Laurea Triennale in informatica – Università degli studi di Salerno

Corso di ingegneria del software – Prof. Carmine Gravino

ODD

Object Design Document

HotelSmart

Riferimento	G9_ODD_ver.1.0
Versione	1.0
Data	15/02/2021
Destinatario	Prof. Carmine Gravino
Presentato da	Team G9
Approvato da	Raffaele Sais

Laurea Triennale in informatica – Università degli studi di Salerno

Corso di ingegneria del software – Prof. Carmine Gravino

Revision history

Data	Versione	Descrizione	Autore
15/12/2021	0.1	Prima stesura	PC
27/12/2021	0.2	Stesura 1 e 2	PC
02/02/2022	0.3	Stesura 3	AD
10/02/2022	0.4	Stesura 4,5	Tutto il team
12/02/2022	0.5	Revisione generale	Tutto il team
15/02/2022	1.0	Revisione finale	Tutto il team

Laurea Triennale in informatica – Università degli studi di Salerno

Corso di ingegneria del software – Prof. Carmine Gravino

Team members

Nome	Ruolo nel progetto	Acronimo	Info di contatto
Alessandro D'Esposito	Team member	AD	desposito2@studenti.unisa.it
Pierpaolo Cammardella	Team member	PC	p.cammardella@studenti.unisa.it
Raffaele Sais	Project manager	RS	r.sais@studenti.unisa.it
Giovanni De Pierro	Team member	GD	g.depierro@studenti.unisa.it

Laurea Triennale in informatica – Università degli studi di Salerno

Corso di ingegneria del software – Prof. Carmine Gravino

Sommario

Laurea Triennale in informatica – Università degli studi di Salerno

Corso di ingegneria del software – Prof. Carmine Gravino

1. Introduzione

Il progetto HotelSmart è stato creato con l'intenzione di rivolgersi alle medio/piccole realtà alberghiere introducendo tutta una serie di servizi automatici, un'interfaccia semplice e andando a eliminare i metodi cartacei. Lo scopo principale è quello di andare a semplificare e centralizzare tutte quelle che sono le operazioni e necessità per quanto riguarda la gestione di un albergo / b&b.

1.1 Object Design Goals

-Robustezza: Il sistema deve risultare robusto e comportarsi in maniera adeguata anche in situazioni impreviste, ciò è possibile tramite un attento controllo di errori e input non validi e gestione dei sottosistemi con disattivazione.

-**Usabilità:** Il sistema deve risultare facilmente usabile e accessibile, questo è reso possibile tramite interfacce user-friendly

-Sicurezza: Il sistema non deve permettere ad utenti non autorizzati di utilizzare funzioni a cui non hanno il permesso.

1.2 Object trade-offs

I trade-off che sono stati individuati e presi in considerazione durante lo sviluppo del sistema software sono i seguenti:

-Buy vs Build

Nel considerare il trade-off buy vs build, sicuramente possiamo affermare che trovare un modulo sul mercato, corrispondente a uno a del sistema che stiamo realizzando, porterebbe a un risparmio sia di tempo, soldi e risorse umane. Prendendo in considerazione il budget disponibile per la realizzazione del sistema software oltre che l'obiettivo "secondario" di voler andare a migliorare le nostre capacità di costruire, realizzare, componenti software, abbiamo deciso di realizzare interamente da noi tutti i moduli del nostro sistema.

-Tempo di sviluppo vs manutenibilità

Nello sviluppo del sistema software si è cercato di minimizzare il più possibile i tempi di sviluppo, andando tuttavia ad intaccare la manutenibilità futura della piattaforma e non tenendo

particolarmente conto della chiarezza e comprensibilità del codice. Questa scelta è stata dettata principalmente dal budget di tempo disponibile per lo sviluppo del sistema stesso.

-Affidabilità vs Tempi di risposta

Si è preferito mantenere un controllo più rigido su tutti gli input andando a sacrificare la velocità sui tempi di risposta. Questa scelta è stata fatta per aumentare la robustezza del sistema.

1.3 Linee guida per la documentazione dell'interfaccia

Le linee guida includono una lista di regole che gli sviluppatori dovrebbero rispettare durante la progettazione delle interfacce.

Link a documentazione ufficiale sulle convenzioni

- [HTML] https://www.w3schools.com/html/
- [Java] https://docs.oracle.com/javase/tutorial/
- [CSS] https://www.w3.org/Style/CSS/specs.en.html
- [Stripe] https://stripe.com/docs
- [JavaScript] https://www.w3schools.com/js/
- [Bootstrap] https://www.w3schools.com/bootstrap/
- [jQuery] https://jquery.com/
- [jUnit] https://junit.org/junit5/
- [Mockito] https://site.mockito.org/

1.4 Definizioni, acronimi e abbreviazioni

Vengono riportati di seguito alcune definizioni presenti nel documento:

- **Package:** raggruppamento di classi, interfacce o file correlati.
- **Design pattern:** template di soluzioni a problemi ricorrenti impiegati per ottenere riuso e flessibilità.
- **Interfaccia:** insieme di signature delle operazioni offerte dalla classe.
- **Trade-off:** un compromesso tra due aspetti desiderabili ma incompatibili.
- **Javadoc:** sistema di documentazione offerto da Java, che viene generato sottoforma di interfaccia in modo da rendere la documentazione accessibile e facilmente leggibile.

1.5 Riferimenti

Di seguito una lista di riferimenti ad altri documenti utili durante la lettura:

- Statement of work
- Requirement analysis document
- System design document
- Object design document
- Test plan
- Manuale di tracciabilità
- Manuale di installazione
- Manuale utente

Il link contente la documentazione javadoc di HotelSmart:

- https://drlele08.github.io/HotelSmart/javadoc/index.html

Il link contenete la documentazione JaCoCo di HotelSmart:

- https://drlele08.github.io/HotelSmart/jacoco/index.html

2. Packages

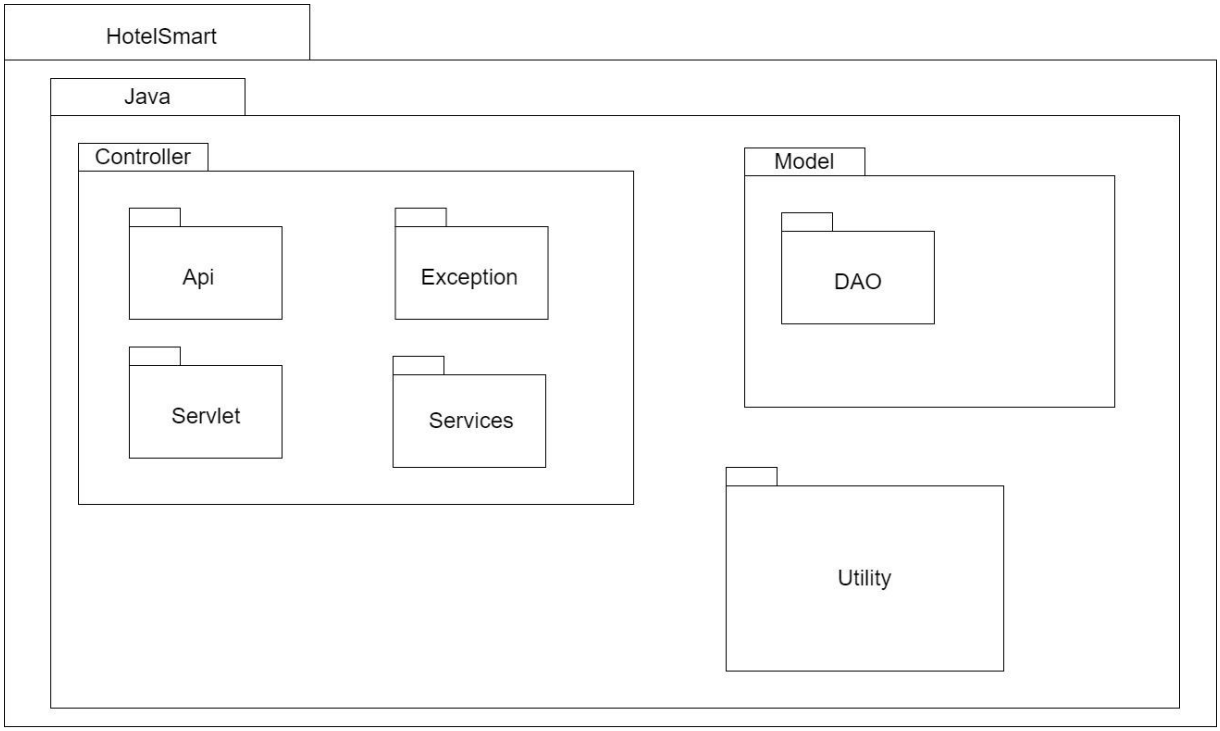
In questa sezione viene mostrata la suddivisione del sistema in package. Tale suddivisione è motivata dalle scelte architetturali prese.

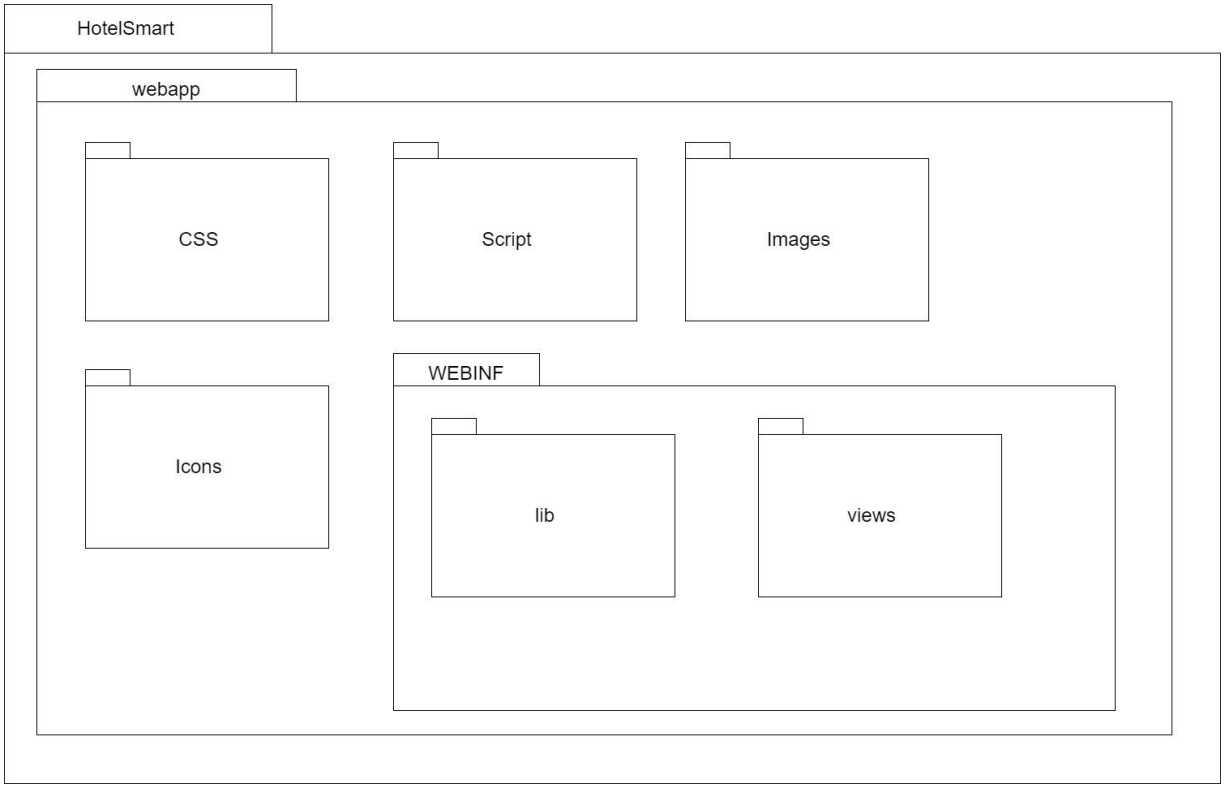
- **.idea**

- **src,** contiene tutti i file sorgente
  - **main**
    - **java**, contiene le classi relative alle componenti controller e model
    - **webapp,** contiene i file relativi alle componenti view
  - **test,** contiene tutto il necessario per il testing
    - **java,** contiene le classi java per l'implementazione del testing

- **target,** contiene tutti i file prodotti dal sistema di build Maven

Package HotelSmart





3. Class Interfaces

Di seguito saranno presentate le interfacce del package service.

Javadoc di HotelSmart

Al seguente link è possibile trovare la documentazione javadoc di HotelSmart.

https://drlele08.github.io/HotelSmart/javadoc/index.html

Package Service

Nome classe	PrenotazioneStanzaService
Descrizione	Questa classe permette di gestire le operazioni relative alla prenotazione stanza
Metodi	+inserisciPrenotazione(int ksUtente, int ksStanza, String dataInizio, String dataFine, List<PersonaAggiuntiva> listExtra):PrenotazioneStanza +selectBy(int value, int type): List<PrenotazioneStanza> +getAll(): List<PrenotazioneStanza> +editStato(int idPrenotazioneStanza, int stato):void +getPrenotazioneById(int idPrenotazione): PrenotazioneStanza +isRimborsabile(int idPrenotazione):boolean +addTokenStripe(int idPrenotazione, String tokenStripe):void +generateQrCode(int idPrenotazione):void
Invariante di classe	/

Nome Metodo	+inserisciPrenotazione(int ksUtente, int ksStanza, String dataInizio, String dataFine, List<PersonaAggiuntiva> listExtra) : PrenotazioneStanza
Descrizione	Questo metodo permette di inserire una nuova prenotazione stanza
Pre-condizione	**Context**: PrenotazioneStanzaService:: inserisciPrenotazione(int ksUtente, int ksStanza, String dataInizio, String dataFine, List<PersonaAggiuntiva> listExtra) **Pre** : not stanza.isDisponibile() && listExtra.size() + 1 == posti
Post-condizione	**Context**: PrenotazioneStanzaService:: inserisciPrenotazione(int ksUtente, int ksStanza, String dataInizio, String dataFine, List<PersonaAggiuntiva> listExtra) **Post** : service.selectBy(us.get().getIdUtente(),*UTENTE*).size == @pre.service.selectBy(us.get().getIdUtente(),*UTENTE*).size +1
Nome Metodo	+selectBy(int value, int type): List<PrenotazioneStanza>
Descrizione	Questo metodo ritorna una lista di stanze dato un tipo(Utente,Stanza,Stato) ricercare la corrispettiva ks dall'input value
Pre-condizione	/
Post-condizione	/
Nome Metodo	+getAll(): List<PrenotazioneStanza>
Descrizione	Questo metodo ritorna una lista di tutte le prenotazioni stanze
Pre-condizione	/

Post-condizione	/
Nome Metodo	+editStato(int idPrenotazioneStanza, int stato): void
Descrizione	Questo metodo modifica lo stato di una prenotazione
Pre-condizione	/
Post-condizione	**Context**: PrenotazioneStanzaService:: editStato(int idPrenotazioneStanza, int stato) **Post** : not service.getPrenotazioneById(idPrenotazione).getKsStato() == @pre. service.getPrenotazioneById(idPrenotazione).getKsStato()
Nome Metodo	+getPrenotazioneById(int idPrenotazione) : PrenotazioneStanza
Descrizione	Questo metodo ritorna una prenotazione dato un id
Pre-condizione	/
Post-condizione	/
Nome Metodo	+isRimborsabile(int idPrenotazione) : Boolean
Descrizione	Questo metodo ritorna True se la prenotazione è rimborsabile
Pre-condizione	/
Post-condizione	
Nome Metodo	+addTokenStripe(int idPrenotazione, String tokenStripe) : void
Descrizione	Questo metodo inserisce il token stripe alla prenotazione dopo aver effettuato il pagamento della prenotazione
Pre-condizione	/
Post-condizione	**Context**: PrenotazioneStanzaService:: addTokenStripe(int idPrenotazione, String tokenStripe) **Post** : not service.getPrenotazioneById(idPrenotazione). getTokenStripe().equals(@pre.service.getPrenotazioneById(idPrenotazione).get TokenStripe())
Nome Metodo	+generateQrCode(int idPrenotazione) : void
Descrizione	Questo metodo genera uma stringa casuale univoca
Pre-condizione	**Context**: PrenotazioneStanzaService:: generateQrCode(int idPrenotazione) **Pre** : service.getPrenotazioneById(idPrenotazione).getKsStato == 2
Post-condizione	**Context**: PrenotazioneStanzaService:: generateQrCode(int idPrenotazione) **Post** : not service.getPrenotazioneById(idPrenotazione). getTokenQr().equals(@pre.service.getPrenotazioneById(idPrenotazione).getTok enQr())

Nome classe	PrenotazioneServizioService
Descrizione	Questa classe permette di gestire le operazioni relative alla prenotazione servizio
Metodi	+createPrenotazione(int ksPrenotazioneStanza, int ksServizio, int numPersone, Date dataPrenotazioneServizio) : void +getAllByUser(int idUtente): List<PrenotazioneServizio> +deletePrenotazioneById(): void
Invariante di classe	/

Nome Metodo	+createPrenotazione(int ksPrenotazioneStanza, int ksServizio, int numPersone, Date dataPrenotazioneServizio) : void
Descrizione	Questo metodo permette di inserire una nuova prenotazione servizio
Pre-condizione	**Context**: PrenotazioneServizioService:: createPrenotazione(int ksPrenotazioneStanza, int ksServizio, int numPersone, Date dataPrenotazioneServizio) **Pre** : service.getPrenotazioneById(idPrenotazione).getKsStato == 3

Post-condizione	**Context**: PrenotazioneServizioService:: createPrenotazione(int ksPrenotazioneStanza, int ksServizio, int numPersone, Date dataPrenotazioneServizio) **Post** : service.getAllByUser(idUtente).size == @pre.service.getAllByUser(idUtente).size + 1
Nome Metodo	+ getAllByUser(int idUtente): List<PrenotazioneServizio>
Descrizione	Questo metodo ritorna una lista di tutte le prenotazioni servizio
Pre-condizione	/
Post-condizione	/
Nome Metodo	+deletePrenotazioneById(int idPrenotazione): void
Descrizione	Questo metodo elimina una prenotazione servizio
Pre-condizione	/
Post-condizione	**Context**: PrenotazioneServizioService:: deletePrenotazioneById(int idPrenotazione) **Post** : service.getAllByUser(idUtente).size == @pre.service.getAllByUser(idUtente).size - 1

Nome classe	RuoloService
Descrizione	Questa classe permette di gestire le operazioni relative al ruolo di um utente
Metodi	+getAll() : List<Ruolo> +getById(int Ruolo): String +getByRuolo(): int
Invariante di classe	/

Nome Metodo	+getAll() : List<Ruolo>
Descrizione	Questo metodo ritorna una lista di tutti i ruoli
Pre-condizione	/
Post-condizione	/
Nome Metodo	+ getById(int ruolo): String
Descrizione	Questo metodo ritorna una stringa contenente il nome del ruolo associato all'id dell'utente
Pre-condizione	/
Post-condizione	/
Nome Metodo	+getByRuolo(String ruolo): int
Descrizione	Questo metodo ritorna l'id del ruolo associato al nome
Pre-condizione	/
Post-condizione	/

Nome classe	ServizioService
Descrizione	Questa classe permette di gestire le operazioni relative al servizio
Metodi	+createServizio(String nome, String descrizione, String foto, double costo, int limitePosti) : void +updateServizio(int idServizio, String nome, String descrizione, String foto, double costo, int limitePosti) : void +getAll(): List<Servizio> +getById(int idServizio): Servizio
Invariante di classe	/

Nome Metodo	+ createServizio(String nome, String descrizione, String foto, double costo, int limitePosti) : void
Descrizione	Questo metodo crea un nuovo servizio
Pre-condizione	**Context**: ServizioService:: createServizio(String nome, String descrizione, String foto, double costo, int limitePosti) **Pre** : user.getRuolo == 1
Post-condizione	**Context**: ServizioService:: createServizio(String nome, String descrizione, String foto, double costo, int limitePosti) **Post** : service.getAll().size == @Pre.service.getAll().size + 1
Nome Metodo	+ updateServizio(int idServizio, String nome, String descrizione, String foto, double costo, int limitePosti) : void
Descrizione	Questo metodo modifica i dati di un servizio già esistente
Pre-condizione	**Context**: ServizioService:: updateServizio(int idServizio, String nome, String descrizione, String foto, double costo, int limitePosti) **Pre** : user.getRuolo == 1
Post-condizione	**Context**: ServizioService:: updateServizio(int idServizio, String nome, String descrizione, String foto, double costo, int limitePosti) **Post**:not service.getById(idServizio).equals(@pre.service.getById(idServizio))
Nome Metodo	+ getAll(): List<Servizio>
Descrizione	Questo metodo ritorna una lista di tutti i servizi
Pre-condizione	/
Post-condizione	/
Nome Metodo	+getById(int idServizio): Servizio
Descrizione	Questo metodo ritorna il servizio associato all'id
Pre-condizione	/
Post-condizione	/

Nome classe	StanzaService
Descrizione	Questa classe permette di gestire le operazioni relative alla stanza
Metodi	+getStanze(): List<Stanza> +getOfferte(): List<Stanza> +get_Min_And_Max_Prices(): List<Double> +search(Boolean animaleDomestico, Boolean fumatore, Integer numeroOspiti, Double costoNotteMinimo, Double costoNotteMassimo, Double scontoMinimo, Double scontoMassimo, java.sql.Date dataIn, Date dataOut): List<Stanza> +selectById(Integer stanzaId): Stanza +insertStanza(boolean animale, boolean fumatore, int lettiSingoli, int lettiMatrimoniali, double costoNotte, double sconto): void +updateStanza(int idStanza, boolean animale, boolean fumatore, int lettiSingoli, int lettiMatrimoniali, double costoNotte, double sconto):void
Invariante di classe	/

Nome Metodo	+getStanze(): List<Stanza>
Descrizione	Questo metodo restituisce una lista di tutte le stanze
Pre-condizione	/
Post-condizione	/
Nome Metodo	+getOfferte(): List<Stanza>
Descrizione	Questo metodo ritorna una lista di tutte le stanze che in offerta
Pre-condizione	**Context**: StanzaService:: getOfferte() **Pre** : stanza.getSconto > 0

Post-condizione	/
Nome Metodo	+get_Min_And_Max_Prices(): List<Double>
Descrizione	Questo metodo ritorna una lista di Double contenente il prezzo minimo e il prezzo massimo
Pre-condizione	/
Post-condizione	/
Nome Metodo	+search(Boolean animaleDomestico, Boolean fumatore, Integer numeroOspiti, Double costoNotteMinimo, Double costoNotteMassimo, Double scontoMinimo, Double scontoMassimo, java.sql.Date dataIn, Date dataOut): List<Stanza>
Descrizione	Questo metodo ritorna una lista di tutte le stanze che soddisfano i requisiti in input
Pre-condizione	/
Post-condizione	/
Nome Metodo	+selectById(Integer stanzaId) : Stanza
Descrizione	Questo metodo ritorna una stanza dato un id
Pre-condizione	/
Post-condizione	/
Nome Metodo	+insertStanza(boolean animale, boolean fumatore, int lettiSingoli, int lettiMatrimoniali, double costoNotte, double sconto): void
Descrizione	Questo metodo crea una nuova stanza
Pre-condizione	**Context**: StanzaService:: insertStanza(boolean animale, boolean fumatore, int lettiSingoli, int lettiMatrimoniali, double costoNotte, double sconto) **Pre** : user.getRuolo == 1
Post-condizione	**Context**: StanzaService:: insertStanza(boolean animale, boolean fumatore, int lettiSingoli, int lettiMatrimoniali, double costoNotte, double sconto) **Post** : service.getStanze().size == @Pre.service. getStanze ().size + 1
Nome Metodo	+updateStanza(int idStanza, boolean animale, boolean fumatore, int lettiSingoli, int lettiMatrimoniali, double costoNotte, double sconto):void
Descrizione	Questo metodo modifica i dati di una stanza già esistente
Pre-condizione	**Context**: StanzaService:: updateStanza(int idStanza, boolean animale, boolean fumatore, int lettiSingoli, int lettiMatrimoniali, double costoNotte, double sconto) **Pre** : user.getRuolo == 1
Post-condizione	**Context**: StanzaService:: updateStanza(int idStanza, boolean animale, boolean fumatore, int lettiSingoli, int lettiMatrimoniali, double costoNotte, double sconto) **Post**:not service.selectById(stanzaId).equals(@pre.service.selectById(stanzaId))

Nome classe	StatoService
Descrizione	Questa classe permette di gestire le operazioni relative allo stato di uma prenotazione stanza
Metodi	+getAll() : List<Stato> +getById(int idStato): String
Invariante di classe	/

Nome Metodo	+getAll() : List<Stato>
Descrizione	Questo metodo ritorna una lista di tutti i stati
Pre-condizione	/
Post-condizione	/
Nome Metodo	+ getById(int idStato): String

Descrizione	Questo metodo ritorna una stringa contenente il nome dello stato associato all'id della prenotazione stanza
Pre-condizione	/
Post-condizione	/

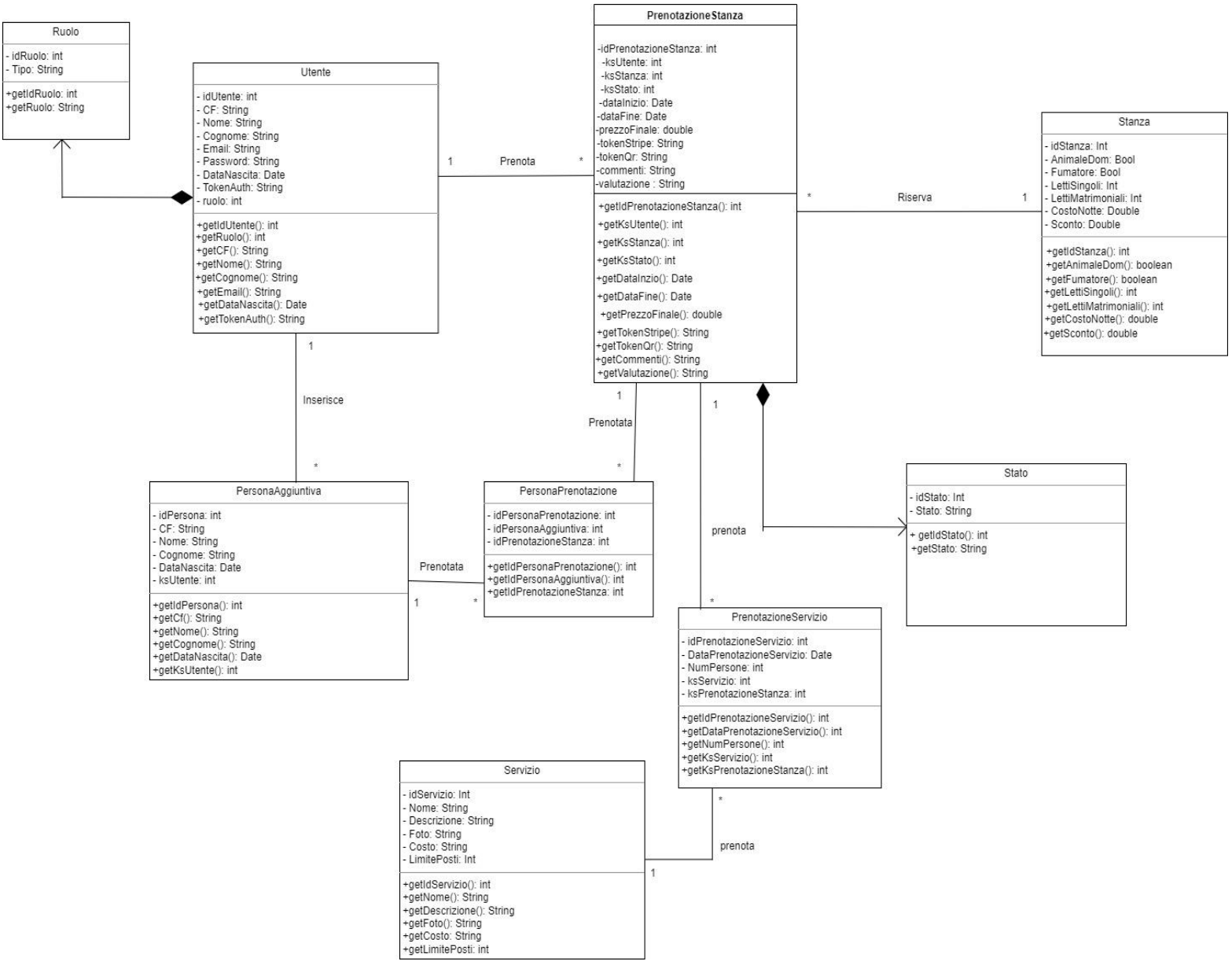
Nome classe	UtenteService
Descrizione	Questa classe permette di gestire le operazioni relative all'utente
Metodi	+doLogin(String email, String pwd): Utente +doLogin(int idUtente, String tokenAuth): Utente +doRegistrazione(String cf, String nome, String cognome, String email, Date data,String pwd): Utente +editPassword(int idUtente,String token,String oldPwd,String newPwd): void +editAnagrafica(int idUtente, String tokenAuth, String nome, String cognome, String cf, String dataNascitaStr, String email): void +getUtenteByPrenotazioneStanza(int idPrenotazione): Utente +getAll(): List<Utente> +editRuolo(int idUtente, int ruolo): void
Invariante di classe	/

Nome Metodo	+doLogin(String email, String pwd): Utente
Descrizione	Questo metodo restituisce un utente se l'email e la password corrispondono ad un utente all'interno del database
Pre-condizione	/
Post-condizione	**Context**: UtenteService:: doLogin(String email, String pwd) **Post** : utente != null
Nome Metodo	+doLogin(int idUtente, String tokenAuth): Utente
Descrizione	Questo metodo restituisce un utente
Pre-condizione	/
Post-condizione	**Context**: UtenteService:: doLogin(int idUtente, String tokenAuth) **Post** : utente != null
Nome Metodo	+ doRegistrazione(String cf, String nome, String cognome, String email, Date data,String pwd): Utente
Descrizione	Questo metodo crea un nuovo utente
Pre-condizione	/
Post-condizione	**Context**: UtenteService:: doRegistrazione(String cf, String nome, String cognome, String email, Date data,String pwd) **Post** : service.getAll().size == @Pre.service.getAll().size + 1
Nome Metodo	+ editPassword(int idUtente,String token,String oldPwd,String newPwd): void
Descrizione	Questo metodo modifica la password di un utente registrato
Pre-condizione	/
Post-condizione	/
Nome Metodo	+ editAnagrafica(int idUtente, String tokenAuth, String nome, String cognome, String cf, String dataNascitaStr, String email): void
Descrizione	Questo metodo modifica i dati di un utente registrato
Pre-condizione	/
Post-condizione	**Context**: UtenteService:: editAnagrafica(int idUtente, String tokenAuth, String nome, String cognome, String cf, String dataNascitaStr, String email) **Post** : not service.doLogin(int idUtente, String tokenAuth).equals(@Pre.service.doLogin(int idUtente, String tokenAuth))

Nome Metodo	+ getUtenteByPrenotazioneStanza(int idPrenotazione): Utente
Descrizione	Questo metodo ritorna l'utente che ha effettuato la prenotazione della stanza dato un id
Pre-condizione	/
Post-condizione	/
Nome Metodo	+getAll(): List<Utente>
Descrizione	Questo metodo ritorna la lista di tutte le stanze
Pre-condizione	/
Post-condizione	/
Nome Metodo	+editRuolo(int idUtente, int ruolo): void
Descrizione	Questo metodo modifica il ruolo assegnato all'utente
Pre-condizione	**Context**: UtenteService:: editRuolo(int idUtente, int ruolo) **Pre** : user.getRuolo == 1
Post-condizione	**Context**: UtenteService:: editRuolo(int idUtente, int ruolo) **Post :**not service.doLogin(int idUtente, String tokenAuth).getRuolo() = @Pre.service.doLogin(int idUtente, String tokenAuth).getRuolo()

4. Class Diagram

Di seguito viene riportato il class diagram del sistema. Per quanto riguarda quest'ultimo, è stato scelto di rimuovere l'entità Foto perché ritenuta con pochi comportamenti e attributi. Era stata considerata la stessa cosa per quanto riguarda le entità Ruolo e Stato; tuttavia, si è scelto di non effettuare modifiche per motivi di efficienza.



5. Design Pattern

Nella presente sezione si andranno a descrivere e dettagliare i design patterns utilizzati nello sviluppo dell'applicativo HotelSmart. Per ogni pattern si darà:

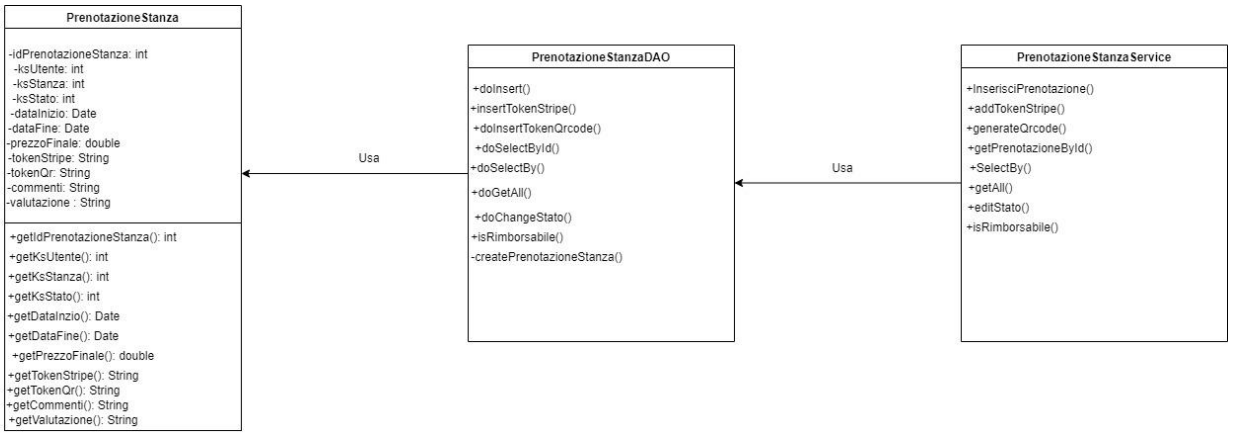
- Una breve introduzione teorica
- Il problema che doveva risolvere all'interno di HotelSmart
- Una brevissima spiegazione di come si è risolto il problema su HotelSmart
- Un grafico della struttura delle classi che implementano il pattern

DAO

Un DAO (Data Access Object) è un pattern che offre un'interfaccia astratta per alcuni tipi di database. Mappando le chiamate dell'applicazione allo stato persistente, il DAO fornisce alcune operazioni specifiche sui dati senza esporre i dettagli del database. I DAO sono utilizzabili nella maggior parte dei linguaggi e la maggior parte dei software con bisogni di persistenza, principalmente viene associato con applicazioni JavaEE che utilizzano database relazionali.

In HotelSmart abbiamo deciso di utilizzare il pattern DAO per andare realizzare tutte le interazioni con il database, questo ci ha permesso di creare un maggiore livello di astrazione e una più facile manutenibilità.

Di seguito un esempio dell'utilizzo del pattern DAO in HotelSmart:

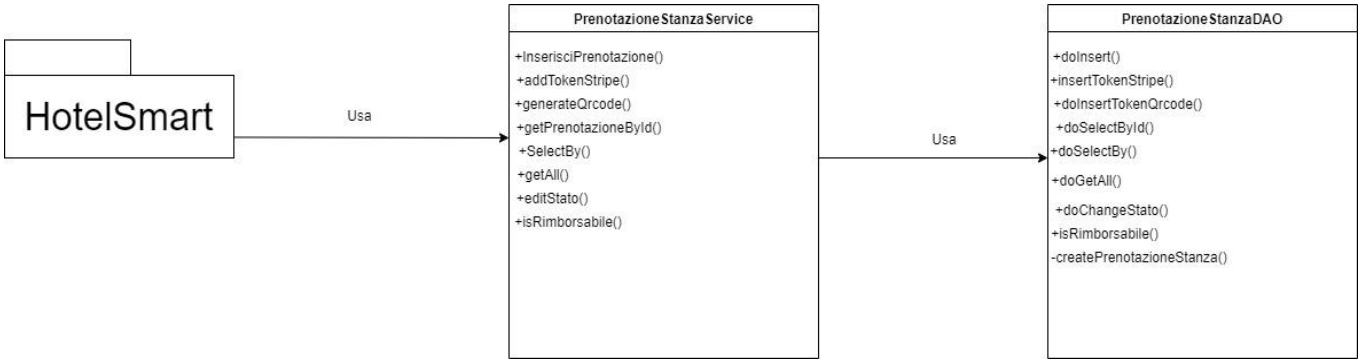


FACADE

Il Facade è un design pattern che permette, implementando una interfaccia semplificata di accedere a sottosistemi più complessi. Tramite questo pattern andiamo a nascondere l'interazione tra più classi per l'esecuzione di operazioni complesse e la complessità stessa dell'operazione. Andiamo a nascondere i dettagli implementativi e favoriamo un basso accoppiamento e più alta manutenibilità.

In HotelSmart abbiamo deciso di utilizzare il design pattern Facade per andare a nascondere la complessità e l'interazione tra le varie classi per realizzare una operazione di interazione con i dati persistenti. Andando a fornire così un'interfaccia che favorisce un basso accoppiamento fra i vari DAO, inoltre nasconde e facilità l'utilizzo degli stessi da parte della logica del sistema.

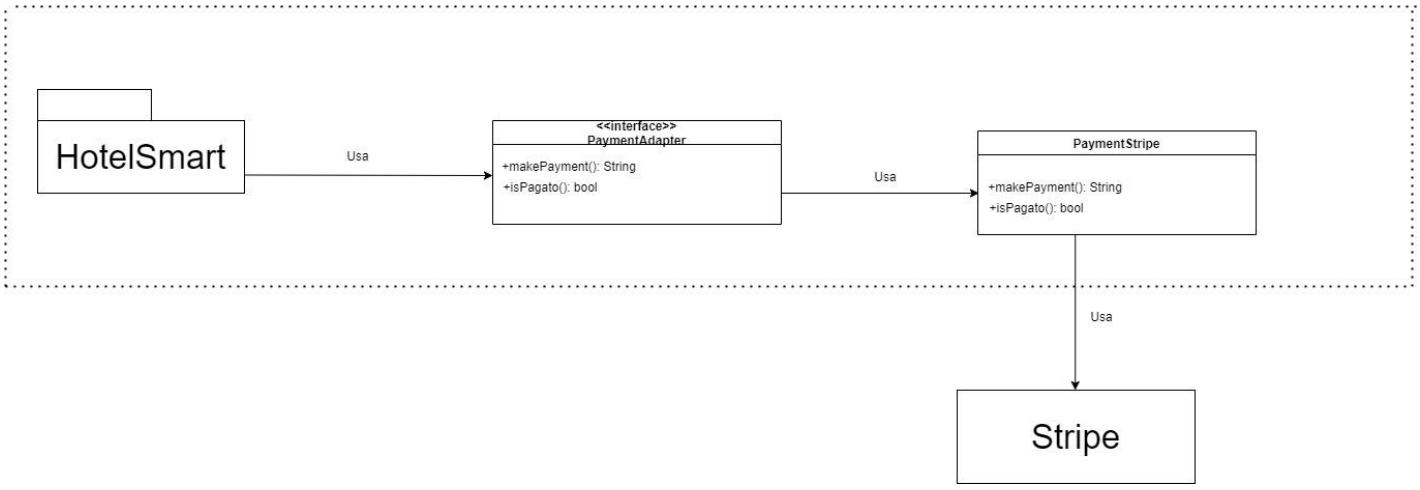
Di seguito un esempio dell'utilizzo del pattern Facade in HotelSmart:



Adapter

L'Adapter è un design pattern strutturale, ovvero quei design pattern che facilitano la progettazione attraverso la semplificazione delle relazioni tra entità. Adapter, in particolare, permette ad oggetti con differenti interfacce di collaborare. Si implementa attraverso una classe "adapter", che si occupa di convertire i dati in oggetti comprensibili dal sistema.

In HotelSmart si è scelto di utilizzare un pattern Adapter per facilitare l'interfacciarsi del sistema con gateway di pagamento online. Il gateway di pagamento online con cui ci interfacciamo è Stripe. Tramite l'utilizzo di questo pattern inoltre sarà possibile aggiungere con facilità nuovi metodi di pagamento in futuro.



6. Glossario

Sigla/termine	Descrizione
HotelSmart	Nome dell'applicativo che si andrà a realizzare
Piattaforma	Base software o hardware sulla quale sono sviluppato o eseguite applicazioni
Package	Raggruppamento di classi e interfacce
DAO	Data Access Object, implementazione dell'omonimo pattern architetturale che si occupa di fornire un accesso in modo astratto ai dati persistenti.
FACADE	Un oggetto che permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro.