|  |
| --- |
| **Università degli Studi di Salerno Corso di Ingegneria del Software** |

**Hotel Campus  
Object Design Document  
Versione 0.1**

Immagine che contiene schermata, Elementi grafici, design

Descrizione generata automaticamente

Data: 05/12/2024

**Coordinatore del progetto:**

|  |  |
| --- | --- |
| **Nome** | **Matricola** |
|  |  |
|  |  |

**Partecipanti:**

|  |  |
| --- | --- |
| **Nome** | **Matricola** |
| Luca Del Bue | 0512116173 |
| Salvatore Di Martino | 0512116932 |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **Scritto da:** | Team members |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autore** |
| 05/12/2024 | 0.1 | Prima stesura | Team members |
|  |  |  |  |

Indice

1. Introduzione 4

1.1. Scopo del Sistema 4

1.2. Obiettivi di progettazione 4

1.3. Linee guida per la documentazione dell’interfaccia 4

1.4. Ottimizzazione del modello a oggetti 4

2. Packages 6

2.1. Struttura del progetto 6

3. Interfaccia delle classi 9

4. Design Pattern 11

1. Introduzione

## Scopo del Sistema

Lo scopo del sistema è permettere la gestione dell’attività di una struttura alberghiera. Si intende

gestire le camere, i servizi offerti e le prenotazioni effettuate dai clienti. Queste operazioni

verranno eseguite rispettivamente dal gestore delle camere e servizi, ossia il direttore, e dal

gestore delle prenotazioni. Inoltre, i clienti avranno la possibilità di registrarsi, accedere,

finalizzare una prenotazione e visualizzarle.

## Obiettivi di progettazione

**Usabilità**

Il sistema deve garantire la validità dei dati di input attraverso vincoli e notifiche d’errore che segnalano eventuali incongruenze, assicurando consistenza e correttezza. L'interfaccia utente deve essere semplice, intuitiva, responsive e adattabile a diversi dispositivi, con una barra di navigazione che garantisca accessibilità e navigabilità uniforme.

**Riusabilità**

Il sistema deve garantire risuabilità, attraverso il concetto di ereditarietà del paradigma object oriented e l’utilizzo di diversi design pattern.

**Affidabilità**

Il sistema realizzato deve essere robusto, effettuando delle verifiche sui dati inseriti in modo tale da gestire gli input non validi, aggiungendo un ulteriore livello di controllo oltre alla prima verifica effettuata dopo l’inserimento degli input da parte dell’utente. Questo consente di affrontare situazioni non previste attraverso la gestione delle eccezioni.

## Linee guida per la documentazione dell’interfaccia

Le convenzioni usate nell’implementazione del sistema fanno riferimento alla specifica:

* Java Sun: [*https://checkstyle.sourceforge.io/sun\_style.html*](https://checkstyle.sourceforge.io/sun_style.html)

## Ottimizzazione del modello a oggetti

**Immagine che contiene testo, diagramma, linea, Piano

Descrizione generata automaticamente**

Il modello a oggetti, definito in fase di analisi, è stato ristrutturato aggiungendo l'attributo booleano isDeleted nelle classi Servizio e Camera per implementare la cancellazione logica degli oggetti nel database. In questo modo è possibile mantenere sempre un riferimento alla camera o al servizio prenotato, nonché alle relative informazioni persistenti, anche se l'oggetto viene "eliminato", ossia reso non disponibile.

Inoltre, la classe di associazione Servizio Prenotato è stata resa una classe concreta per gestirne la persistenza.

1. Packages

## Struttura del progetto

In questa sezione viene presentata la struttura organizzativa dei file all’interno del progetto. È la tipica organizzazione delle directory di un progetto SpringBoot, con Maven come gestore delle dipendenze.

* **.mvn**, contiene i file di configurazione Maven
* **src**, contiene i file sorgente del progetto
  + **main**
    - **java**, contiene i package e i file Java
    - **resources**, contiene le risorse relative all’interfaccia utente
      * **static**, contiene le risorse statiche (CSS, JS)
      * **templates**, contiene i file HTML dinamici realizzati con Thymeleaf
  + **test**
    - **java**, contiene le classi di testing
* **target**, contiene i file build di Maven

Il sistema è costituito da un package generale chiamato *it.unisa.hotelcampus*, e all’interno di esso è presente un package per ogni sottosistema individuato. Inoltre, all’interno del package generale, vi è un package *model*, contenente le classi entity e i DAO per gestire la persistenza, ed un package *utils*, dove vengono inserite le classi di utilità per funzionalità comuni all’intero sistema, come ad esempio la validazione.

Immagine che contiene testo, Rettangolo, schermata, diagramma

Descrizione generata automaticamente

**Package Gestione Utenti**

Immagine che contiene testo, schermata, Rettangolo, diagramma

Descrizione generata automaticamente

**Package Gestione Camere**

**Immagine che contiene testo, schermata, diagramma, Rettangolo

Descrizione generata automaticamente**

**Package Gestione Servizi**

**Immagine che contiene testo, schermata, Rettangolo, diagramma

Descrizione generata automaticamente**

**Package Gestione Prenotazioni**

Immagine che contiene testo, schermata, diagramma, Rettangolo

Descrizione generata automaticamente

1. Interfaccia delle classi

**JavaDoc**

La documentazione del codice delle classi verrà realizzata utilizzando JavaDoc.

Link JavaDoc HotelCampus: //da aggiungere

Di seguito vengono descritte le interfacce pubbliche di ciascun sottosistema. Viene specificata una descrizione dell’interfaccia, le dipendenze con altre classi o packages, attributi, operazioni ed eccezioni che possono essere sollevate.

**Package Gestione Utenti**

|  |  |
| --- | --- |
| Interfaccia | GestioneUtentiService |
| Descrizione | Gestione Utenti fornisce il servizio relativo all’autenticazione, creazione ed eliminazione degli account |
| Metodi | + autentica(String email, String password) : Utente  + creaUtente(String nome, String cognome, Date dataDiNascita, String nazionalità, String email, String password) : Utente  + elimina(Utente account) : boolean  + getUtente(String email) : Utente  + getUtenti() : Collection<Utente>  + getClienti() : Collection<Cliente> |
| Invariante di classe | // |

**Package Gestione Camere**

|  |  |
| --- | --- |
| Interfaccia | GestioneCamereService |
| Descrizione | Gestione Camere permette di inserire, rimuovere le camere e verificarne la disponibilità |
| Metodi | + getCamere() : Collection<Camera>  + creaCamera(int numero, TipoCamera tipo, int numeroMaxOspiti, int quadratura, int costo, String immagine) : Camera  + rimuoviCamera(Camera camera) : boolean  + getCamereDisponibili(Date checkIn, Date checkOut, int numeroOspiti) : Collection<Camera>  + verificaDisponibilita(Camera camera) : boolean |
| Invariante di classe | // |

**Package Gestione Servizi**

|  |  |
| --- | --- |
| Interfaccia | GestioneServiziService |
| Descrizione | Gestione Servizi permette di inserire, rimuovere le camere e verificarne la disponibilità |
| Metodi | + getServizi() : Collection<Servizio>  + creaServizio(String nome, String descrizione, int costo, String immagine) : Servizio  + rimuoviServizio(Servizio servizio) : boolean |
| Invariante di classe | // |

**Package Gestione Prenotazioni**

|  |  |
| --- | --- |
| Interfaccia | GestionePrenotazioniService |
| Descrizione | Gestione Prenotazioni ha il compito di aggiungere ed eliminare le prenotazioni effettuate dai clienti |
| Metodi | + getPrenotazioni() : Collection<Prenotazioni>  + creaPrenotazione(Date dataPrenotazione, Date dataCheckIn, Date dataCheckOut, int numeroOspiti, Camera camera, List<ServizioPrenotato> servizi, Cliente cliente) : Prenotazione  + eliminaPrenotazione(Prenotazione prenotazione) : boolean  + cercaPrenotazioni(String email, Date checkIn, Date checkOut) : Collection<Prenotazioni> |
| Invariante di classe | // |

**Package Entity**

|  |  |
| --- | --- |
| Interfaccia | Prenotazione |
| Descrizione |  |
| Metodi |  |
| Invariante di classe |  |

|  |  |
| --- | --- |
| Interfaccia | Cliente |
| Descrizione |  |
| Metodi |  |
| Invariante di classe |  |

1. Design Pattern

**Facade**

Il Facade Pattern è un design pattern strutturale che fornisce un'interfaccia semplificata per un insieme complesso di sottosistemi. Questo pattern nasconde la complessità del sistema sottostante fornendo un'interfaccia unificata di più alto livello, rendendo il sistema più facile da utilizzare. È particolarmente utile quando si ha bisogno di fornire un'interfaccia semplice per un sottosistema complesso. Si garantisce così un alto disaccoppiamento e si rende la piattaforma più manutenibile e più aggiornabile, poiché basterà cambiare l’implementazione dei metodi dell’interfaccia per implementare le modifiche.

Nel nostro caso ogni sottosistema avrà la propria interfaccia:

* GestioneServiziService
* GestioneUtentiService
* GestioneCamereService
* GestionePrenotazioniService

**DAO**

Un Data Access Object (DAO) è un pattern progettuale che fornisce un'interfaccia per interagire con un database in modo astratto. Grazie a questa struttura, è possibile eseguire operazioni sui dati senza preoccuparsi dei dettagli specifici del database. Il DAO consente di mappare le operazioni dell'applicazione sullo stato persistente dei dati, offrendo un modo standardizzato per gestire la persistenza.

In un'applicazione Spring Boot, il pattern DAO viene implementato creando una classe che rappresenta l'entità del database e un'interfaccia repository che estende una delle interfacce fornite da Spring Data JPA, come JpaRepository. Questo approccio consente di accedere e gestire i dati tramite metodi già pronti, come il salvataggio, la ricerca e la cancellazione, senza dover scrivere query SQL manuali.