

Cruise for single Management System

Matteo Amato

Contents

1	Specifica dei Requisiti Software	2
1.1	Introduzione	2
1.1.1	Scopo del Documento	2
1.1.2	Panoramica del Sistema	2
1.1.3	Requisiti Hardware e Software	2
1.1.4	Analisi di Piattaforme Simili	2
1.2	User Stories	3
1.3	Functional Requirements	3
1.4	Use Case Diagram	3
1.5	Internal Steps	5
1.5.1	Prenota una stanza	5
2	Storyboard	5
3	Class Diagram	6
3.1	BCE-VOPC	6
3.2	Vopc-MVC	8
3.3	Design level diagram	10
3.4	Design pattern	12
3.5	Activity diagram	13
3.6	Sequence diagram	14
3.7	State diagram	15
4	Testing	16
5	Gestione della persistenza	16
5.1	File System	16
5.2	Database	17
6	Link	17

1 Specifica dei Requisiti Software

1.1 Introduzione

1.1.1 Scopo del Documento

Questo documento specifica i requisiti per il **Cruise for single Management System**, un'applicazione Java che supporta le operazioni quotidiane di un sistema per le prenotazioni per una crociera, inclusa l'amministrazione del personale.

1.1.2 Panoramica del Sistema

Il Cruise for single Management System è distribuito come JAR eseguibile, realizzato con Oracle JDK SE 24. Esso espone sei funzionalità principali:

UC1 **Prenotare una Stanza** – l'utente può prenotare una stanza per la destinazione che preferisce.

UC2 **Recensione** – l'utente può lasciare una recensione del servizio.

UC3 **Gestire Prenotazioni** – lo staff può approvare o rifiutare le prenotazioni dei clienti.

UC4 **Gestire stanze** – lo staff può creare, modificare e archiviare le stanze.

UC5 **Gestire Attività e Servizi** – lo staff può creare, modificare e archiviare attività e servizi.

UC6 **Gestire Destinazioni** – lo staff può creare, modificare e archiviare le destinazioni possibili.

1.1.3 Requisiti Hardware e Software

Hardware

- **Processore:** CPU moderna in grado di eseguire bytecode Java.
- **Memoria:** ≥ 4 GB di RAM.
- **Archiviazione:** almeno 500 MB di spazio libero su disco per i file dell'applicazione e i dati offline.

Software

- **Oracle JDK SE 24** — runtime richiesto.
- **MySQL Server 8.x** — archiviazione persistente per stanze, utenti e prenotazioni.
- **IDE di sviluppo** (opzionale) — Eclipse 2025-06 o IntelliJ IDEA 2025.1.

1.1.4 Analisi di Piattaforme Simili

Esistono già diverse piattaforme nei settori delle prenotazioni per crociera, come **Booking.com** e **crocierissime.it**. Tuttavia presentano alcune limitazioni:

- **Booking.com:** offre un'ampia gamma di strutture, ma l'interfaccia utente può risultare eccessivamente complessa per alcuni utenti.
- **crocierissime.it:** consente di prenotare una camera tramite un catalogo, ma la sua interfaccia potrebbe non essere fruibile su tutti i dispositivi.

1.2 User Stories

US-1

Come utente, voglio prenotare una stanza sulla nave per data e itinerario specifici, per poter organizzare al meglio la mia vacanza.

US-2

Come utente, voglio ricevere una e-mail di conferma con i dettagli della mia prenotazione, in modo da avere certezza che la mia prenotazione sia andata a buon fine.

US-3

Come membro dello staff, voglio gestire le prenotazioni dei clienti, così da poter rifiutare quelle non conformi.

1.3 Functional Requirements

FR-1

Il sistema deve inviare un'email di conferma al cliente che ha prenotato una stanza.

FR-2

Il sistema deve richiedere che l'utente abbia effettuato con successo l'accesso prima di poter creare una prenotazione per una stanza.

FR-3

Il sistema deve consentire ai membri dello staff di visualizzare tutte le prenotazioni e consultarne i dettagli.

1.4 Use Case Diagram

Il diagramma rappresenta le funzionalità principali del Cruise for single Management System.

1.5 Internal Steps

1.5.1 Prenota una stanza

1. L'utente effettua il login.
2. Il sistema mostra la home page con i bottoni per la navigazione.
3. L'utente seleziona *Prenota una Stanza*.
4. Il sistema visualizza il form di prenotazione.
5. L'utente inserisce data di partenza, itinerario, tipologia di stanza e numero di ospiti.
6. Il sistema valida i dati inseriti e verifica la disponibilità della stanza per la data indicata.
7. Il sistema mostra un riepilogo dei dati inseriti.
8. L'utente conferma.
9. Il sistema memorizza la prenotazione nel database.

Estensioni

6a. Nessuna stanza disponibile per la data richiesta:

il sistema avvisa l'utente che la stanza non è disponibile per la data richiesta; torna al Passo 5.

6b. Validazione dati fallita:

il sistema rileva che uno o più campi contengono errori (es. data in formato errato, campo obbligatorio vuoto), visualizza un messaggio d'errore e invita l'utente a correggere i dati; torna al Passo 5.

7a. Annulla prenotazione:

l'utente annulla la prenotazione; torna al Passo 2.

8a. Errore di salvataggio su DB:

il sistema riscontra un problema nel salvataggio della prenotazione, notifica l'utente e torna al Passo 2.

2 Storyboard

Questa sezione mostra le principali viste dell'utente.

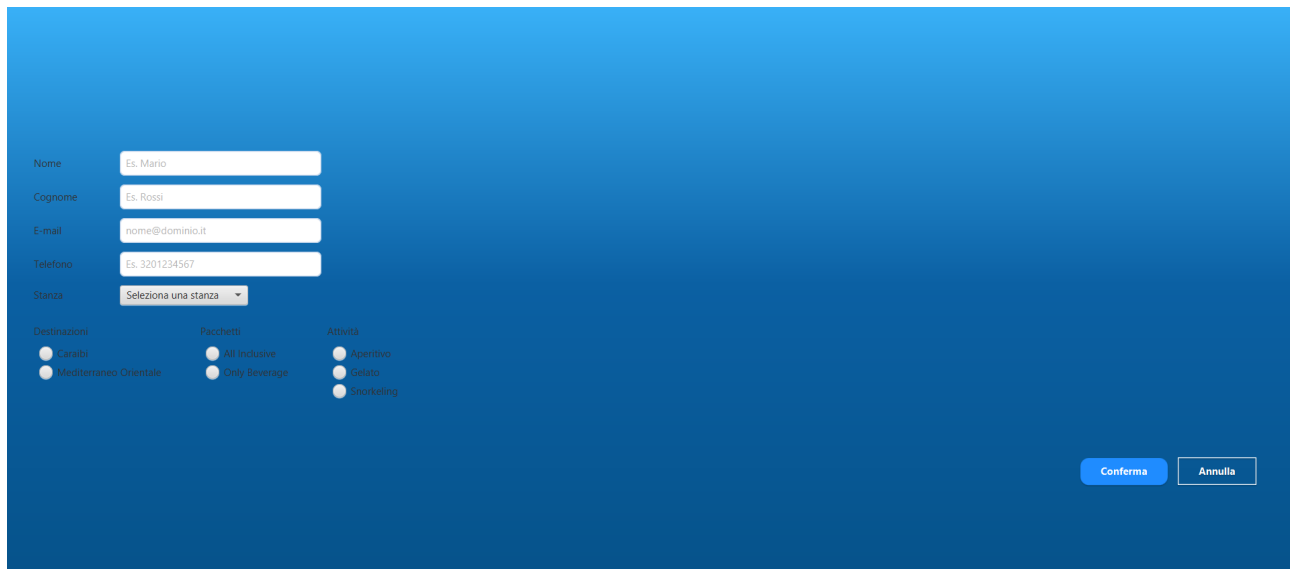


Figure 2: vista Prenota una stanza.

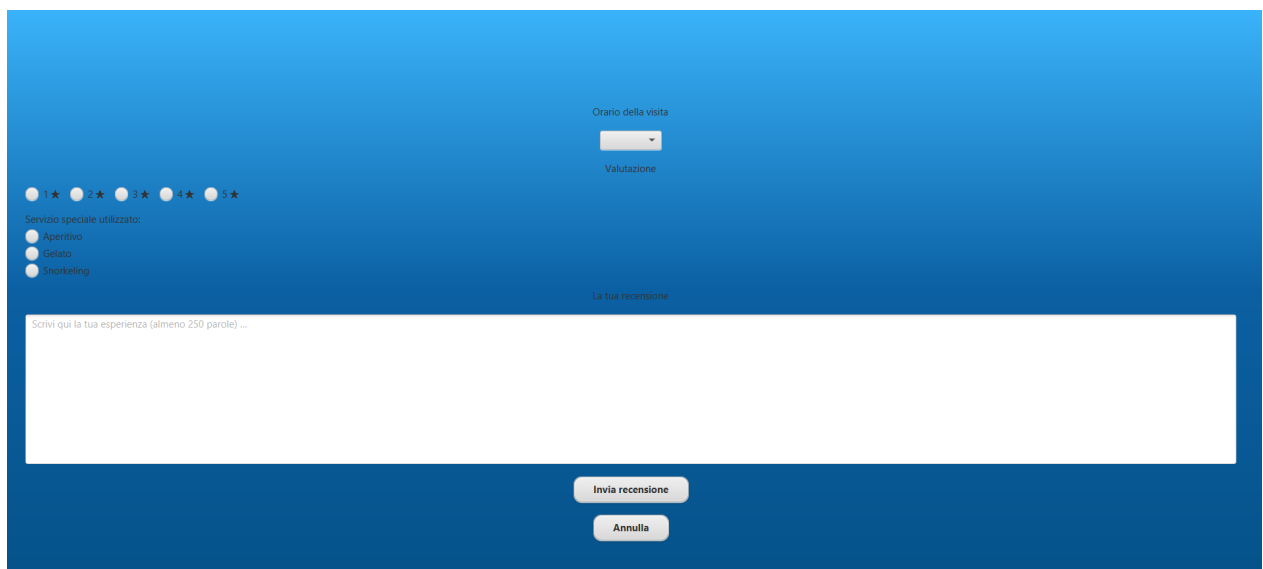


Figure 3: vista recensione

3 Class Diagram

3.1 BCE-VOPC

Il diagramma delle classi del *Cruise for single Management System* è organizzato in un insieme di pacchetti ben definiti:

Entity

Contiene le classi di dominio — ad es. `User` e `Booking`, `Itinerary`. Questi oggetti incapsulano lo stato ed espongono i metodi getter/setter standard.

Control

`BookingService` coordina il workflow di prenotazione della stanza.

Boundary

Le classi *Boundary* mediano tra il sistema e l'ambiente esterno.

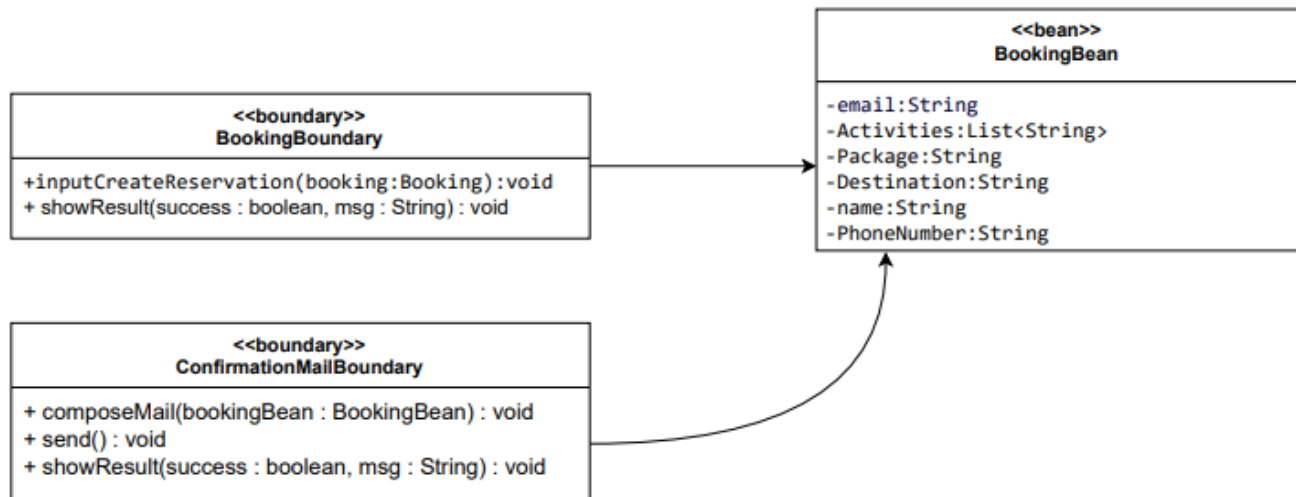


Figure 4: Prenota una stanza BCE Boundary e beans

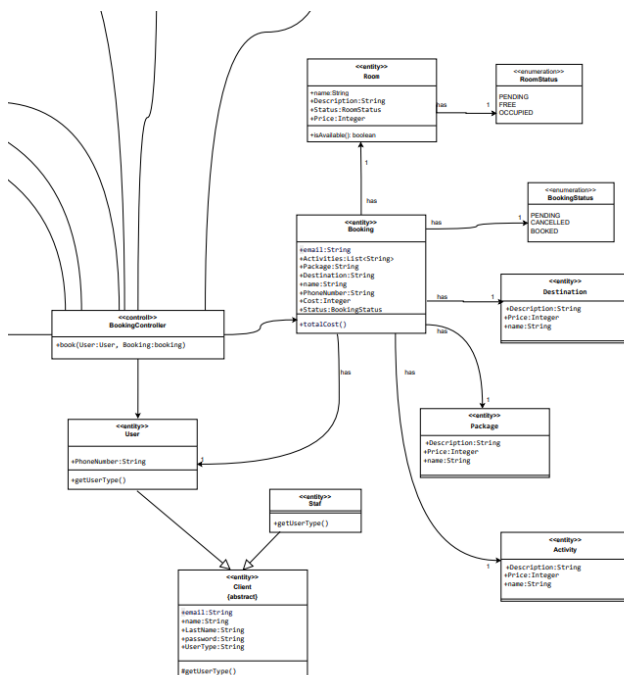


Figure 5: Prenota una stanza BCE Controller e entity

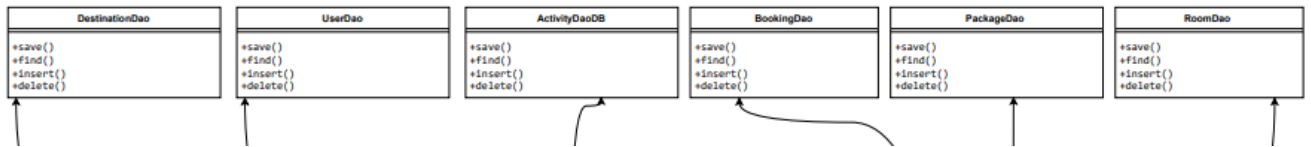


Figure 6: Prenota una stanza BCE DAO

3.2 Vopc-MVC

Entity

Contiene le classi di dominio — ad es. `User`, `Booking`, `Room`, `Itinerary`. Questi oggetti incapsulano lo stato ed espongono i metodi getter/setter standard.

Controller — Applicativo & Grafico

La classe `BookingService` coordina il workflow di prenotazione della stanza. Il livello di *controllerGrafico* gestisce gli eventi dell'utente e aggiorna di conseguenza le viste JavaFX.

View

Ospita le schermate JavaFX che costituiscono l'interfaccia grafica (ad es. `BookingView`).

Le classi rispettano il pattern Model–View–Controller (MVC), con i modell collocati nel package `Entity`.

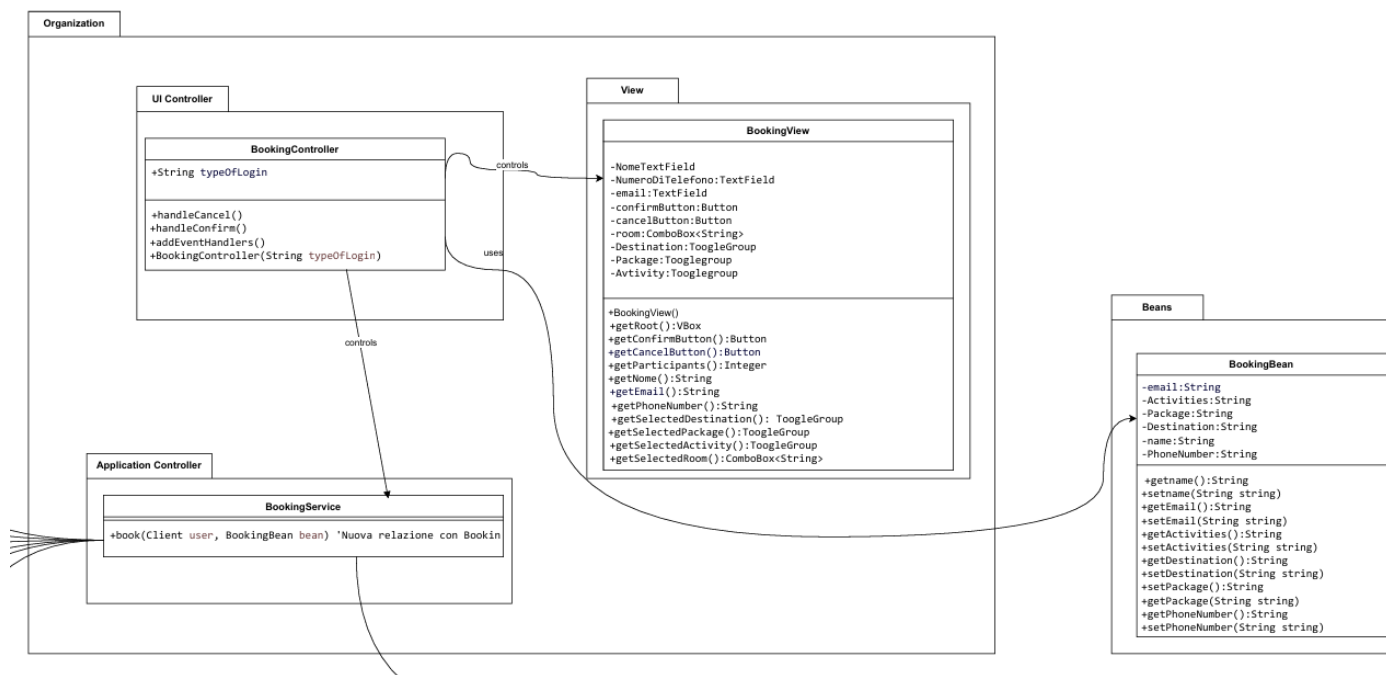


Figure 7: Prenota una stanza-Controller View e beans

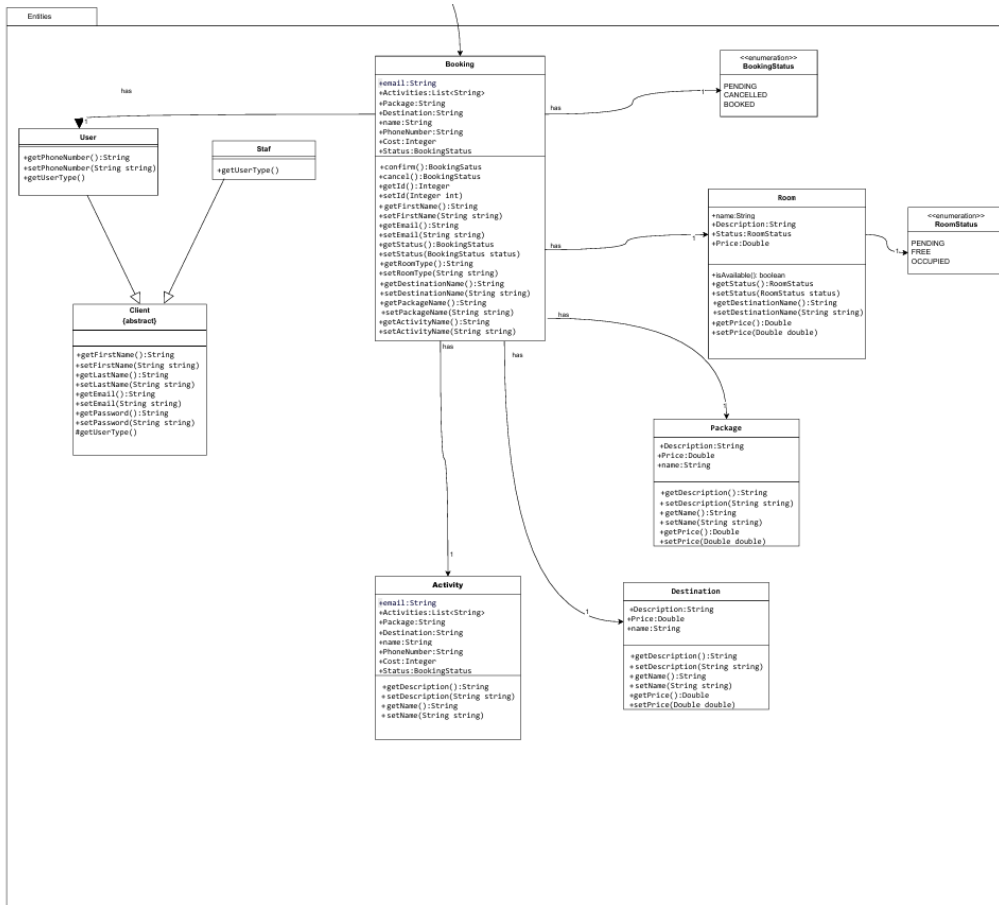


Figure 8: Prenota una stanza-Entity

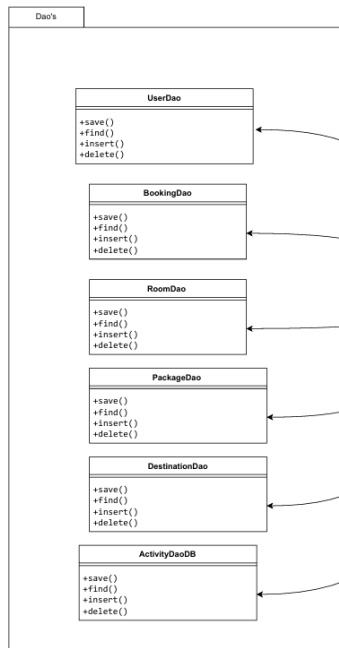


Figure 9: Prenota una stanza DAO

3.3 Design level diagram

Le classi sono descritte in dettaglio insieme alle rispettive relazioni e responsabilità. Come nel VOPC, le classi sono organizzate secondo il pattern MVC

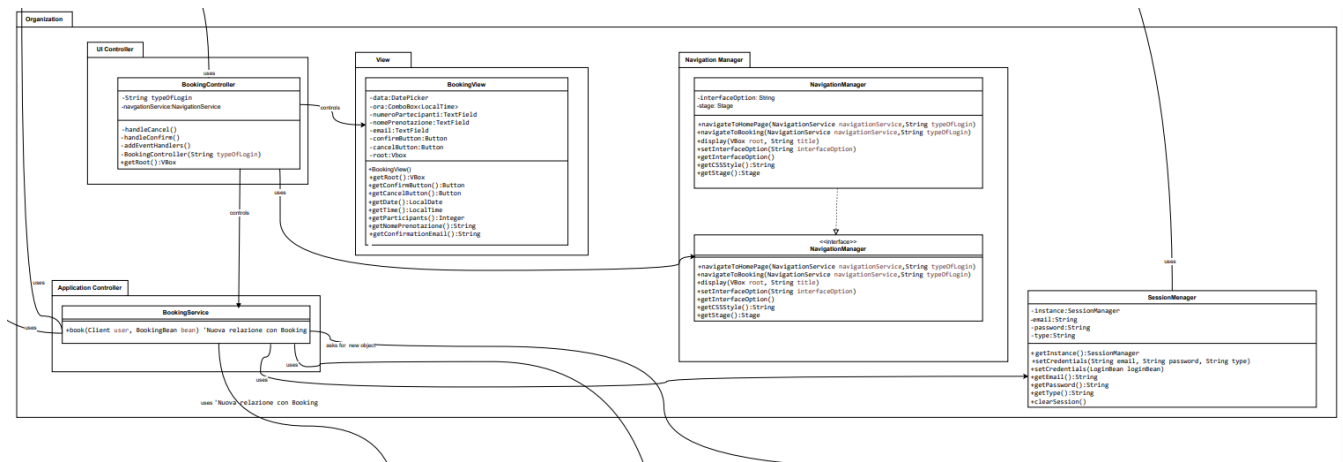


Figure 10: Prenota una stanza-Controller e View

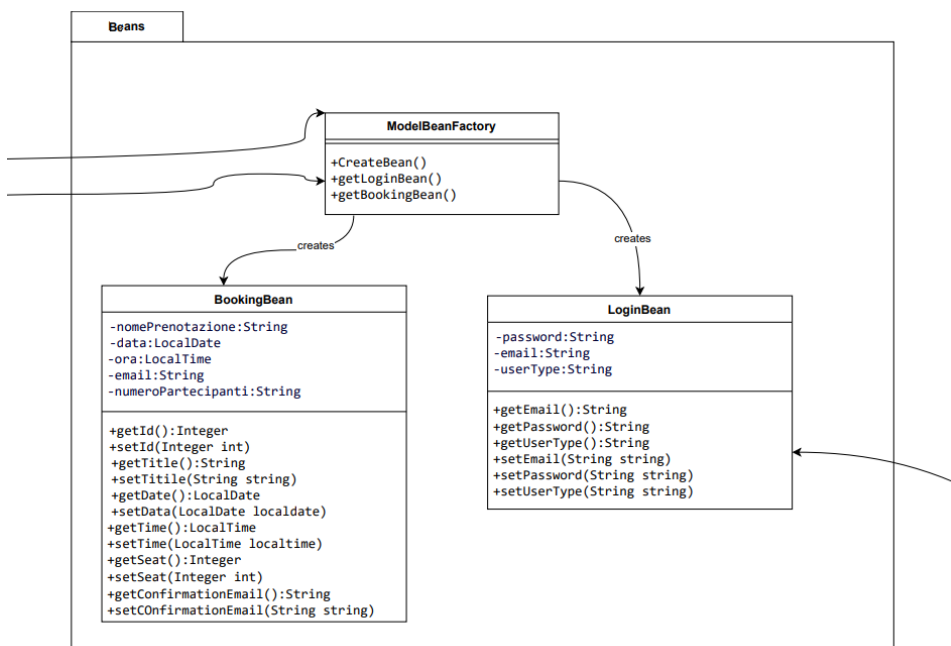


Figure 11: Prenota una stanza-Beans

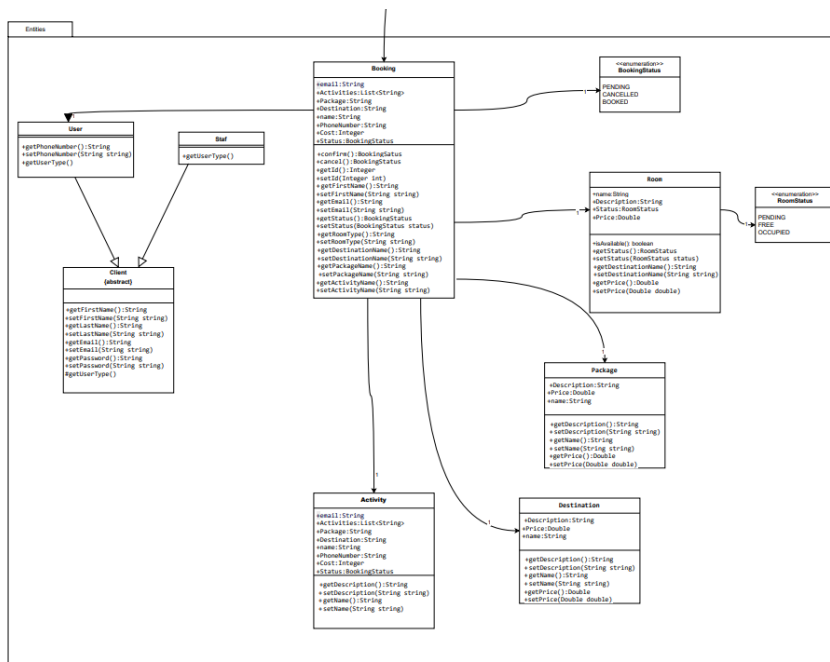


Figure 12: Prenota una stanza-Entity

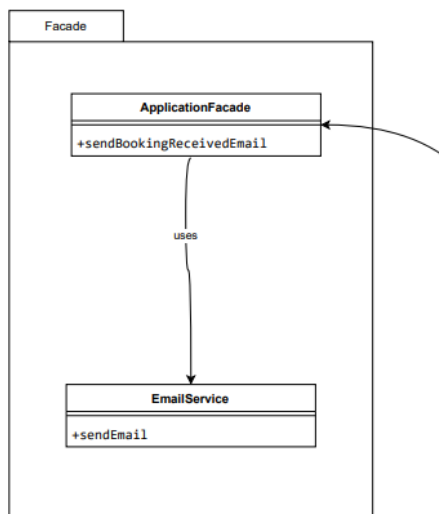


Figure 13: Prenota una stanza-*Pattern Facade*

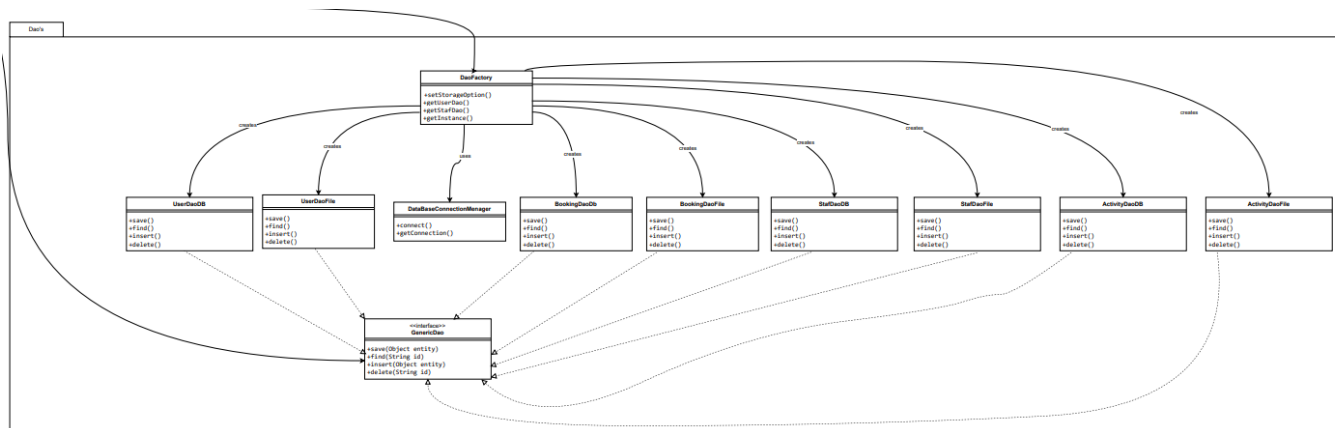


Figure 14: Prenota una stanza-Daos- Pattern Factory e Singleton

3.4 Design pattern

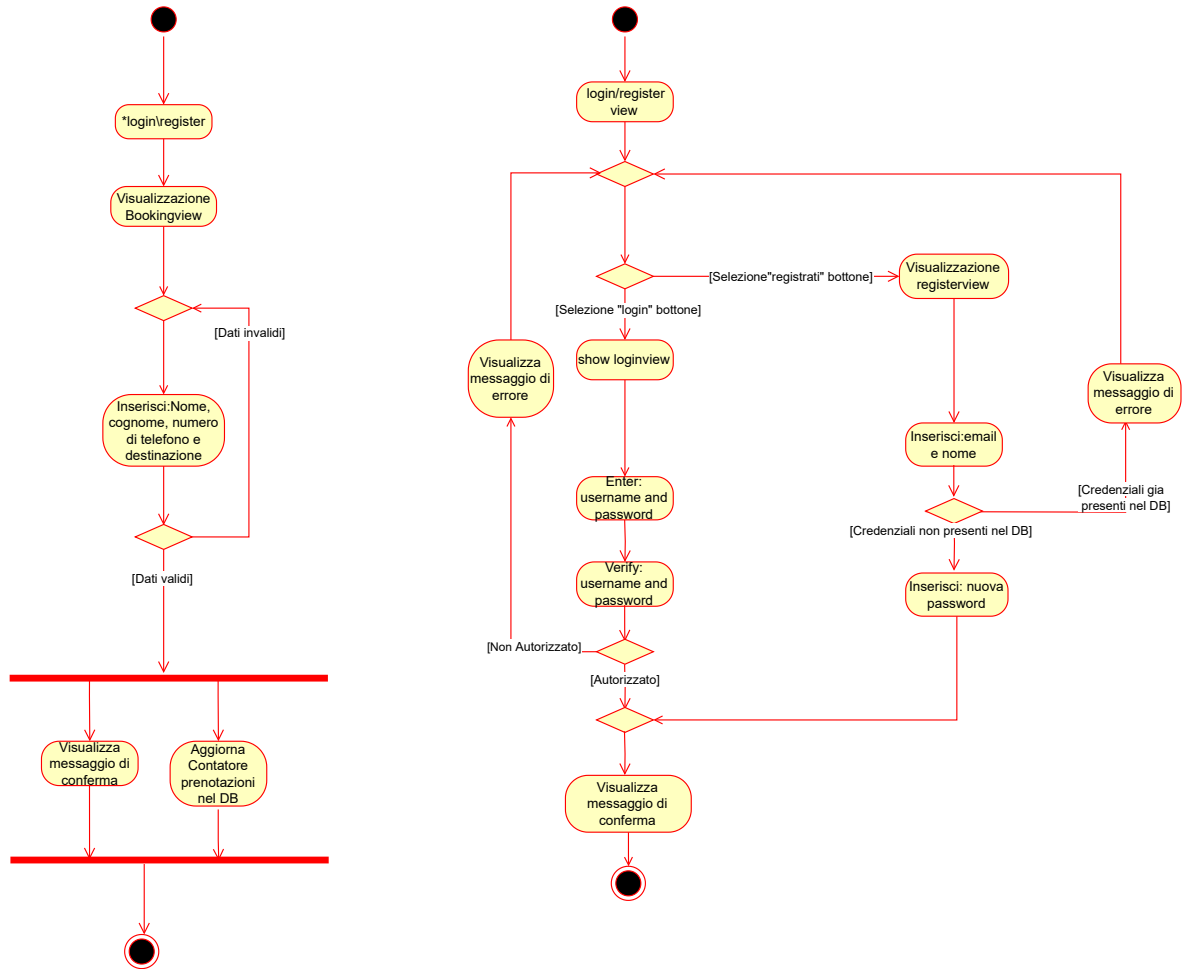
Ho adottato il pattern *Singleton* nella classe **SessionManager** per garantire che la sessione (e quindi l'istanza dell'utente che interagisce con il sistema) sia unica durante l'utilizzo.

In secondo luogo, ho applicato un *pattern Factory* per la creazione dei DAO utilizzati da **BookingService**, in modo che il DAO corretto possa essere scelto in base alla configurazione del sistema (Database, File System o Stateless). In questo modo preserviamo una separazione delle responsabilità in linea con il pattern MVC.

Infine, la classe **ApplicationFacade** implementa il pattern *Facade*, consentendo a ogni classe di accedere ai servizi implementati da altre classi, indipendentemente da quale classe li fornisca. **ApplicationFacade** si impegna a fornire questa funzionalità senza dover conoscere il funzionamento interno delle altre classi coinvolte nel processo. Ciò consente alle classi che la utilizzano di interagire con più sottosistemi senza conoscerne i dettagli di implementazione.

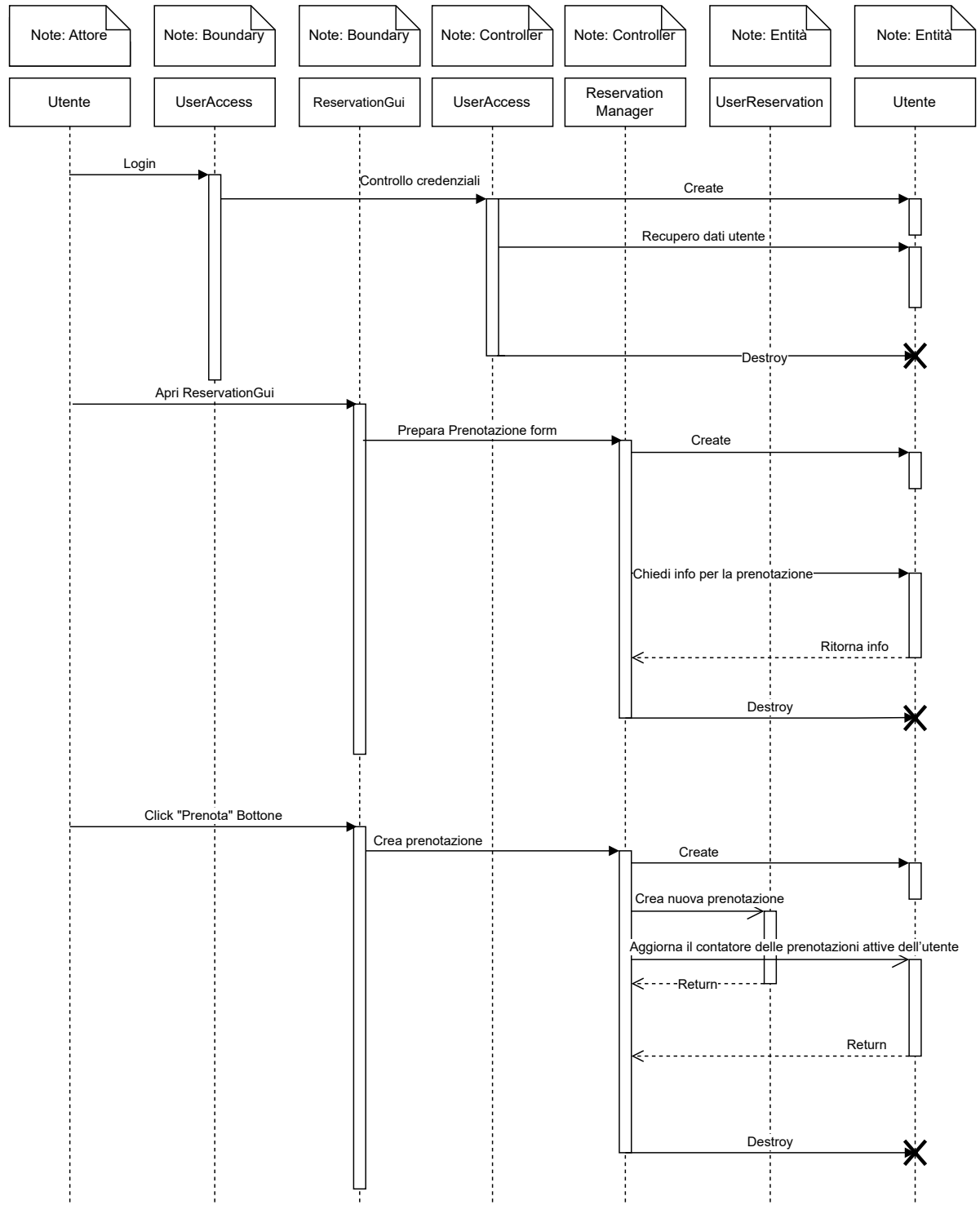
3.5 Activity diagram

USE CASE: PRENOTA UNA STANZA



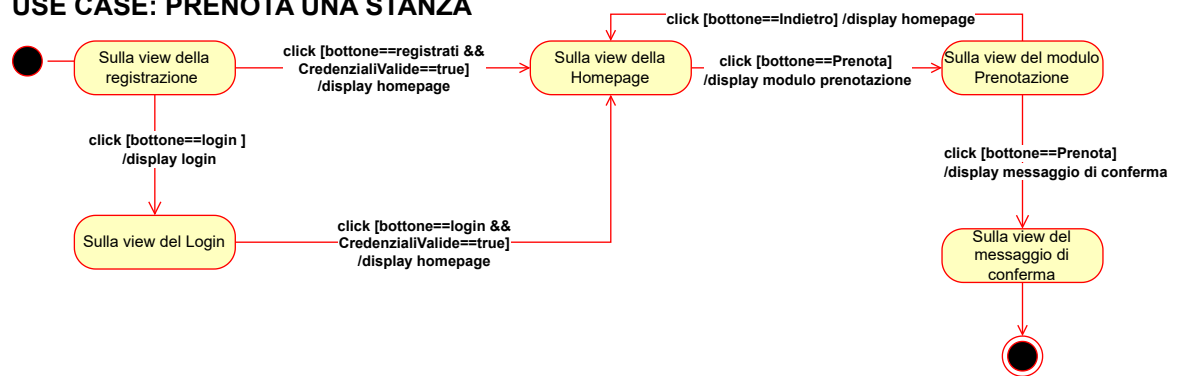
3.6 Sequence diagram

USE CASE: PRENOTA UNA STANZA



3.7 State diagram

USE CASE: PRENOTA UNA STANZA



4 Testing

Nel modulo di testing dell'applicazione sono state implementate tre classi di test principali per verificare il corretto funzionamento delle funzionalità fondamentali del sistema.

BookingServiceTest

- **testPrenotazione:** verifica che una prenotazione valida venga creata con successo e memorizzata correttamente nel database.
- **testValidazione:** verifica che il sistema rifiuti una prenotazione con dati di input non validi.

ReviewServiceTest

- **testReviewValida:** verifica che una recensione con dati corretti venga accettata e salvata correttamente.
- **testReviewNonValida:** conferma che il sistema rifiuti recensioni con dati errati e sollevi un'eccezione appropriata.
- **testErroreDatabase:** controlla che il sistema gestisca correttamente gli errori di persistenza durante il salvataggio di una recensione (es. campi vuoti).

DaoFactoryTest

- **givenStateless_whenGetUserDao:** verifica che, in base alla modalità di storage selezionata, venga restituita l'implementazione corretta del DAO per gli utenti.
- **givenStateless_whenGetStaffDao:** verifica che il DAO per i membri dello staff venga inizializzato correttamente.
- **givenStateless_whenGetBookingDao:** conferma che la gestione delle prenotazioni sia delegata alla classe DAO appropriata.
- **givenDatabase_whenGetDaos_thenUseConnectionManager:** verifica che la connessione al database venga stabilita correttamente tramite il connection manager.

5 Gestione della persistenza

Il sistema utilizza due modalità di persistenza, oltre alla memoria per la modalità demo.

5.1 File System

Il sistema si basa su un insieme di file JSON che memorizzano le informazioni sulle entità usate dall'applicazione e, nel caso delle prenotazioni — che sono aggregati di più entità — le loro associazioni. I file sono organizzati come segue:

- **connectionSettings.json:** contiene le impostazioni di connessione al server, inclusi URL, nome utente e password necessari per accedere al database remoto.
- **Room.json:** serializza i dati relativi alle stanze.
- **bookings.json:** serializza i dati delle prenotazioni effettuate.
- **staff.json:** serializza i dati relativi ai membri dello staff.

- `users.json`: serializza i dati relativi agli utenti.
- `activity.json`: serializza i dati relativi alle attività/servizi.

5.2 Database

L'altra modalità di persistenza prevede la connessione a un server MySQL per accedere alle informazioni in tempo reale. Il database MySQL memorizza, come nel caso del file system, le entità, usando almeno una tabella per ciascuna entità. In particolare:

- Le tabelle del database corrispondono alle principali entità del sistema, come cliente, stanze, prenotazioni e servizi.
- Gli aggiornamenti effettuati ai dati tramite il file system sono sincronizzati con il database MySQL, per garantire la consistenza delle informazioni.

6 Link

The following links are available for the project:

- **Source Code (GitHub):** <https://github.com/MatteoAmato18/CrocieraISPW/tree/971bb4b8ecc924ea27980fed8f218a901b97732d/src> — il codice completo del programma.
- **Class Diagrams:** <https://github.com/MatteoAmato18/CrocieraISPW/tree/971bb4b8ecc924ea27980/Deliverables> — Tutti i class diagram.
- **SonarCloud Report:** Il sistema è stato analizzato tramite SonarCloud per identificare possibili bug o vulnerabilità. Risultato disponibile al seguente link https://sonarcloud.io/summary/new_code?id=MatteoAmato18_CrocieraISPW&branch=main.
- **Presentation Video:** <https://github.com/MatteoAmato18/CrocieraISPW/tree/510ecbb3b8bc2b3963/video> — Video di presentazione del progetto.