UNIVERSITÀ DEGLI STUDI
DI SALERNO

SYSTEM DESIGN DOCUMENT

Stefano Santoro – Giovanni Riccardi
-Renato Natale – Samuele Valiante

Coordinatore del progetto:

Nome	Matricola
Stefano Santoro	0512120778

Partecipanti:

Nome	Matricola
Giovanni Riccardi	0512119392
Renato Natale	0512119641
Samuele Valiante	0512119125

Scritto da:	Giovanni Riccardi

Revision History

Data	Versione	Descrizione	Autore
25/11/2025	1.0	Prima versione SDD	Giovanni Riccardi
02/02/2026	2.0	Modifiche Boundary Conditions e altre minori	Giovanni Riccardi

Sommario

INTRODUCTION

1.1 Purpose of the system

Il sistema software è progettato per la gestione alberghiera dell'hotel Colossus, che demanda un'architettura modulare e distribuita, pensata per supportare le diverse figure operative che lavorano all'interno dell'hotel. L'obiettivo principale dell'architettura è fornire una piattaforma unica, coerente, capace di gestire in modo integrato tutte le attività amministrative, operative e ristorative della struttura.

L'applicazione è organizzata in più sottosistemi cooperanti: gestione delle camere, gestione dei clienti e delle prenotazioni, gestione dello staff, contabilità, front desk, housekeeping .Questi sottosistemi comunicano tra loro attraverso delle interfacce che forniscono servizi esponendo operazioni ben definite, così da mantenere basso l'accoppiamento e permettere evoluzioni future senza impatti sull'intera applicazione.

L'architettura segue un modello client–server: lato client sono presenti le interfacce grafiche dedicate ai vari attori (front-desk, governante, manager), mentre lato server risiede la logica applicativa centrale che coordina i flussi dei dati, applica le regole di business ed effettua l'accesso al database. Il sistema software permette di mantenere separati l'aspetto visuale-grafico, la logica di controllo e la persistenza, garantendo manutenibilità e possibilità di estensioni future.

Nel complesso, la soluzione è pensata per essere portabile, sicura e intuitiva da usare. L'obiettivo che ci poniamo a raggiungere nel design è ottenere un sistema robusto, semplice da mantenere e facilmente estendibile, capace di adattarsi il più facilmente possibile a cambiamenti futuri qual ora fossero necessari. Infine, il documento serve come riferimento per lo sviluppo e l'implementazione del software, fornendo una guida chiara per i progettisti e futuri manutentori.

1.2 Design Goals

DESIGN GOAL DISCUSSION

I Design goals sono stati creati prendendo come base i requisiti non funzionali specificati durante la fase di requirements elicitation.
A seguito di un'analisi approfondita e di un confronto strutturato, è stato deciso di attribuire priorità strategica a due aspetti fondamentali: **manutenibilità e velocità operativa.** Poiché, essendo un software per la gestione alberghiera, la velocità delle operazioni è un requisito importante . Il sistema non deve essere la causa di rallentamenti durante i momenti più concitati e di affluenza.
Tuttavia, pur riconoscendo l'importanza delle prestazioni, **il focus principale** della progettazione è stato posto sulla manutenibilità e sull'integrità del sistema, considerate aspetti ancora più critici in una prospettiva di lungo periodo. Il software è infatti pensato per evolvere nel tempo e per supportare futuri ampliamenti funzionali senza compromettere la stabilità dell'architettura esistente.

per soddisfare questo proposito sarà dedicata una maggiore attenzione sulla modularità del codice e per facilitare aggiornamenti futuri vengono usati pattern come Observer pattern(gestione camere) e Command pattern usato per la maggior parte delle funzionalità e Composite pattern (conto economico).
Queste scelte consentono di facilitare la manutenzione, ridurre l'impatto delle modifiche e supportare l'evoluzione del sistema nel tempo.

1.2.1 Portability

Il sistema deve essere progettato per minimizzare le differenze tra specifici sistemi operativi (come Windows, macOS, Linux), architetture di processore o configurazioni di rete. Il codice sorgente richiede grazie all'ambiente fornito da java modifiche minime o nulle per essere compilato ed eseguito in ambienti eterogenei, basta infatti avere una versione superiore a java 17 per garantire la completa funzionalità nella esecuzione.

1.2.2 Transparency

Il sistema deve essere distribuito con un architettura **client-server three tier** e deve essere sviluppato di modo che l'utente finale non percepisca questa proprietà. Il sistema distribuito deve essere inteso come un unico sistema logico.

1.2.3 Performance

- **Response time (Priorità media)**

Le operazioni quotidiane: check-in, assegnazione camere, segnalazione pulizie devono essere confermate sotto il minuto per garantire un'esperienza utente immediata per il personale alberghiero. Questo è fondamentale per evitare colli di bottiglia durante i picchi di attività . Qualsiasi ritardo percepibile oltre il limite specificato comporterebbe un aumento dei tempi di attesa per gli ospiti e una diminuzione della produttività del personale.

- **Throughput**

Il sistema deve supportare la concorrenza di più operazioni (es. front desk che registra un cliente mentre la governante aggiorna lo stato di una camera).Le operazioni sono viste come transazioni e come tali devono rispettare le proprietà ACID : Il sistema deve gestire efficacemente il carico di lavoro complessivo, mantenendo l'integrità dei dati e garantendo che le risorse condivise (come il database) abbiamo un failure rate più basso possibile

1.2.4 Dependability

- **Robustness (Priorità Alta)**

Deve gestire input errati e segnalarli all'utente ad esempio in una registrazione qualsiasi quando il front desk inserisce dati errati gli viene notificato da interfaccia e gli viene data la possibilità di rimediare.

- **Reliability (Priorità Alta)**

Le informazioni mostrate devono sempre corrispondere alla realtà operativa e a tutte le funzionalità che le riguardano.

- **Fault tolerance (Priorità media)**

Il sistema deve poter fornire un certo grado di tolleranza ai malfunzionamenti, garantendo che se un sottosistema smetta di funzionare, non impatti significativamente verso gli altri.

Fault tolerance vs trasparency ---

Dato che Front-Desk e governante accedono alle stesse risorse, un malfunzionamento da parte di una componente condivisa da uno dei due lati deve essere percepito dall'altro come malfunzionamento generale nascondendogli la provenienza di esso, poichè non è di suo interesse.

- **Security (Alta)**

Il sistema deve garantire accesso differenziato per ruolo (front desk, manager, governante. Nessun dipendente deve poter accedere a dati non pertinenti e non

riguardanti esso.
La protezione dei dati è un aspetto altrettanto importante: il sistema deve prevenire attacchi come SQL injection e crittografazione delle password nel database. (NFR9, NFR10)

1.2.7 Maintenance

- **Modifiability (Alta)**

Deve essere garantita la facilità di aggiornare regole di business già esistenti. Questo quando vi sono dei cambiamenti in una funzionalità o quando vi è un cambiamento delle politiche di operazione dei vari ruoli.

- **Readability (Alta)**

Il codice dei vari moduli deve essere chiaro, commentato e documentato per facilitare manutenzione e aggiornamenti da parte di sviluppatori futuri.

Performance vs. Maintenance

Verranno usati algoritmi e pattern che garantiscono che il codice sia più manutenibile possibile , la modularizzazione del codice e l'uso di interfacce per separare i compiti interni del server è cruciale per ottenere una architettura chiusa e robusta. La performance anche se importante, non è prioritaria dato che il codice organizzato in questa modalità , non garantisce performance eccellenti. Tuttavia verrà data una attenzione in più nelle strutture dati e nel codice utilizzato per evitare di implementare algoritmi inefficienti che gravano sulle performance generali.

1.2.8 End user criteria

- **Utility**

Il sistema deve supportare concretamente il lavoro quotidiano dell'impiegato fornendogli chiaramente uno stato dell'hotel riguardo le sue mansioni e facilitandogliele di modo da essere più efficiente nello svolgimento di quest'ultime. (Esempio reminder in primo piano nella Dashboard del front desk delle prenotazioni imminenti)

- **Usability**

Le attività alberghiere vengono spesso svolte in condizioni di lavoro dinamiche e sotto pressione; per questo motivo l'interfaccia è stata progettata per essere semplice e pulita, intuitiva e orientata all'azione immediata con l'obbiettivo di essere meno ambigua possibile. Sarà adottato un approccio "clicca e inserisci", evitando interazioni complesse o ambigue, al fine di ridurre al minimo gli errori operativi durante l'attività quotidiana.

Per la gestione visiva degli errori dettati dall'utente Il design privilegia schermate chiare, suggerimenti testuali contestuali e messaggi di errore comprensibili, in grado di indicare con precisione la causa del problema e le azioni necessarie per risolverlo, in conformità ai requisiti non funzionali NFR1, NFR2 e NFR6.
La rappresentazione dello stato delle camere deve inoltre essere immediata e visiva, così da ridurre i tempi decisionali e aumentare l'efficienza operativa (NFR3).
Il manager deve avere anch'esso una interfaccia che gli permetta di avere una veduta generale dello stato attuale della struttura (dipendenti e conto economico) in modo intuitivo.
La governante deve poter aggiornare lo stato di una camera con poche azioni in modo da comunicare velocemente e efficacemente lo stato della camera.

Performance vs Usability

L'interfaccia è progettata ponendo al centro la rapidità di esecuzione delle operazioni essenziali (quali check-in, consultazione dello stato delle camere e operazioni di front desk), riducendo al minimo il numero di interazioni necessarie.
A tal fine, il design privilegia l'efficienza operativa rispetto all'estetica grafica: l'interfaccia rinuncia volutamente a soluzioni visive elaborate per favorire chiarezza, immediatezza e tempi di risposta ridotti, garantendo all'operatore un utilizzo rapido e privo di distrazioni.

Definitions, acronyms, and abbreviations

Hotel Management System (HMS): Software progettato per gestire le operazioni quotidiane di un albergo, inclusi check-in, check-out, gestione camere, prenotazioni, servizi extra, personale e ristorazione.
Front Desk: Il cuore operativo dell'hotel dove i receptionist gestiscono le prenotazioni, i clienti e i servizi.
Housekeeping: Attività di pulizia, manutenzione e aggiornamento dello stato delle camere, svolta dalla governante.
Manager: Figura responsabile della supervisione delle operazioni, gestione del personale e del quadro economico della struttura.
Check-in: Procedura di registrazione del cliente all'arrivo in hotel e assegnazione della camera.
Check-out: Procedura di conclusione del soggiorno del cliente e pagamento dei servizi erogati dalla struttura.

Prenotazione: Registrazione di un soggiorno e/o di servizi aggiuntivi richiesti da un cliente.
Servizi Extra: Servizi aggiuntivi usufruiti dal cliente, come ristorazione in camera, spa o altri servizi a pagamento.
Ordine: Procedura mediante la quale il maître invia al servizio di cucina la comanda con i piatti da preparare, indicando quantità e tipologia. In questo momento la comanda diventa un effettivo ordine
Comanda: Registrazione sul sistema della richiesta del servizio ristorativo del cliente

Acronyms e abbreviations

- **HMS:** Hotel Management System
- **FR:** Functional Requirement (Requisito Funzionale)
- **NFR:** Non-Functional Requirement (Requisito Non Funzionale)
- **UI:** User Interface (Interfaccia Utente)
- **DB:** Database
- **SQL:** Structured Query Language

1.4 References

Il presente documento fa riferimento al Requirement Analysis Document (RAD), dal quale sono state tratte le specifiche sui requisiti funzionali, sui requisiti non funzionali e sui casi d'uso del sistema.

Current software application

Guardando i gestionali alberghieri attualmente sul mercato, appare chiaro che le soluzioni più usate e apprezzate integrano modularità, sicurezza e opzioni di deployment flessibili. La scelta di sviluppare un sistema distribuito in java mira a realizzare in maniera concreta le best pratice osservate nei sistemi gestionali alberghieri attualmente disponibili sul mercato.

HOASYS (GP Dati / Zucchetti)

- È un PMS (Property Management System) che può essere **in cloud** oppure in locale
- Ha modulo di ristorazione: nella loro documentazione è indicato che è possibile inviare "in tempo reale le comande raccolte ai vari reparti

produttivi” tramite tablet wireless.
- Uso di database relazionale

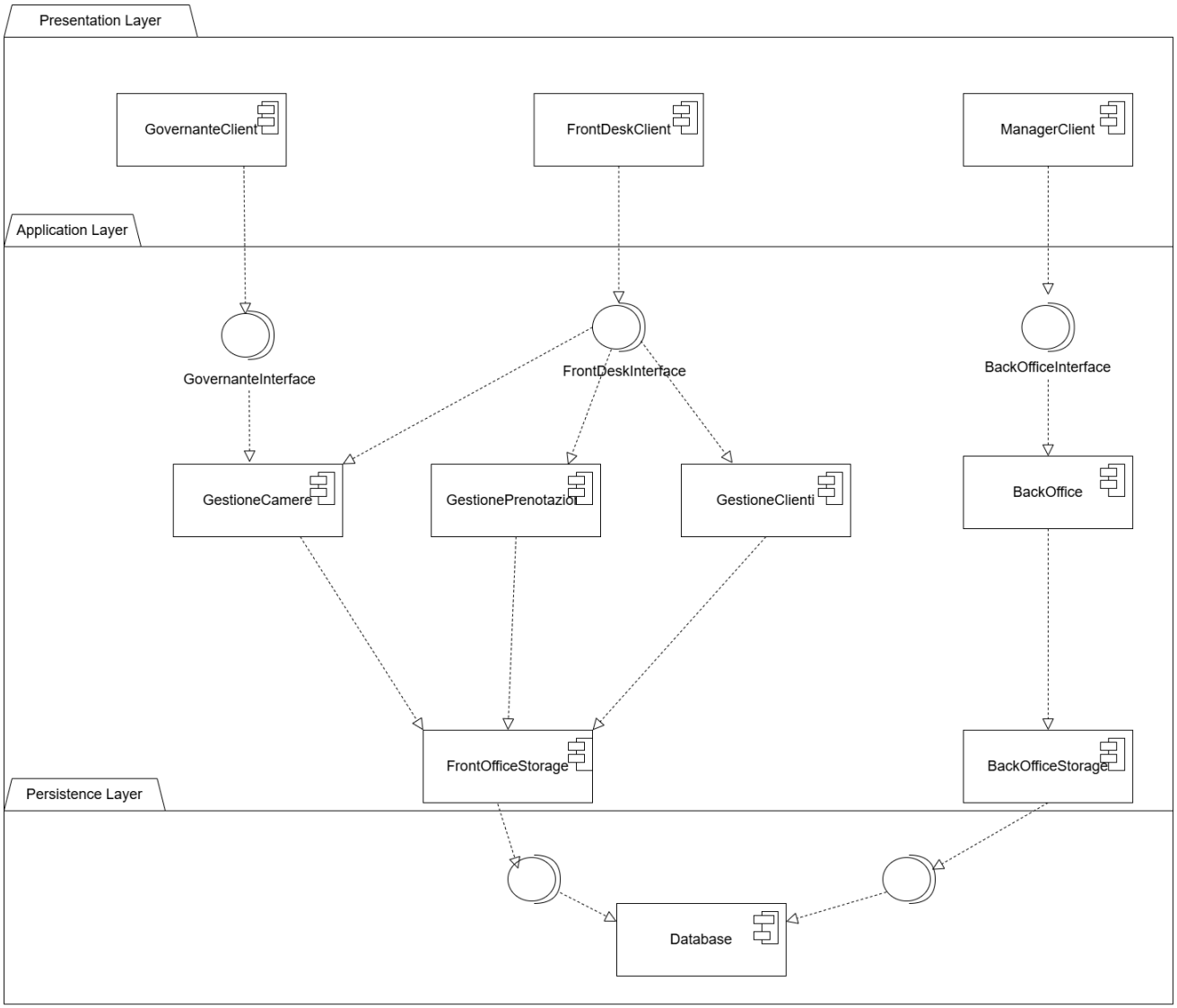
Infinity Scrigno PMS (Zucchetti)
- Suite cloud per hotel indipendenti, catene, resort, con moduli per front-office, housekeeping, F&B, vendite, amministrazione, ecc.
- Usa un **database centralizzato** a cui accedono tutti i moduli applicativi.
- Architettura modulare e scalabile: moduli separati per i diversi reparti (front office, economato, ristorazione, etc.)

Hotelkit
- È cloud-native, progettato per la cooperazione in tempo reale tra reparti, con un modello distribuito per la comunicazione tra il front desk e il personale di servizio (governanti, manutentori).

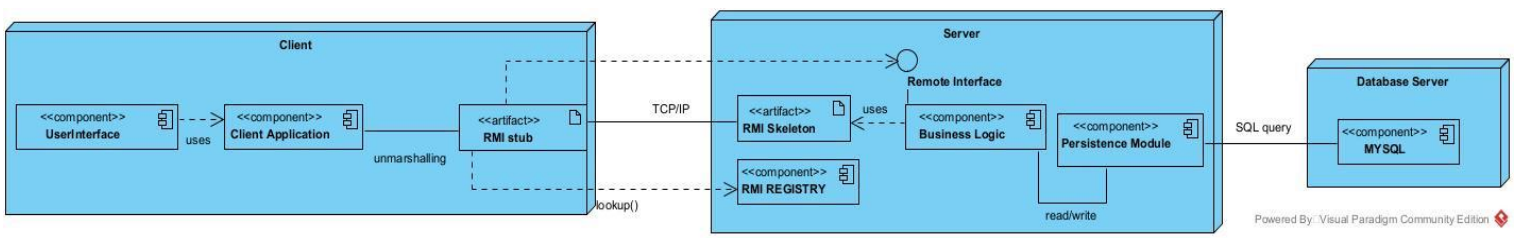
Proposed software architecture

Hardware software mapping

Component Diagram



Deployment Diagram



Persistent Data management

La gestione della persistenza viene implementata secondo un'architettura, in cui il Data Access Layer funge da intermediario tra la logica applicativa e il database relazionale.
DATABASE SCHEMA ALLEGATO AL DOCUMENTO

Abbiamo scelto di memorizzare le seguenti azioni tramite dei log file, questo per evitare di appesantire inefficientemente il database:

- Operatore Front Desk registra prenotazione
- Operatore Front Desk aggiunge servizio alla prenotazione
- Operatore Front Desk registra cliente
- Operatore Front Desk modifica prenotazione
- Manager inserisce un nuovo impiegato
- Manager rimuove un nuovo impiegato

Access Control and security

Attori \ Oggetti	Utente	ServizioAutenticazione	Impiegato	CatalogoImpiegati	Cliente	CatalogoClienti	Camera	CatalogoCamere	Prenotazione	CatalogoPrenotazioni	Servizio	RicevutaFiscale	Trattamento
Operatore front-desk		autenticazione cambioPassword			<<getter>> <<setter>>	registraCliente aggiornaDatiCliente cercaClienti ban / unBan getListaBannati			checkout aggiungiCliente aggiungiServizio rimuoviCliente rimuoviServizio	registraPrenotazione aggiornaDatiPrenotazione cercaPrenotazioni getStoricoPrenotazioni eliminaPrenotazione effettuaCheckOut	<<getter>> <<setter>>	<<getter>> <<setter>>	<<getter>> <<setter>>
Governante		autenticazione cambioPassword					<<getter>> setStato	setStatoOutOfOrder setStatoInPulizia getCamera					
Manager	setNewHashPassword	autenticazione cambioPassword	<<getter>> <<setter>>	registraImpiegato aggiornaDatiImpiegato eliminaImpiegato CercaImpiegati generaPasswordTemp									

Boundary conditions

Il nostro sistema software per gestire le problematiche relative alla manutenzione prevede una figura amministrativa, a cui viene dedicata una sezione per la gestione amministrativa delle problematiche.
L'amministratore dovrà effettuare le operazioni di avvio del software di shutdown tramite una interfaccia da terminale dedicata.

UC1: Startup del sistema

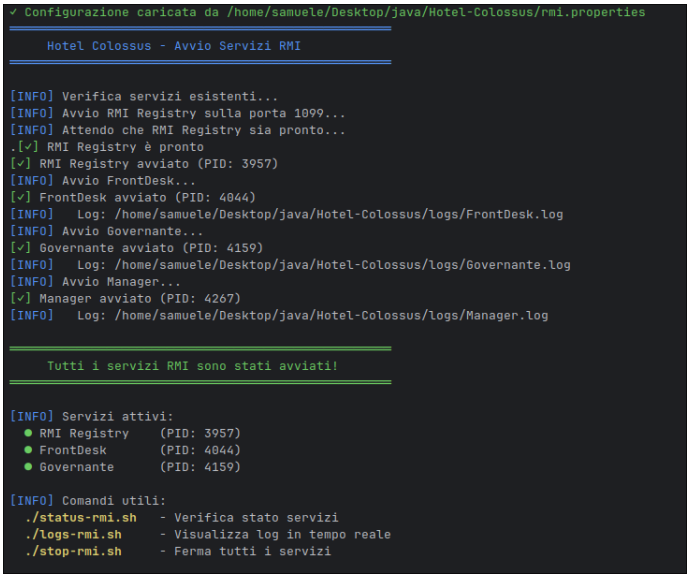
Attore: Admin

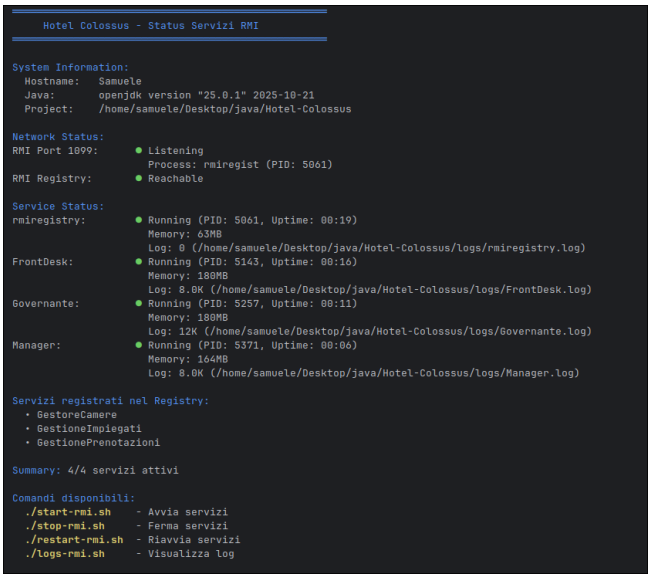
Entry Condition: l'operatore sta per scegliere il comando da digitare

Flusso di eventi:

1. Clicca il pulsante "Start system"
2. Il sistema fa partire uno script che fa avviare l'RMI registry

Exit condition: Il sistema è avviato e gli altri operatori possono usare il sistema





UC2: Spegnimento del sistema

Attore: Admin

Entry Condition: l'operatore sta per scegliere il comando da digitare

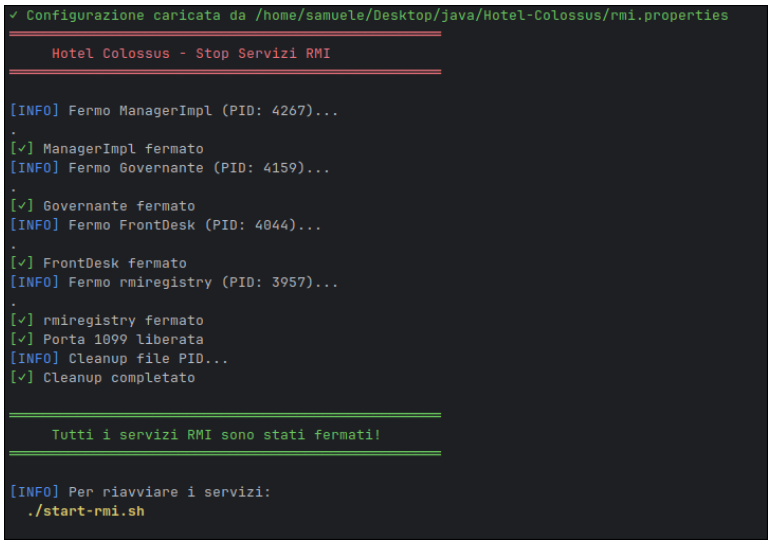
Flusso di eventi:

1. L'amministratore clicca il pulsante "Gestione del

Software" e gli viene mostrata una schermata.

2. L'amministratore clicca il pulsante "shutdown System"

Exit condition: il sistema riceve il segnale di spegnimento del software e spegne il software



UC3: Configurazione del software

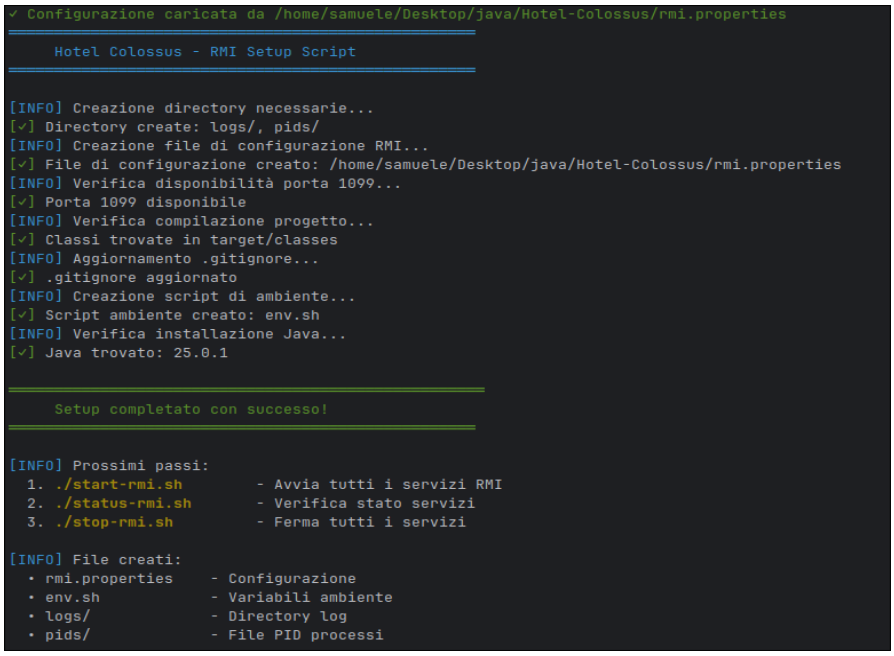
Attore: Admin

Entry Condition: l'admin deve stare nella sua schermata home

Flusso di eventi:

1. L'amministratore clicca il pulsante "Gestione del Software" e gli viene mostrata una schermata.
2. L'amministratore clicca il pulsante "shutdown System"

Exit condition: il sistema riceve il segnale di spegnimento del software e spegne il software



UC4: Avviso di manutenzione al sistema

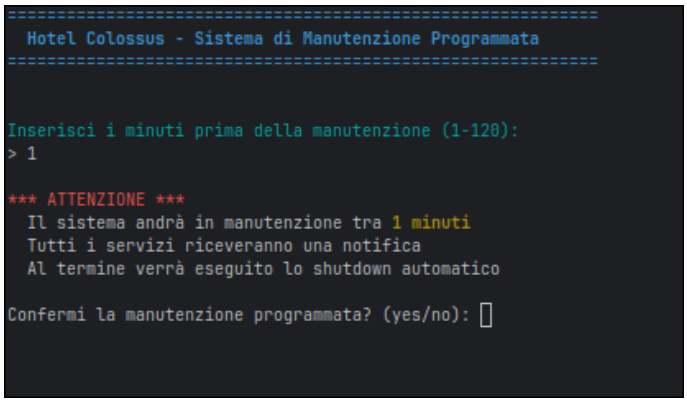
Attore: Admin

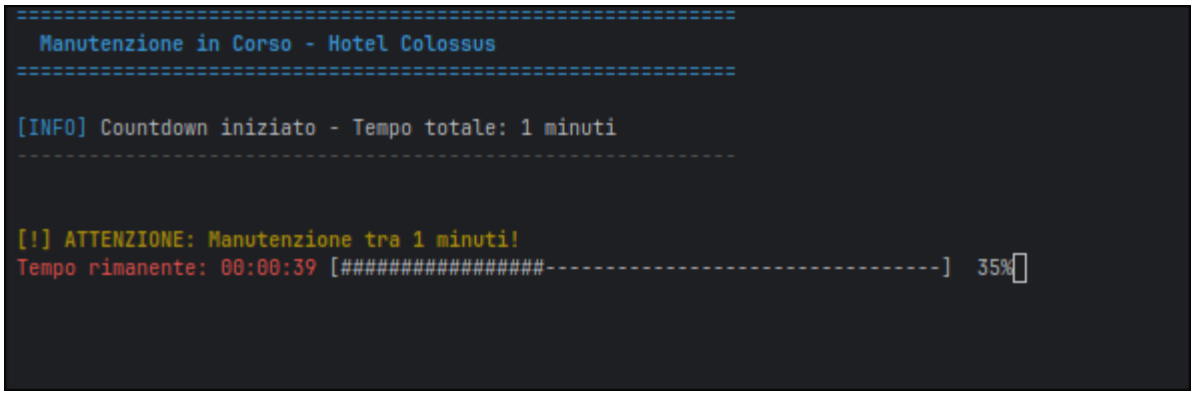
Entry Condition: l'operatore sta per scegliere il comando da digitare

Flusso di eventi:

1. L'amministratore clicca il pulsante "Comunicazione" e gli compare una piccola schermata.
2. L'amministratore scrive il messaggio (ad esempio: "in data 27 novembre alle ore 15:30 fino il 28 novembre alle 15:30 sarà effettuata una manutenzione").
3. successivamente inserisce nel sistema la data e l'ora
4. L'amministratore conferma l'operazione

Exit condition: il sistema ha memorizzato l'avviso





UC5: Visualizzazione log di sistema

Per aiutare l'amministratore a effettuare le sue mansioni ordinarie sarà messo a sua disposizione un sistema di log dinamico, utile a capire le condizioni del sistema

Attore: Admin

Entry Condition: l'operatore sta per scegliere il comando da digitare

Flusso di eventi:

1. L'amministratore digita il comando /logs-rmi.
2. Il sistema mostra un menù testuale in cui può scegliere quale log desidera visualizzare
3. L'amministratore sceglie il log specifico da visualizzare

Exit condition: l'amministratore visualizza il resoconto del log

```
============================================================
     Hotel Colossus - Visualizzazione Log
============================================================

Seleziona il servizio di cui visualizzare i log:

  1) RMI Registry
  2) FrontDesk Server
  3) Governante Server
  4) Manager Server
  5) Maintenance (manutenzione)

Visualizzazione multipla:
  6) Tutti i servizi (interleaved)

Filtri e ricerca:
  7) Cerca nel log (grep)
  8) Solo ERRORI
  9) Solo WARNING
  10) Solo INFO

Utilità:
  11) Statistiche log
  12) Pulisci log
  13) Esporta log
  14) Mostra file log disponibili

  0) Esci

Scelta:
```

```
============================================================
     Hotel Colossus - Visualizzazione Log
============================================================

▶ Visualizzazione tutti i log (interleaved)
  (Premi Ctrl+C per tornare al menu)

[REGISTRY] [19:13:35] ==> /home/samuele/Desktop/java/Hotel-Colossus/logs/rmiregistry.log <==
[19:13:35]
[FRONTDESK] [19:13:35] ==> /home/samuele/Desktop/java/Hotel-Colossus/logs/FrontDesk.log <==
[19:13:35] [INFO] --- exec:3.6.3:java (default-cli) @ HotelColossus ---
[FRONTDESK] [19:13:35] 19:07:56 INFO  FrontDesk: Avvio RMI Registry sulla porta 1099...
[FRONTDESK] [19:13:35] 19:07:56 INFO  FrontDesk: ✓ Connesso a RMI Registry esistente
[FRONTDESK] [19:13:35] 19:07:56 INFO  FrontDesk: Genero il gestore prenotazioni...
[19:13:35] 5
[FRONTDESK] [19:13:35] 19:07:57 INFO  FrontDesk: ✓ Gestore prenotazioni creato
[FRONTDESK] [19:13:35] 19:07:57 INFO  FrontDesk: Effettuo il rebind di gestione prenotazioni...
[FRONTDESK] [19:13:35] 19:07:57 INFO  FrontDesk: ✓ Gestore prenotazioni registrato con successo!
[FRONTDESK] [19:13:35] 19:07:57 INFO  FrontDesk: ✓ Servizio 'GestionePrenotazioni' pronto
[FRONTDESK] [19:13:35] 19:07:57 INFO  FrontDesk: Server in attesa di connessioni...
[19:13:35]
[GOVERNANTE] [19:13:35] ==> /home/samuele/Desktop/java/Hotel-Colossus/logs/Governante.log <==
[GOVERNANTE] [19:13:35] Feb 05, 2026 7:08:01 PM it.unisa.Server.gestioneCamere.Governante main
[19:13:35] INFO: Sto creando il gestore camere...
[GOVERNANTE] [19:13:35] Feb 05, 2026 7:08:01 PM it.unisa.Server.gestioneCamere.Governante main
[19:13:35] INFO: Gestore camere creato...
[GOVERNANTE] [19:13:35] Feb 05, 2026 7:08:01 PM it.unisa.Server.gestioneCamere.Governante main
[19:13:35] INFO: Effettuo il rebind del gestore camere...
[GOVERNANTE] [19:13:35] Feb 05, 2026 7:08:01 PM it.unisa.Server.gestioneCamere.Governante main
[19:13:35] INFO: Il gestore camere è pronto e registrato!
[GOVERNANTE] [19:13:35] Feb 05, 2026 7:08:01 PM it.unisa.Server.gestioneCamere.Governante main
[19:13:35] INFO: Server in attesa di connessioni...
[19:13:35]
[MAINTENANCE] [19:13:35] ==> /home/samuele/Desktop/java/Hotel-Colossus/logs/maintenance.notify <==
```

Subsystem Servicies

GESTIONE CAMERE	
Nome servizio	**Servizi**
Modifica stato	setStatoOutOfOrder() setStatoInPulizia()
Recupera camera	getCamera()

GESTIONE PRENOTAZIONE	
Nome servizio	**Servizi**
Creazione prenotazione	registraPrenotazione(datiPren.)
Modifica prenotazione	aggiornaDatiPrenotazione(prenotazione) aggingiServizio(servizio) rimuoviServizio(servizio) aggingiCliente(cliente) rimuoviCliente(cliente)
Rimozione prenotazione	eliminaPrenotazione(prenotazione)
Recupero prenotazioni	getListaPrenotazioni(datiPren.)
Checkout prenotazione	effettuaCheckoutPrenotazione(datiPren.)

GESTIONE CLIENTI	
Nome servizio	**Servizi**
Registrazione cliente	registraCliente(datiCliente)
Modifica cliente	aggiornaDatiCliente(cliente)
Rimozione cliente	eliminaCliente(cliente)
Recupero clienti	getListaClienti(nome, cognome, nazionalità, dataNascita, Sesso)
Moderazione Clienti	ban(cliente) unBan(cliente) getListaBannati()

BACKOFFICE	
Nome servizio	**Servizi**
Registrazione impiegato	registraImpiegato(datiImpiegato)
Modifica impiegato	aggiornaDatiImpiegato(impiegato)
Rimozione impiegato	eliminaImpiegato (impiegato)

Recupero impiegato	getListaImpiegato (nome, cognome, sesso, ruolo)
Gestione accesso impiegati	generaPasswordTemporanea(impiegato)

AUTENTICAZIONE	
Nome servizio	**Servizi**
Autenticazione	autenticazione (username, password)
Recupero password	cambioPassword(newPassword, confirmPassword)