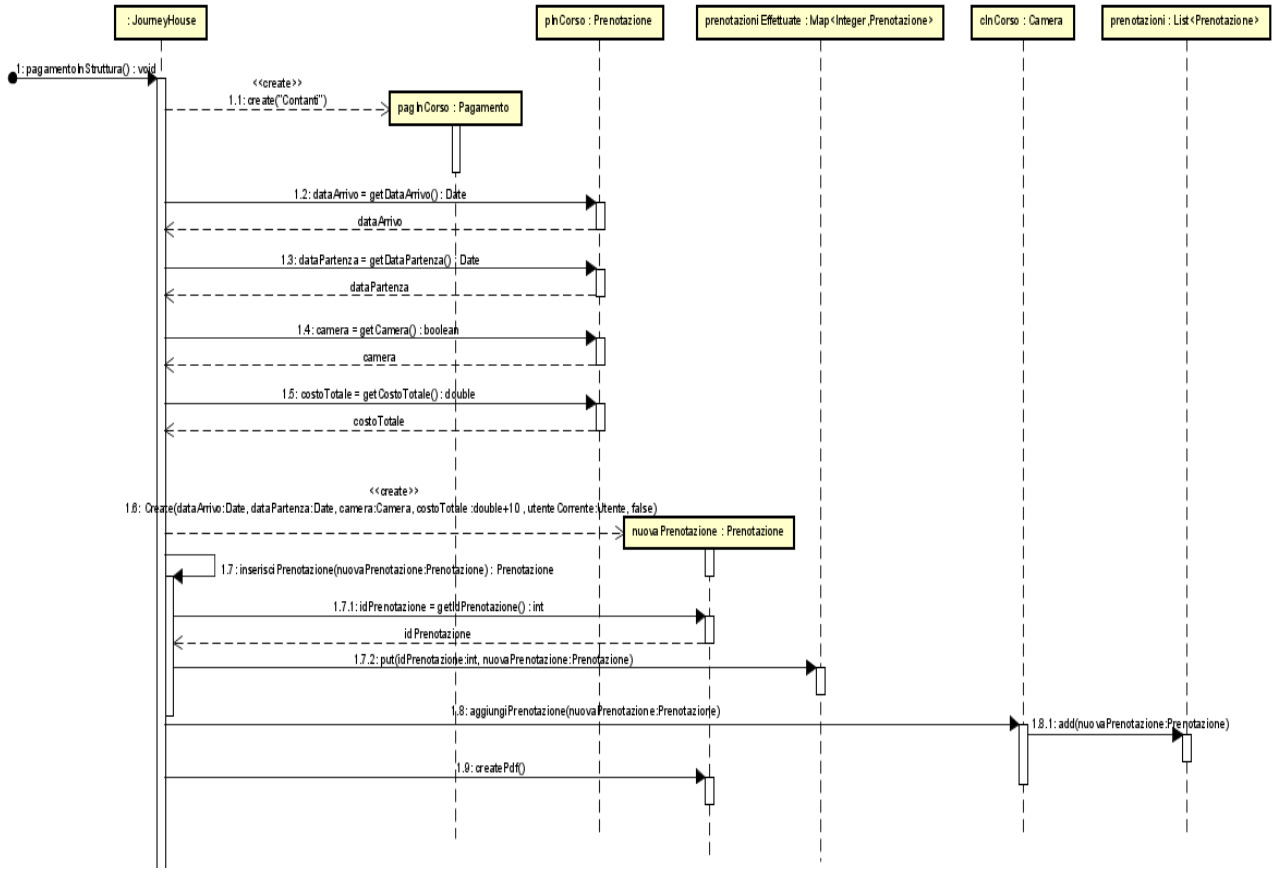


UC1 – pagamentoCarta

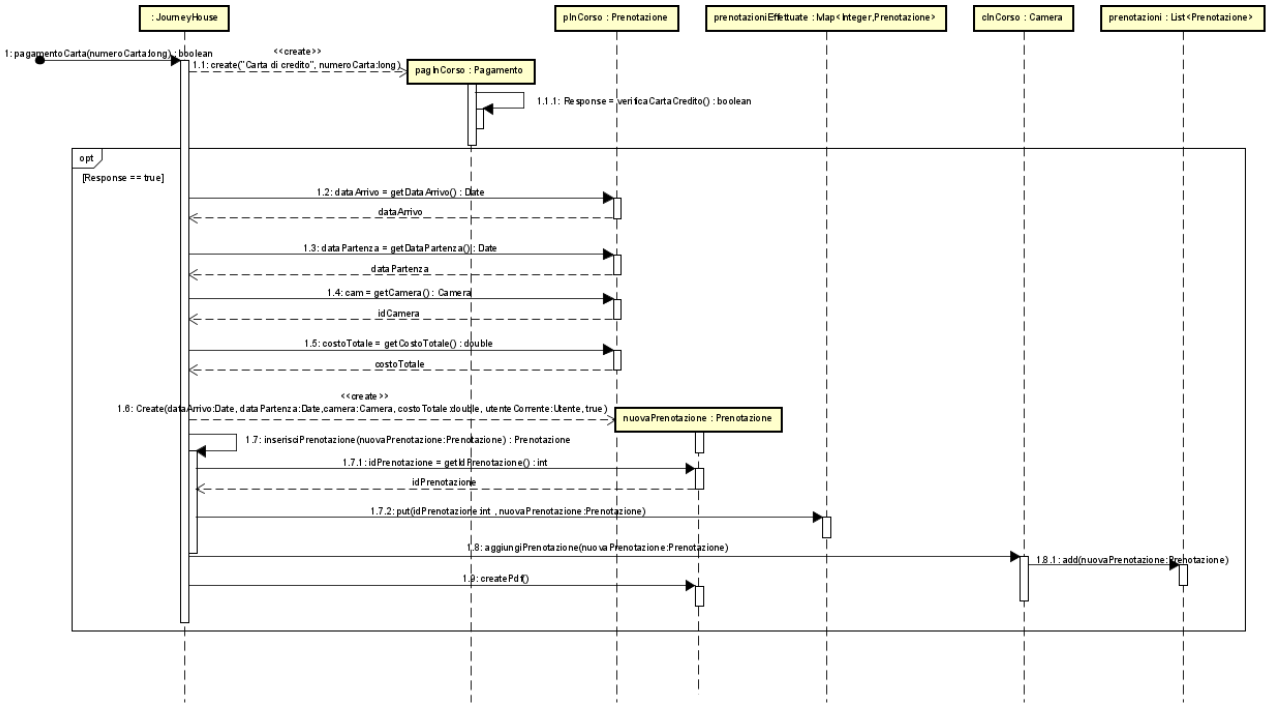


Iterazione 2:

UC1 – pagamentoInStruttura

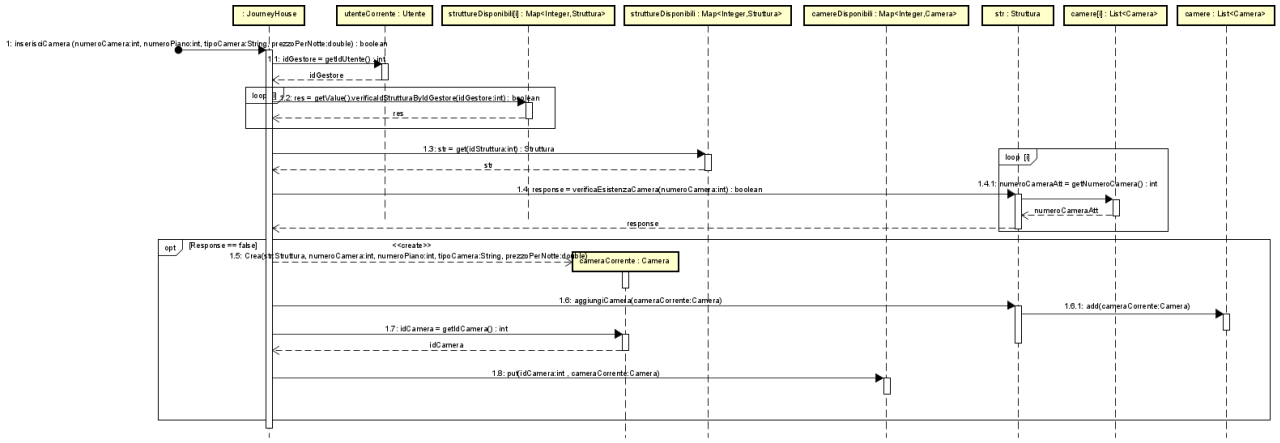


UC1 – pagamentoCarta (Aggiornamento)

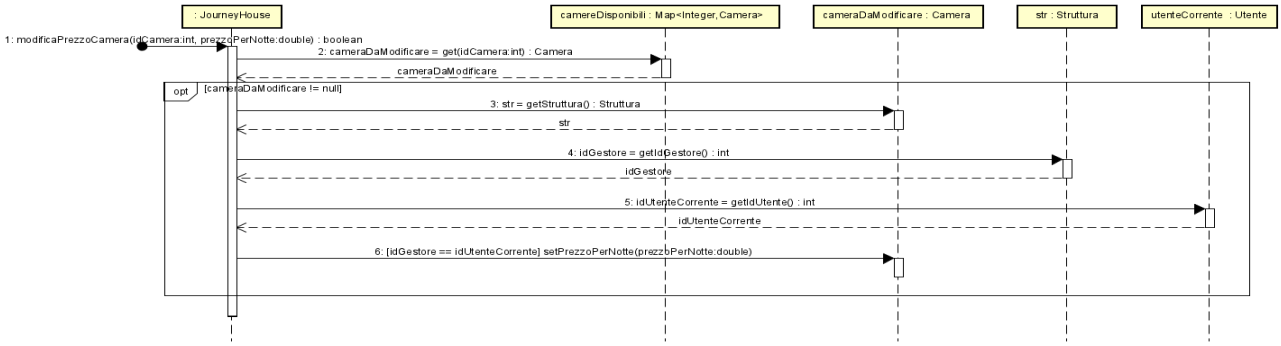


Dopo attente valutazioni si è ritenuto opportuno inserire all'interno della classe Prenotazione il campo booleano *pagato* che sta ad indicare se la prenotazione è stata già pagata o meno (pagata online con carta o al contrario pagamento in struttura). Si riporta dunque di seguito anche l'aggiornamento del diagramma di sequenza *pagamentoCarta*.

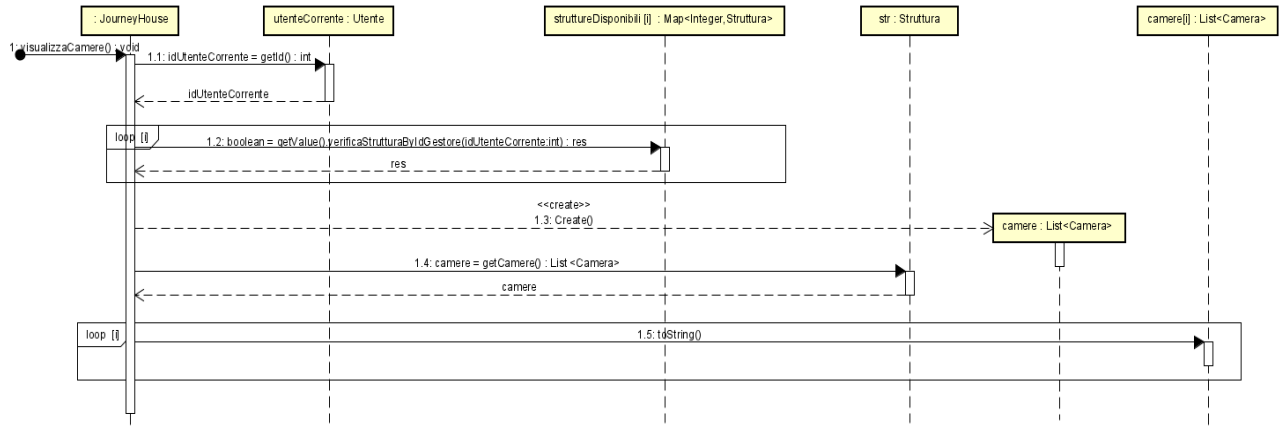
UC8 – InserisciCamera



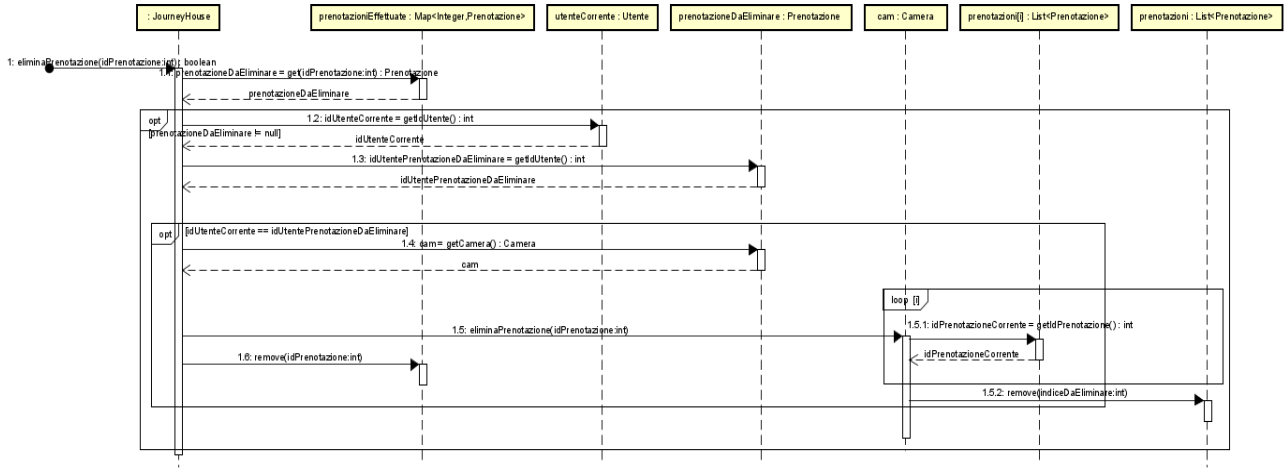
UC8 – modificaPrezzoCamera



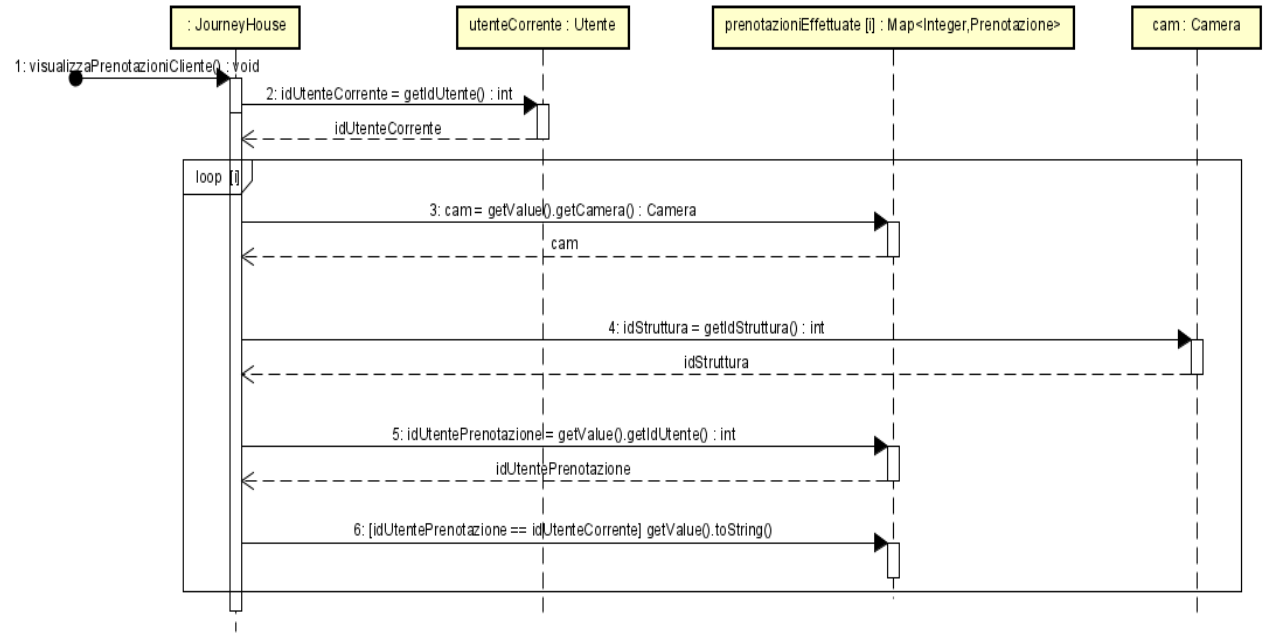
UC8 – visualizzaCamere



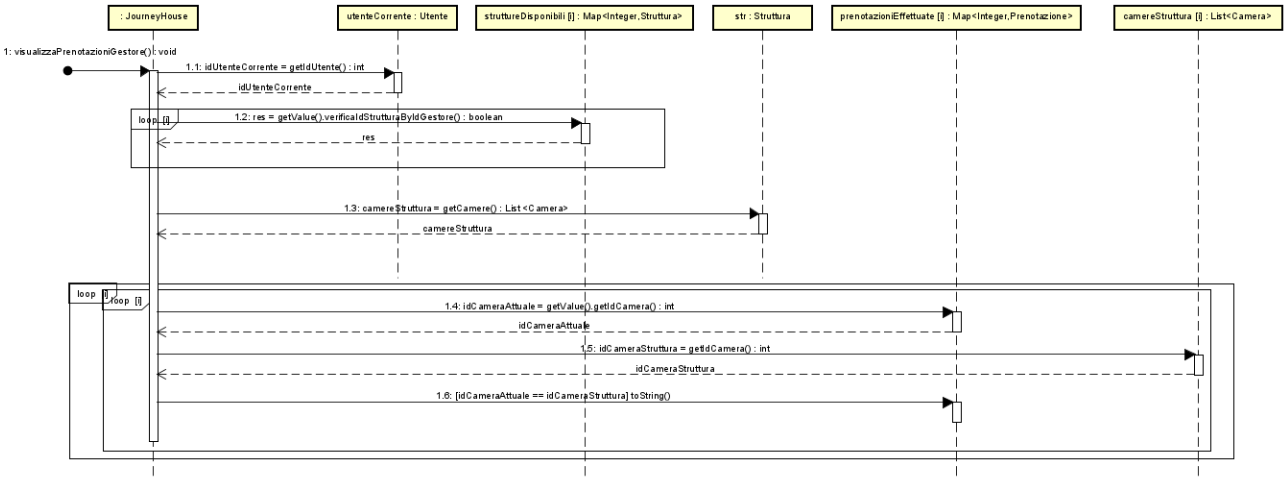
UC9 – eliminaPrenotazione



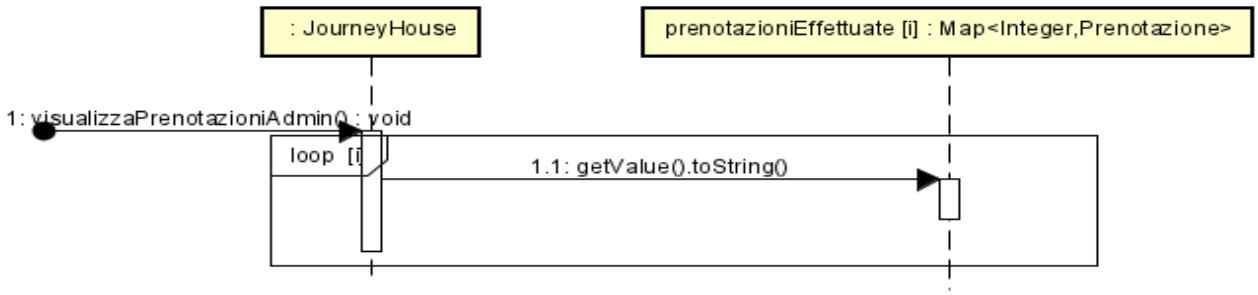
UC9 – visualizzaPrenotazioniCliente



UC9 – visualizzaPrenotazioniGestore



UC9 – visualizzaPrenotazioniAdmin

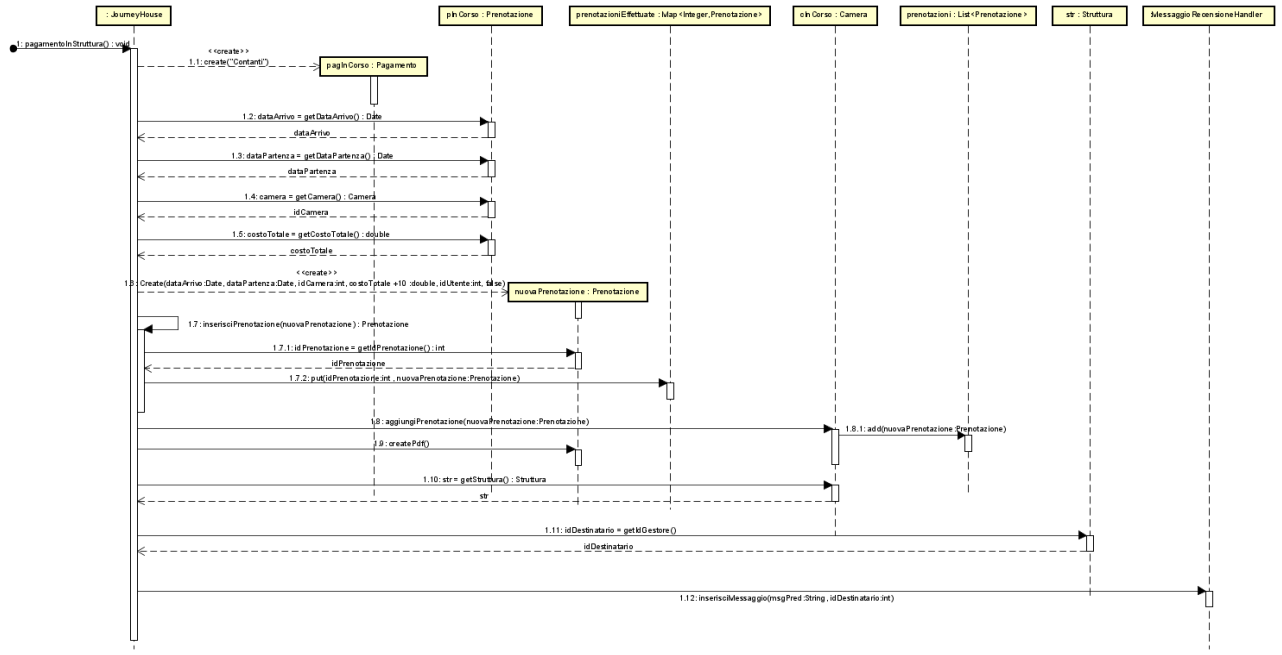


Iterazione 3:

Dopo attente valutazioni è stato ritenuto opportuno utilizzare il controller di caso d'uso *MessaggioRecensioneHandler* che permette di non appesantire *:JourneyHouse*, permette inoltre di ottenere una coesione migliore e di delegare il lavoro da eseguire per quanto riguarda i casi di uso presi in esame in questa iterazione al suddetto controller di caso d'uso.

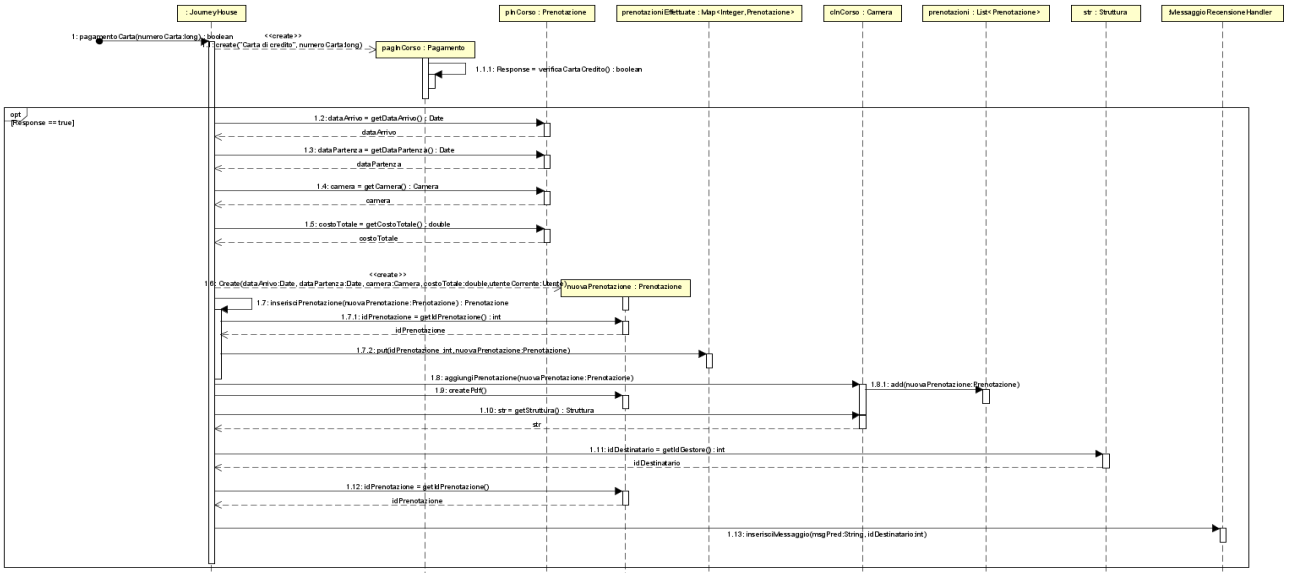
Vengono inoltre riportati gli SD aggiornati di *pagamentoInStruttura*, *pagamentoCarta* e *eliminaPrenotazione* in quanto una volta effettuata una prenotazione o l'eliminazione della stessa viene inviato un messaggio al gestore corrispondente per informarlo dell'evento occorso.

UC1 – pagamentoInStruttura (aggiornamento)



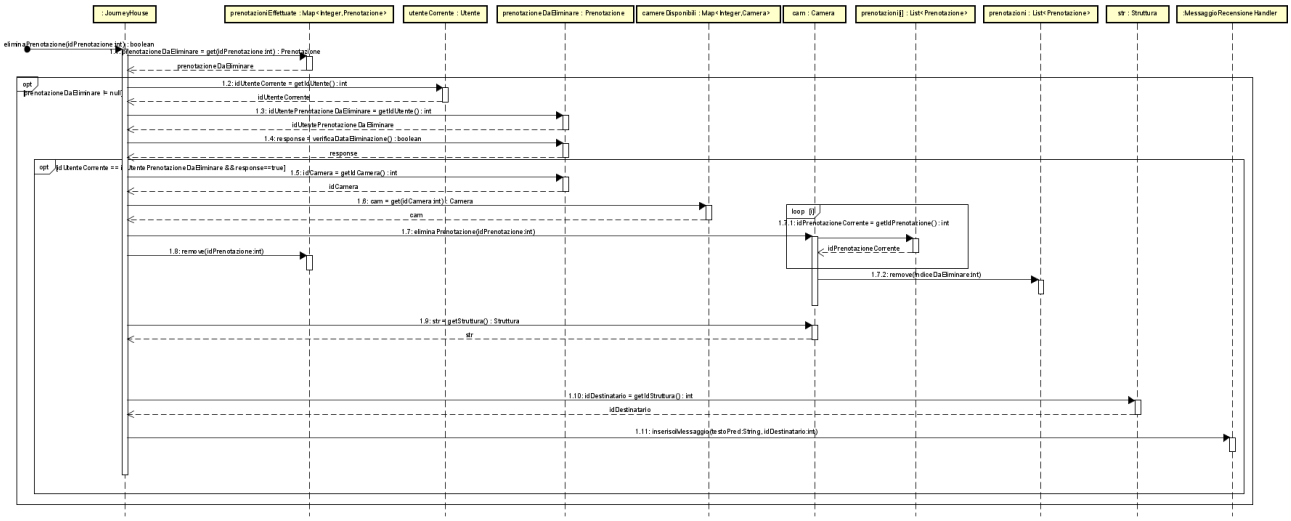
Per un maggiore dettaglio grafico si consulti l'allegato **A2**.

UC1 – pagamentoCarta (aggiornamento)



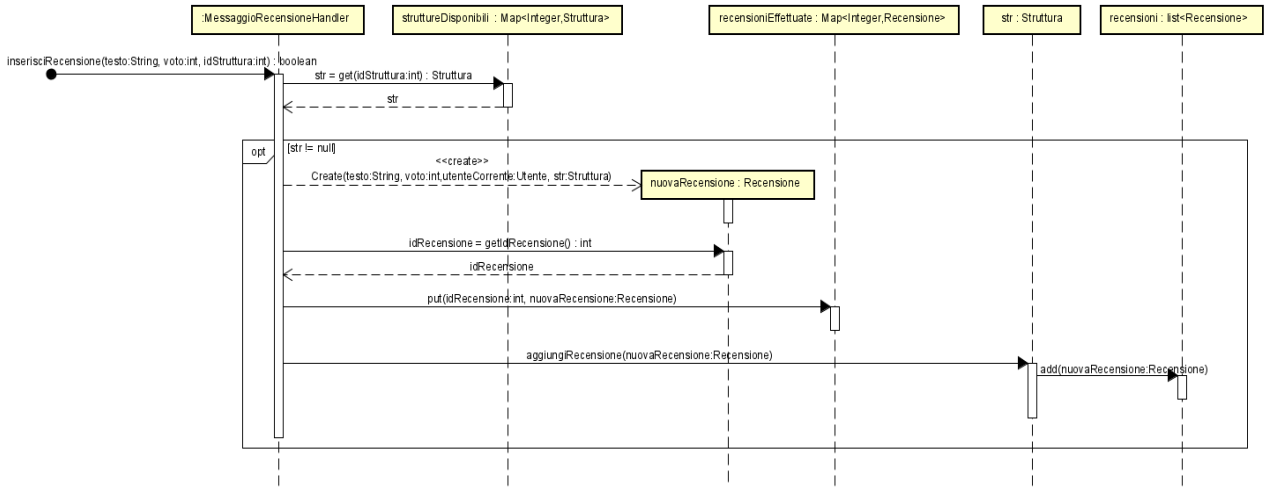
Per un maggiore dettaglio grafico si consulti l'allegato **A3**.

UC3 – eliminaPrenotazione (aggiornamento)

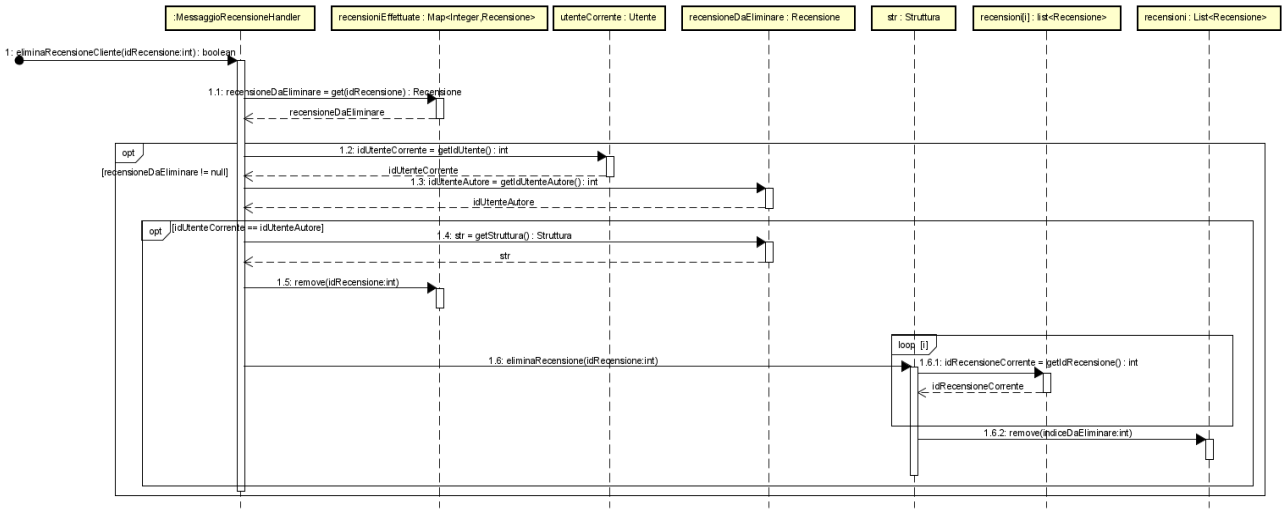


Per un maggiore dettaglio grafico si consulti l'allegato **A4**.

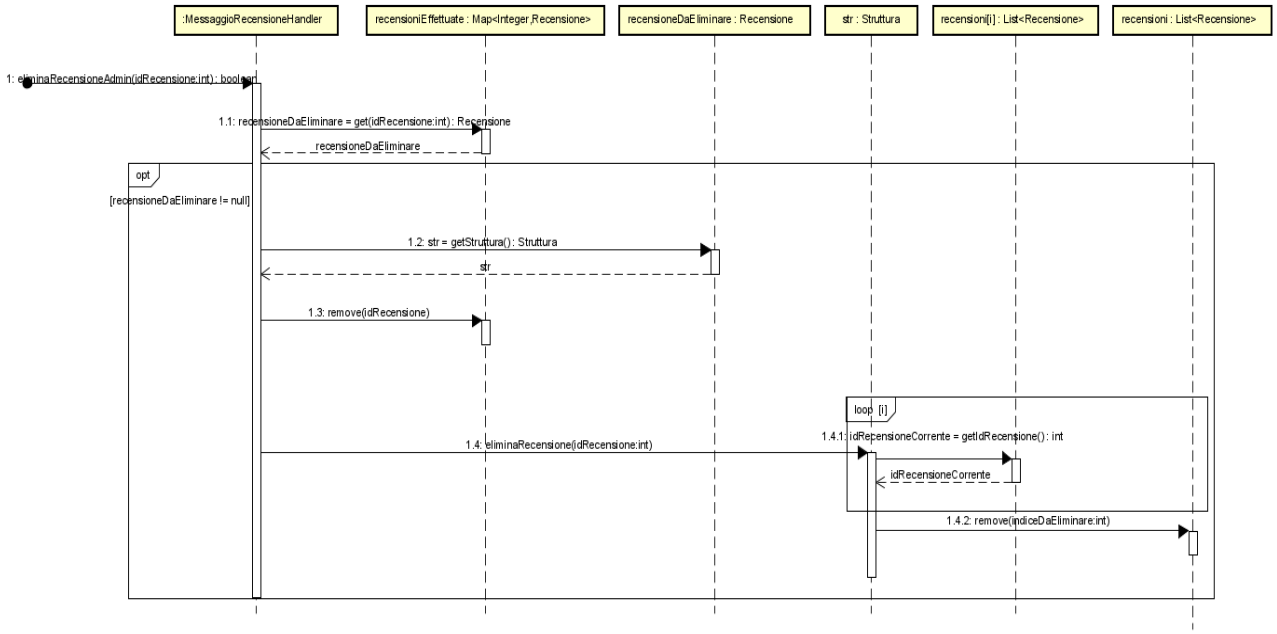
UC10 – inserisciRecensione



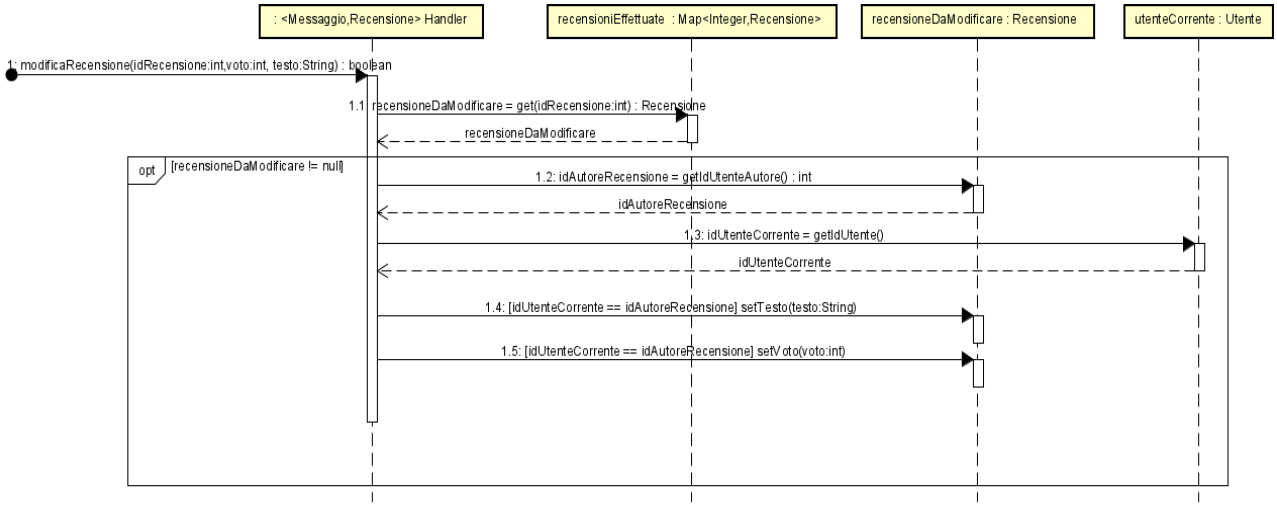
UC10 – eliminaRecensioneCliente



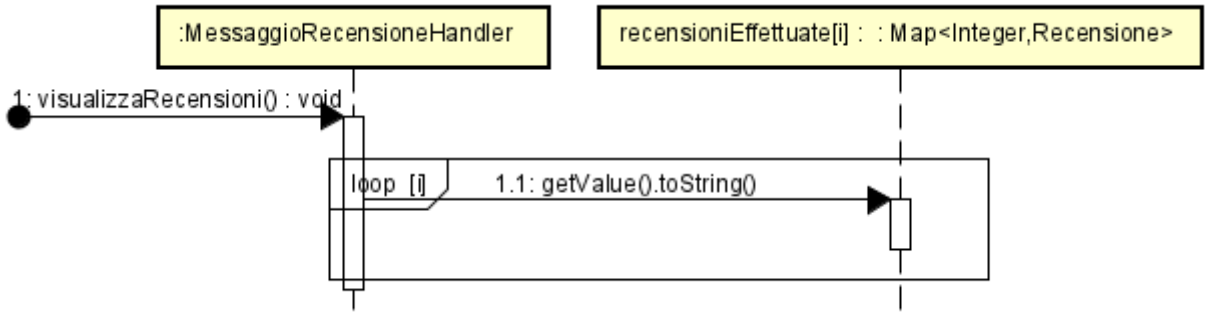
UC10 – eliminaRecensioneAdmin



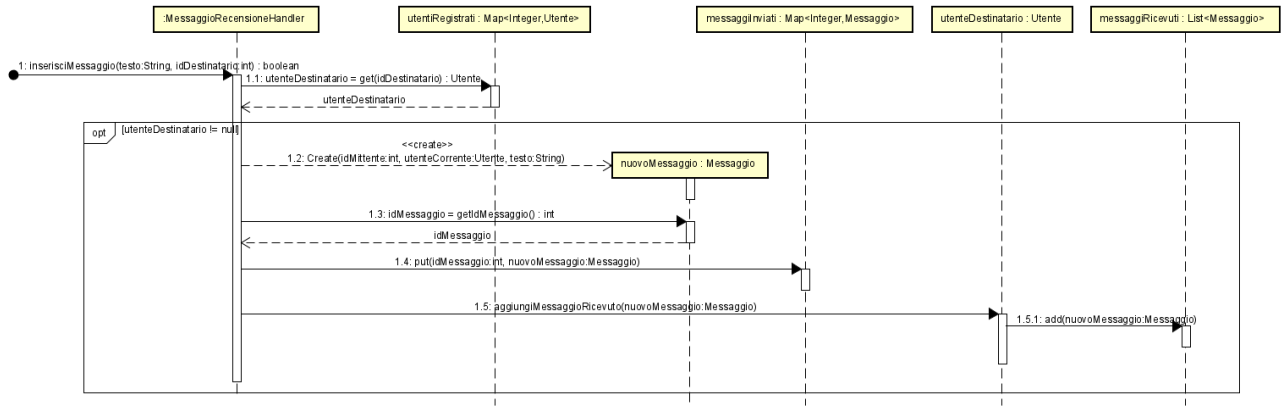
UC10 – modificaRecensione



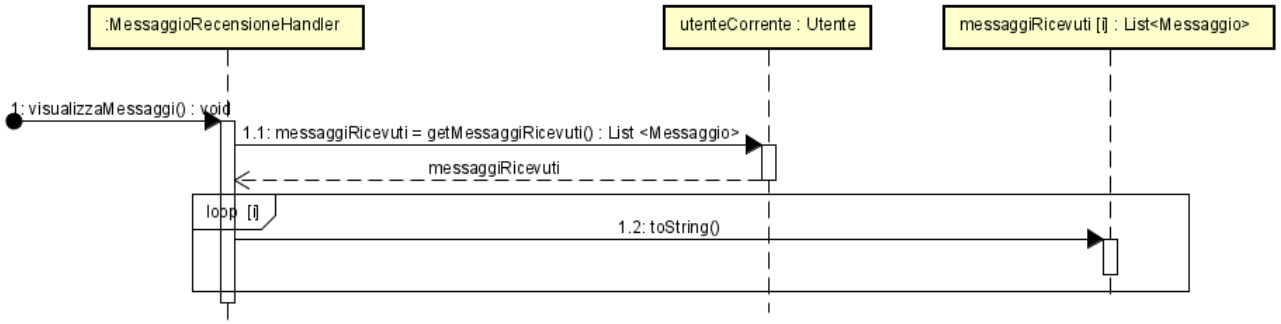
UC10 – visualizzaRecensioni



UC12 – inserisciMessaggio



UC12 – visualizzaMessaggi

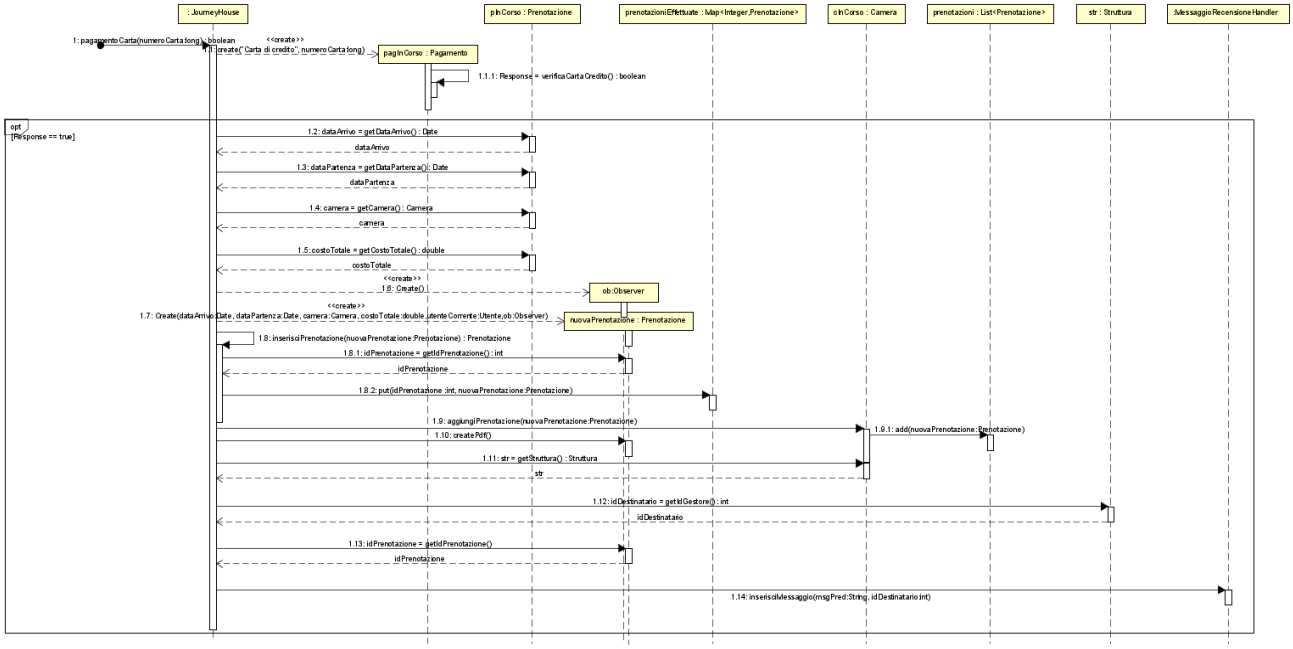


Iterazione 4:

Si osservi in particolare che nella progettazione del caso d'uso *Aggiornamento Stato Prenotazione (UC13)* si è scelto di sfruttare il pattern GoF **Observer.** Si utilizza inoltre un altro controller di caso d'uso *LoginRegGestReportHandler* per gli stessi motivi citati precedentemente.

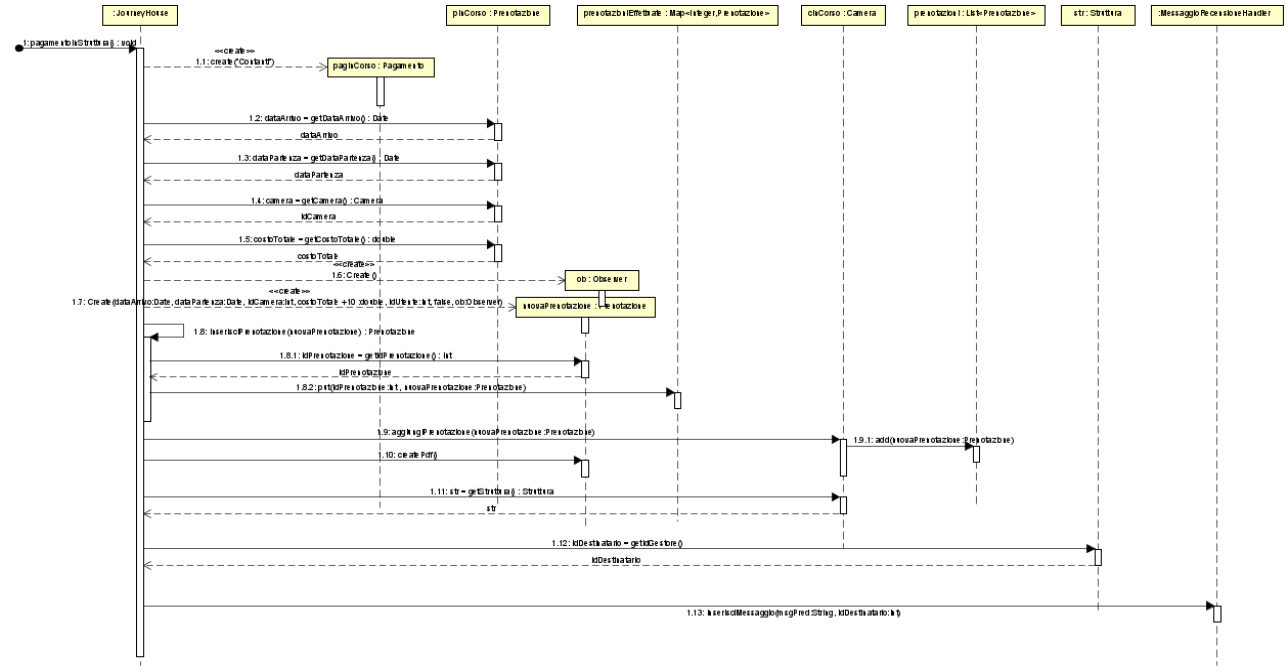
Vengono riportati gli SD aggiornati di *pagamentoInStruttura*, *pagamentoCarta* in quanto viene aggiunto alla prenotazione effettuata l'observer atto a monitorare lo stato della prenotazione stessa.

UC1 – PagamentoCarta (Aggiornamento)



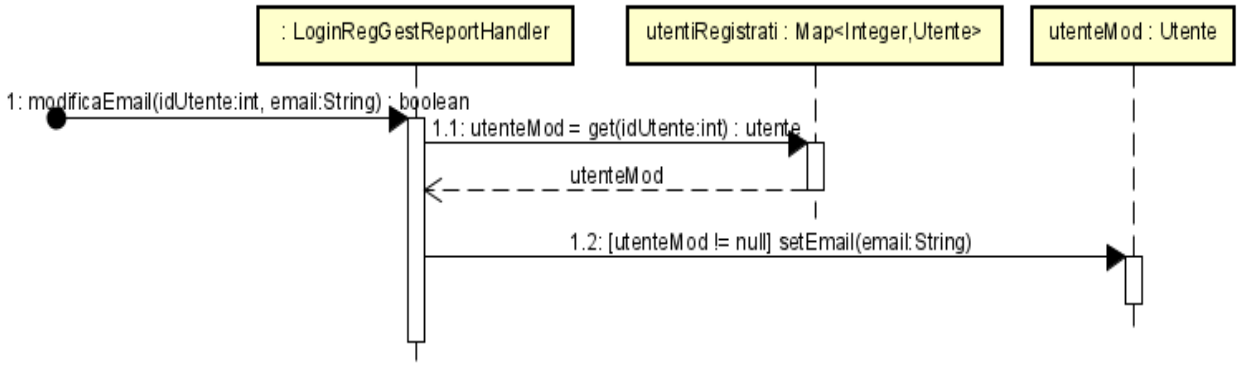
Per un maggiore dettaglio grafico si consulti l'allegato **A5**.

UC1 – PagamentoStruttura (Aggiornamento)

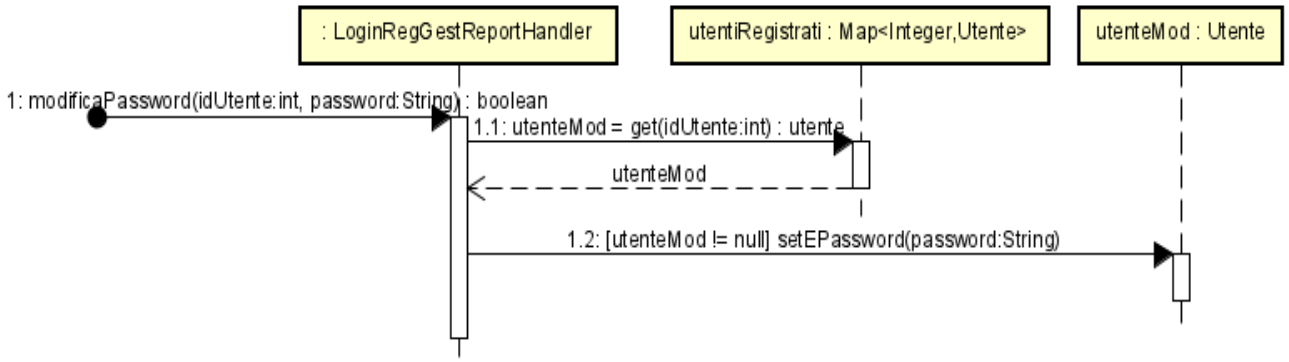


Per un maggiore dettaglio grafico si consulti l'allegato **A5**.

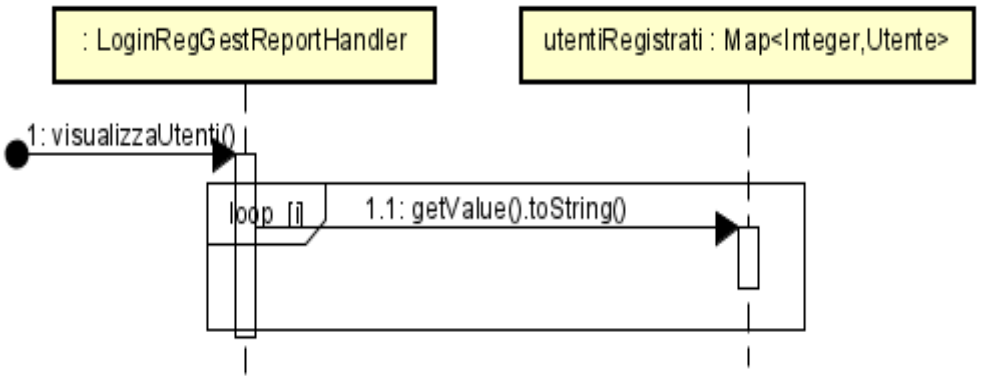
UC11 – modificaEmail



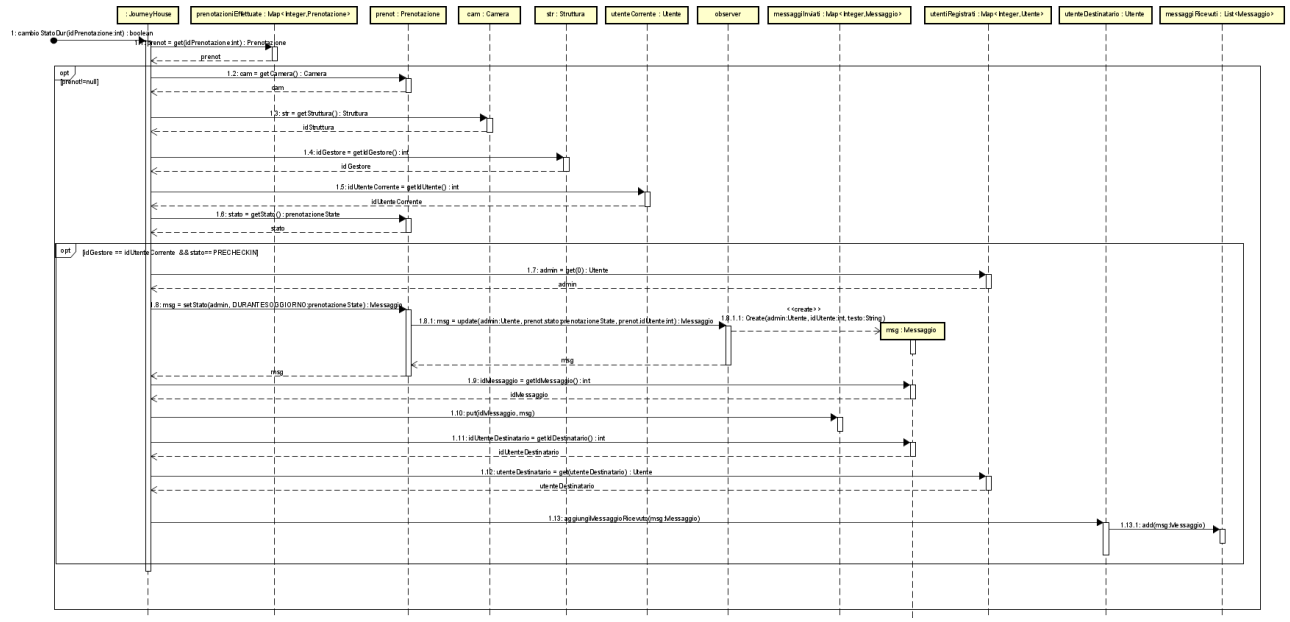
UC11 – modificaPassword



UC11 – visualizzaUtenti

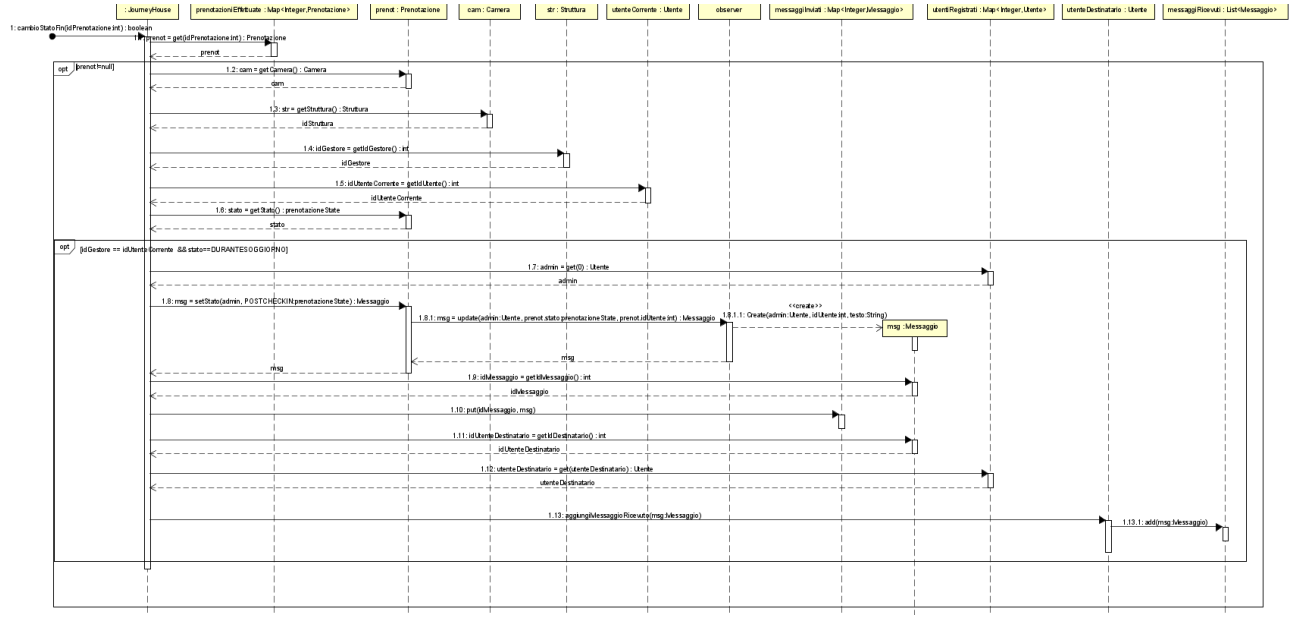


UC13 – cambiaStatoDur



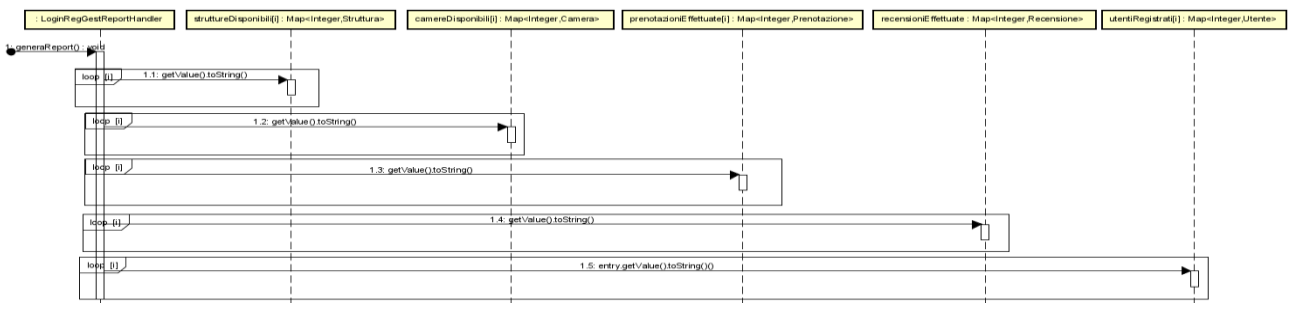
Per un maggiore dettaglio grafico si consulti l'allegato **A7**.

UC13 – cambiaStatoFin

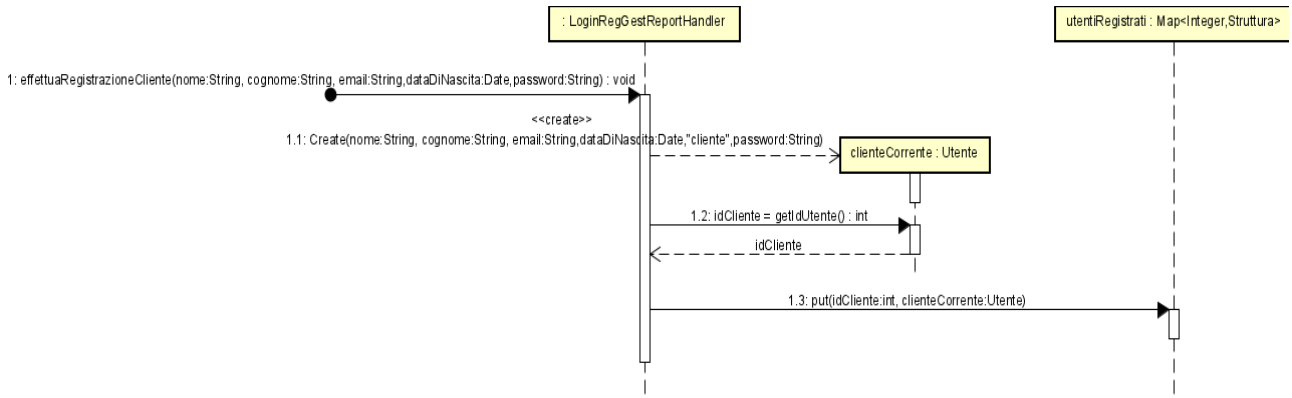


Per un maggiore dettaglio grafico si consulti l'allegato **A7**.

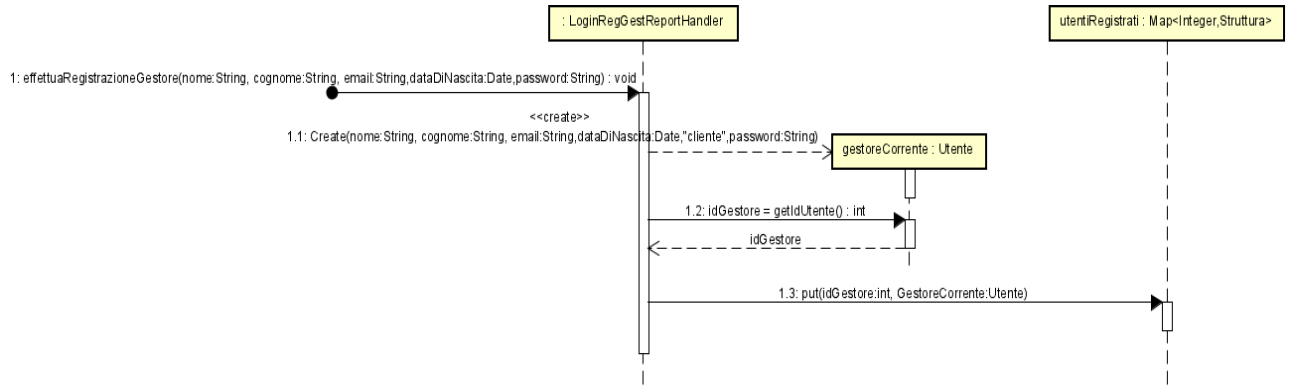
UC14 – generaReport



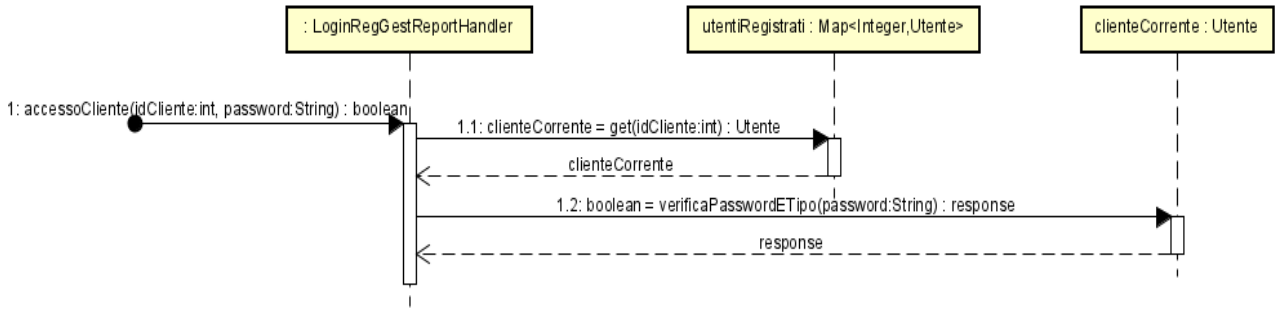
UC3 – effettuaRegistrazioneCliente



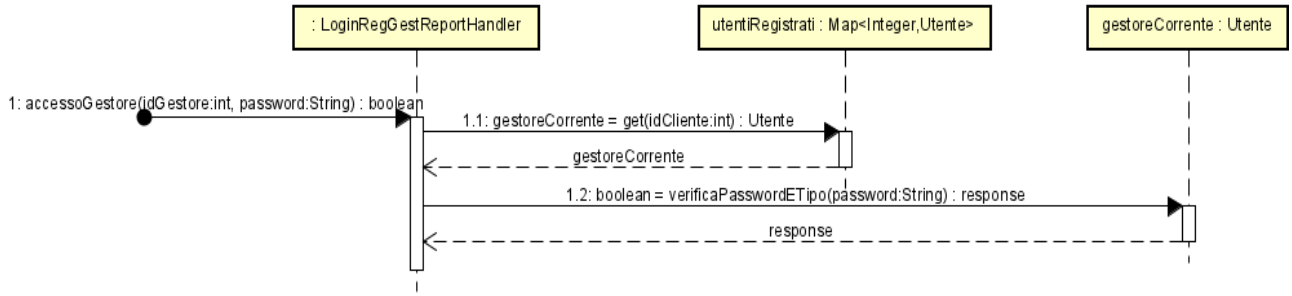
UC4 – effetturaRegistrazioneGestore



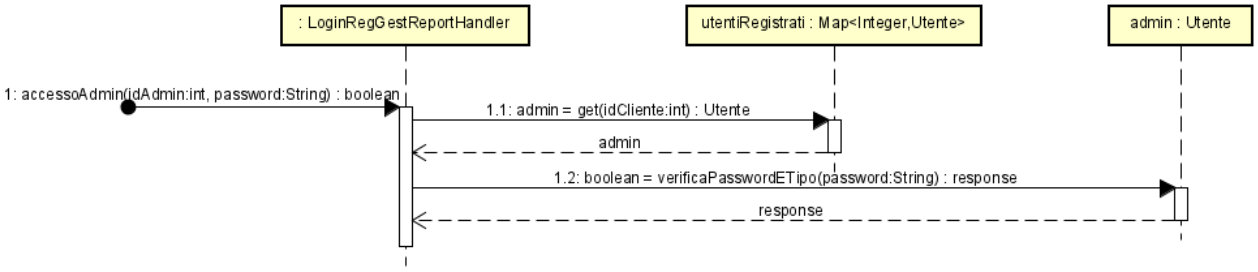
UC5 – accesso cliente



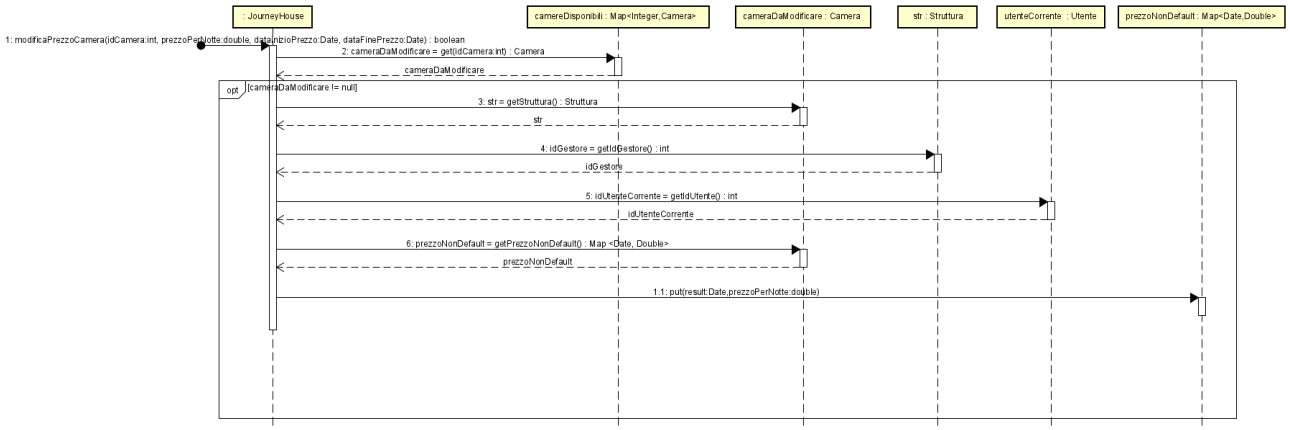
UC6 – accessoGestore



UC7 – accessoAdmin



UC8 – modificaPrezzoCamera AGGIORNAMENTO



4. Testing

Il testing è una fase essenziale del processo di sviluppo di un programma robusto ed efficiente. La probabilità di rilevare malfunzionamenti aumenta all'aumentare dei testi eseguiti. Si vuole dunque rendere il software realizzato funzionante e funzionale.
Sono stati effettuati dei testing di tipo "Unit Test", attenzionando quindi i metodi, visti nella loro singolarità.
I metodi sono stati testati in modo esaustivo, analizzando per ciascuno, ove possibile, tutte le casistiche che si potrebbero verificare.
In particolar modo i testing sono stati realizzati tramite l'ausilio del framework JUnit5: un JUnit Test è una classe java contenente un insieme di metodi attraverso i quali verificare la correttezza del codice in analisi. Tra questi metodi JUnit, quelli più utilizzati sono gli appartenenti alla categoria *Assert*, utili a confrontare il risultato del metodo in analisi con un risultato atteso, in base anche al tipo di *Assert* stesso. Oltre a questo tipo di test sono stati svolti dei test manuali del sistema lanciando l'applicazione. Non sono emersi errori nel funzionamento.

CameraTests

- **TEST CALCOLA PREZZO TOTALE**:
  - **testCalcolaPrezzo:** Calcola il prezzo totale di un soggiorno. Il prezzo totale di un soggiorno di due notti in una camera che ha un prezzo per notte di 20 euro è pari a 42 euro (comprensivo delle tasse). Si verifica allora il corretto calcolo del totale.

- **TEST VERIFICA TIPO:**
  - **testVerificaTipo:** Nel test, viene creata un'istanza di Struttura e di camera, viene quindi chiamato il metodo *verificaTipo (tipologiaCamera)*. Il test utilizza gli assert *assertTrue ()* e *assertFalse ()* per verificare se la funzione restituisce il risultato atteso. Nel caso di un test positivo (la camera è dello stesso tipo di quello passato come argomento), ci si aspetta che la funzione restituisca true mentre nel caso negativo ci si aspetta che la funzione restituisca false.

StrutturaTests

- **VERIFICA ESISTENZA CAMERA**:
  - **testVerificaEsistenzaCamera:** Nel test, viene creata un'istanza di Struttura e di camera, viene quindi chiamato il metodo *verificaEsistenzaCamera (numCamera)*. Il test utilizza gli assert *assertTrue ()* e *assertFalse ()* per verificare se la funzione restituisce il risultato atteso. Nel caso di un test positivo (la struttura possiede la camera passata come argomento), ci si aspetta che la funzione restituisca true mentre nel caso negativo ci si aspetta che la funzione restituisca false.

- **TEST VERIFICA STRUTTURA BY CITTA':**
  - **testVerificaStrutturaByCittà:** Nel test, viene creata un'istanza di Struttura, viene quindi chiamato il metodo *verificaStrutturaByCittà (Città)*. Il test utilizza gli assert *assertTrue ()* e *assertFalse ()* per verificare se la funzione restituisce il risultato atteso. Nel caso di un test positivo (la struttura risiede nella città passata come argomento), ci si aspetta che la funzione restituisca true mentre nel caso negativo ci si aspetta che la funzione restituisca false.

- **TEST VERIFICA STRUTTURA BY GESTORE:**
  - **testVerificaStrutturaByGestore:** Nel test, viene creata un'istanza di Struttura, viene quindi chiamato il metodo *verificaStrutturaByIdGestore (idGestore)*. Il test utilizza gli assert *assertTrue ()* e *assertFalse ()* per verificare se la funzione restituisce il risultato atteso. Nel caso di un test positivo (la struttura è gestita dal gestore passato come argomento), ci si aspetta che la funzione restituisca true mentre nel caso negativo ci si aspetta che la funzione restituisca false.

PrenotazioneTests

- **TEST VERIFICA DISPONIBILITA'**:
  - **testVerificaDisponibilita:** La funzione *verificaDisponibilita (dataArrivoReq, dataPartenzaReq)* restituisce un valore booleano true se la prenotazione coincide con la data richiesta dall'utente altrimenti false. Nel test viene creata un'istanza di Prenotazione, viene quindi chiamato il metodo *verificaDisponibilita (dataArrivoReq, dataPartenzaReq)* con due date differenti, nel caso di test positivo (la data passata come argomento coincide con quella della prenotazione), ci si aspetta che la funzione restituisca true viceversa ci si aspetta che la funzione restituisca false.

- **TEST VERIFICA DATA ELIMINAZIONE:**
  - **testVerificaDataEliminazione:** La funzione *verificaDataEliminazione ()* restituisce un valore booleano true se la prenotazione ha una distanza di almeno tre giorni dalla data della prenotazione altrimenti false. Nel test viene creata un'istanza di Prenotazione, viene quindi chiamato il metodo *verificaDataEliminazione ()*, nel caso di test positivo (la differenza di giorni è superiore a tre), ci si aspetta che la funzione restituisca true viceversa ci si aspetta che la funzione restituisca false.

JourneyHouseTests

- **TEST COSTRUTTORE**:
  - **testJourneyHouse:** È stata istanziata un'unica classe JourneyHouse ed è stato controllato il corretto funzionamento del costruttore stesso.

- **TEST RICERCA CAMERA**:
  - **testRicercaCamera:** La funzione *ricercaCamera (città, dataArrivo, dataPartenza, tipologiaCamera)* stampa le camere disponibili in base alle richieste del cliente e restituisce una lista di interi contenenti gli id delle camere disponibili. Nel test si verifica che le camere disponibili per le richieste inserite siano in numero pari a 2.

- **TEST SCELTA CAMERA E PAGAMENTO ONLINE:**
  - **testFinePrenotazione:** Viene scelta una camera e viene scelto il pagamento tramite carta di credito, si verifica infine che la prenotazione si andata a buon fine.

- **TEST INSERIMENTO CAMERA**:
  - **testInserisciCamera:** La funzione *inserisciCamera (numeroCamera, numeroPiano, tipoCamera, prezzoPerNotte)* restituisce un valore booleano true se l'inserimento della camera è avvenuto con successo viceversa false nel caso in cui il numero della camera inserita dall'utente è già esistente. Viene chiamata la funzione *inserisciCamera (numeroCamera, numeroPiano, tipoCamera, prezzoPerNotte),* viene utilizzato l'id corretto e dunque viene inserita la camera, si utilizza dunque *assertTrue* per verificare la restituzione del parametro true.

- **TEST MODIFICA CAMERA**:
  - **testModificaCamera:** La funzione *modificaCamera (idCamera, prezzoPerNotte)* restituisce un valore booleano true se la modifica della camera è avvenuta con successo viceversa false nel caso in cui l'id della camera non è esistente. Viene chiamata la funzione *modificaCamera (idCamera, prezzoPerNotte),* viene utilizzato l'id corretto e dunque viene modificato il prezzo della camera, si utilizza dunque *assertTrue* per verificare la restituzione del parametro true.

- **TEST ELIMINA PRENOTAZIONE:**
  - **testEliminaPrenotazione:** La funzione *eliminaPrenotazione (idPrenotazione)* restituisce un valore booleano true se l'eliminazione della prenotazione è avvenuta con successo viceversa false nel caso in cui l'id della prenotazione inserito dall'utente non dovesse esistere o nel caso in cui l'utenteCorrente non è l'autore della prenotazione che si desidera eliminare. Viene chiamata tre volte la funzione *eliminaPrenotazione (idPrenotazione)*, la prima volta viene utilizzato l'id corretto e dunque viene eliminata la prenotazione, si utilizza dunque *assertTrue*

per verificare la restituzione del parametro true, nel secondo test si inserisce un id errato e si utilizza *assertFalse* per verificare se la funzione restituisce false, nel terzo test si inserisce l'id di una prenotazione che non appartiene all'utenteCorrente.

MessaggioRecensioneHandlerTests

- **TEST INSERIMENTO RECENSIONE**:
  - **testInserimentoRecensione:** La funzione *inserisciRecensione (testo, voto, idStruttura)* restituisce un valore booleano true se l'inserimento della recensione è avvenuto con successo viceversa false nel caso in cui l'id inserito dall'utente non dovesse esistere. Viene chiamata due volte la funzione *inserisciRecensione (testo, voto, idStruttura),* la prima volta viene utilizzato l'id corretto e dunque viene inserita la recensione, si utilizza dunque *assertTrue* per verificare la restituzione del parametro true, nel secondo test si inserisce un id errato e si utilizza *assertFalse* per verificare se la funzione restituisce false.

- **TEST MODIFICA RECENSIONE:**
  - **testModificaRecensione:** La funzione *modificaRecensione (idRecensione, testo, voto)* restituisce un valore booleano true se la modifica della recensione è avvenuta con successo viceversa false nel caso in cui l'id della recensione inserito dall'utente non dovesse esistere o nel caso in cui l'utenteCorrente non è l'autore della recensione di cui si desidera effettuare la modifica. Viene chiamata tre volte la funzione *modificaRecensione (idRecensione, testo, voto),* la prima volta viene utilizzato l'id corretto e dunque viene modificata la recensione, si utilizza dunque *assertTrue* per verificare la restituzione del parametro true, nel secondo test si inserisce un id errato e si utilizza *assertFalse* per verificare se la funzione restituisce false, nel terzo test si inserisce l'id di una recensione che non appartiene all'utenteCorrente.

- **TEST ELIMINA RECENSIONE:**
  - **testEliminaRecensione:** La funzione *eliminaRecensione (idRecensione)* restituisce un valore booleano true se l'eliminazione della recensione è avvenuta con successo viceversa false nel caso in cui l'id della recensione inserito dall'utente non dovesse esistere o nel caso in cui l'utenteCorrente non è l'autore della recensione da eliminare Viene chiamata tre volte la funzione *eliminaRecensione (idRecensione),* la prima volta viene utilizzato l'id corretto e dunque viene eliminata la recensione, si utilizza dunque *assertTrue* per verificare la restituzione del parametro true, nel secondo test si inserisce un id errato e si utilizza *assertFalse* per verificare se la funzione restituisce false, nel terzo test si inserisce l'id di una recensione che non appartiene all'utenteCorrente.

- **TEST INSERIMENTO MESSAGGIO**:
  - **testInserimentoMessaggio:** La funzione *inserisciMessaggio (testo, idUtenteDestinatario)* restituisce un valore booleano true se l'inserimento del messaggio è avvenuto con successo viceversa false nel caso in cui l'id inserito dall'utente non dovesse esistere. Viene chiamata due volte la funzione *inserisciMessaggio (testo, idUtenteDestinatario),* la prima volta viene utilizzato l'id corretto e dunque viene inserito il messaggio, si utilizza dunque *assertTrue* per verificare la restituzione del parametro true, nel secondo test si inserisce un id di un utente destinatario errato e si utilizza *assertFalse* per verificare se la funzione restituisce false.

LoginRegGestReportHandlerTests

- **TEST LOGIN**:
  - **testAccesso:** Nel test viene creata un'istanza di Utente (Cliente, Gestore o Admin) viene quindi chiamato il metodo *accesso (id, password)* effettuando il test inserendo id e password corretti, id corretto ma password sbagliata, id errato ed infine viene differenziata la tipologia di login, ad esempio, un utente cliente non può effettuare il login da admin. Nel caso di test

positivo (accesso effettuato correttamente), ci si aspetta che la funzione restituisca true viceversa ci si aspetta che la funzione restituisca false.

- **TEST MODIFICA PASSWORD:**
  - **testModificaPassword:** La funzione *modificaPassword (idUtente, nuovaPassword)* restituisce un valore booleano true se l'operazione di modifica è stata eseguita con successo viceversa false. Viene chiamata due volte la funzione *modificaPassword (idUtente, nuovaPassword),* la prima volta viene utilizzato l'id corretto e dunque viene assegnata una nuova password, si utilizza dunque *assertTrue* per verificare la restituzione del parametro true nel secondo test si inserisce un id errato e si utilizza *assertFalse* per verificare se la funzione restituisce false.

- **TEST MODIFICA EMAIL:**
  - **testModificaEmail:** La funzione *modificaEmail (idUtente, nuovaEmail)* restituisce un valore booleano true se l'operazione di modifica è stata eseguita con successo viceversa false. Viene chiamata due volte la funzione *modificaEmail (idUtente, nuovaEmail),* la prima volta viene utilizzato l'id corretto e dunque viene assegnata una nuova e-mail, si utilizza dunque *assertTrue* per verificare la restituzione del parametro true nel secondo test si inserisce un id errato e si utilizza *assertFalse* per verificare se la funzione restituisce false.

5. Pattern

Per la realizzazione del sistema JourneyHouse sono stati utilizzati diversi pattern per migliorare l'architettura e la struttura del sistema. In particolare, sono stati impiegati il pattern Singleton, il pattern Controller e il pattern Observer.

Il pattern GoF (Gangs of Four) Singleton di tipo creazionale è stato utilizzato per garantire l'esistenza di una sola istanza della classe principale, JourneyHouse, che rappresenta il nucleo del sistema.

Al fin di evitare che la classe JourneyHouse diventasse troppo pesante e accumulasse troppe responsabilità, è stato utilizzato il pattern Controller di tipo GRASP (General Responsibility Assignment Software Patterns). Sono stati creati due Use Case Controller, uno per gestire le logiche di Messaggi e Recensioni e l'altro per gestire le logiche di Accesso, Registrazione, Report e Gestione Utente. Questo permette una maggiore modularità, facilità di manutenzione e migliore separazione delle responsabilità all'interno del sistema.

Infine, per monitorare il cambiamento dello stato della prenotazione e reagire di conseguenza, è stato utilizzato il pattern GoF Observer di tipo comportamentale. È stato identificato un oggetto osservatore che si registra per ricevere notifiche quando lo stato della prenotazione cambia, questo permette di eseguire le azioni necessarie in risposta ai cambiamenti di stato delle prenotazioni.

L'utilizzo di tali pattern ha contribuito a migliorare la struttura, l'organizzazione e la flessibilità del sistema JourneyHouse, ha permesso di separare le responsabilità, evitare duplicazioni di risorse, semplificare la manutenzione e facilitare l'aggiunta di nuove funzionalità in futuro.