UNIVERSITÀ DI TRENTO
Dipartimento di Ingegneria e Scienza dell'Informazione

Nome progetto :

BellHotel

Nome documento :

Documento di Architettura

Indice

Scopo del documento

Il presente documento riporta l'architettura del progetto BellHotel, realizzata grazie ad un diagramma delle classi e al linguaggio OCL.
Gli obiettivi di questo documento saranno:

- Elencare e descrivere le classi utilizzate;
- Presentare il diagramma delle classi.

Elenco delle classi

1: Utente autenticato

UtenteAutenticato
- UserID : int - email : String - password : String - logged : bool - isGestore : bool
+ impostazioni(logged : bool) + getLogged() + setLogged()

La classe *UtenteAutenticato* generalizza le due tipologie di utente autenticato, ossia *Cliente* e *Gestore*. La classe implementa un'interfaccia utile per la gestione di questa tipologia di utente, in particolar modo relazionandosi con *Impostazioni* grazie al campo *logged*.

2: Cliente

Cliente
+ prenota(stanza : Stanza,periodo : Periodo,datiPersonali : DatiPersonali,choiceOfPay : String) + navigazioneSito(offerte : Offerta, provincia: String) + modificaPrenotazione(prenotazione : Prenotazione) + recensisci(reservationCode : String, reserveFulfilled : bool) + visualizzaRecensioni() + getCliente()

```
context Cliente::recensisci(reservationCode : String, reserveFulfilled : bool)
pre: self.Cliente.getReserveFulfilled() = true

context Cliente::prenota(stanza : Stanza,periodo : Periodo,datiPersonali : DatiPersonali,choiceOfPay : String
post: stanza.setReserved(true: bool)
```

La classe *Cliente* implementa un'interfaccia utile alla gestione di questa prima tipologia di utente autenticato. Essa, infatti, eredita campi e metodi di *UtenteAutenticato* e implementa ulteriori metodi dedicati. *Cliente* può infatti effettuare prenotazioni e modifiche alle stesse, nonché lasciare delle recensioni al termine del suo soggiorno.

3: Gestore

Gestore
+ visualizzaRecensioni() + rispondiCommenti(commentoScritto : String, isGestore : bool) + creaOfferte(hotel : Hotel, isGestore : bool) + modificaOfferte(offerta : Offerta, isGestore: bool)

La classe *Gestore* implementa un'interfaccia utile alla gestione di questa seconda tipologia di utente autenticato. Essa eredita campi e metodi da utente autenticato ma differisce per operazioni possibili da *Cliente* grazie al campo *isGestore* di *UtenteAutenticato*. Legate alla classe *Gestore* vi sono infatti metodi riguardanti la creazione e gestione di offerte.

4: Utente non autenticato

UtenteNonAutenticato
- logged : bool
+ login() + registrazione(email: String) + esistenzaEmail() + navigazioneSito() + impostazioni(logged : bool) + getLogged() + setLogged()

```
context UtenteNonAutenticato::registrazione()
pre: self.esistenzaEmail() = false

context UtenteNonAutenticato::login()
pre: self.esistenzaEmail() = true
post : self.setLogged(true:bool)
```

La classe *UtenteNonAutenticato* implementa un'interfaccia per la gestione di questa tipologia di utente. Essa differisce da *UtenteAutenticato* per quanto riguarda campi,metodi e operazioni possibili. Infatti questa classe non presenta le stesse possibilità di interazione con *Impostazioni* non potendo accedere ad alcuni metodi di quest'ultima a causa del campo *logged*. Ha però metodi che permettono di modificare questo stato.
Nel contesto viene specificato come questo utente potrà registrarsi con la propria email solo se questa ancora non esiste nel database del sito.
Allo stesso modo, solo se la mail è presente, l'utente potrà fare il login.

5: Impostazioni

Impostazioni
+ logout(logged : bool) + recuperaPsw(logged : bool,email : String) + modificaPsw(logged : bool,email : String) + modificaLingua(lingua : Lingua)

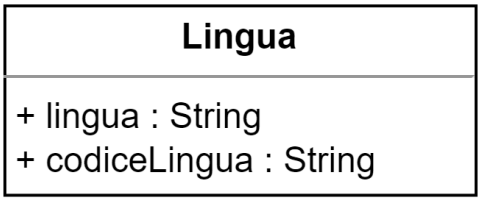
```
context Impostazioni::logout()
pre: self.getLogged() = true
post : self.setLogged(false:bool)

context Impostazioni::recuperaPsw()
pre: self.getLogged() = false

context Impostazioni::modificaPsw()
pre: self.getLogged() = true
```

La classe *Impostazioni* implementa un'interfaccia utile alla gestione delle possibili operazioni che possono essere eseguite da *UtenteAutenticato* e *UtenteNonAutenticato*. Ciò avviene mediante un campo *logged* che ha un valore differente fra *UtenteNonAutenticato* e *UtenteAutenticato*.
Nel contesto viene specificato come un utente, che ha già fatto il login al sito, può effettuare il logout.
L'utente potrà inoltre recuperare la propria password o modificarla, sempre mediante le impostazioni.

6: Lingua

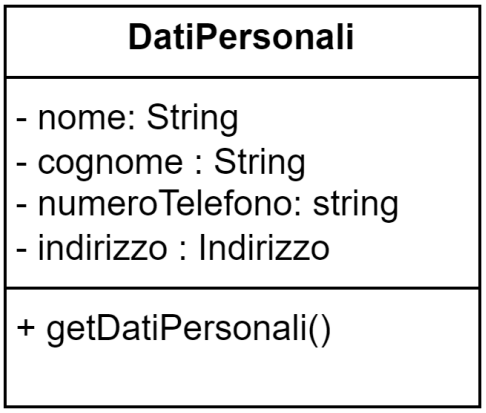


La classe *Lingua* è un'interfaccia utile per il passaggio del campo *lingua* in *Impostazioni*.

7: Offerta

La classe *Offerta* implementa un'interfaccia utile alla gestione delle offerte sul sito. Essa è legata sia a *UtenteNonAutenticato* che alle due tipologie di utente autenticato, ossia *Cliente* e *Gestore*, anche se hanno tipologie differenti di interazione con essa. Infatti *Gestore* può pubblicare e modificare le offerte, mentre *UtenteNonAutenticato* e *Cliente* le possono visualizzare.

8: Dati Personali



DatiPersonali è una classe utile per definire e raggruppare campi aggiuntivi che *Cliente* dovrà inserire al momento della prenotazione.

9: Email

Email
- email : String - object : String - message : String
+ sendEmail(email : String,object: String,message : String)

La classe *Email* è una classe utile per la gestione delle email all'interno del sito, ed è legata in particolar modo a metodi come *recuperaPsw()* e *modificaPsw()* all'interno di *Impostazioni*. Essa si interfaccia con un sistema di invio mail esterno per poter inviare le email.

10: Prenotazione

Prenotazione
- reservationCode : String - periodo : Periodo - numeroPersone : int - IDhotel : int - IDstanza : int - cliente : Cliente - onTimeReserve : bool - costoPrenotazione : float - reserveFulfilled : bool
+ pagamento(cliente : Cliente, costoPrenotazione : float) + setOntimeReserve() + setReserveFulfilled() + getPeriodo() + getReserveFulfilled()

```
context Prenotazione::setOnTimeReserve()
pre: self.Prenotazione.getPeriodo().inizioSoggiorno - actual_Date < 7

context Prenotazione:: setReserveFulfilled()
pre: actual_Date > self.Prenotazione.getperiodo().fineSoggiorno
post: self.Prenotazione.setReserveFulfilled(true: bool)
post: stanza.setReserved(false: bool)
```

La classe *Prenotazione* implementa un'interfaccia utile per la gestione di una prenotazione.
Nello specifico, questa classe raccoglie i dati delle classi *Stanza* e *DatiStanza*, così che il cliente possa prenotare la camera d'hotel da lui desiderata.
Associata vi è inoltre la classe *Pagamento*, indispensabile per portare a termine il processo di prenotazione.
Nel caso in cui manchino meno di sette giorni all'inizio della prenotazione, il campo *onTimeReserve* ritorna utile per definire l'impossibilità del cliente di apportare modifiche alla sua prenotazione.
Inoltre, una volta finito il periodo di soggiorno, la prenotazione del cliente viene dichiarata come completata e la stanza d'hotel precedentemente prenotata ritorna disponibile per un altro utente.

11 : Recensione

Recensione
- numeroStelle : int - commentoScritto : String
+ getRecensione()

La classe *Recensione* è una classe utile a raccogliere le recensioni lasciate da *Cliente*. Essa è infatti associata con *Cliente* ma anche con *Gestore*, dato che quest'ultimo è in grado di interagire con le recensioni degli utenti, dalla semplice visualizzazione a rispondere alle stesse.

12 : Pagamento

Pagamento
- wayOfPay : String[ ] - choiceOfPay : String
+ pay(choiceOfPay : String, costoPrenotazione : float)

La classe *Pagamento* è una classe utile alla gestione dei pagamenti all'interno del sito. Essa raccoglie la scelta della metodologia di pagamento di *Cliente* e in base ad essa si collega con il rispettivo sistema di pagamento esterno.

13: Stanza

Stanza
- IDstanza : int - reserved : bool - datiStanza : DatiStanza
+ isReserved() + setReserved() + getIDstanza()

La classe *Stanza* è una classe che implementa un'interfaccia utile al cliente per visualizzare una stanza d'hotel e, nel caso, prenotarla. Essa è associata, oltre che ad *Hotel*,anche a *DatiStanza* per la visualizzazione di dati ulteriori riguardanti la stanza stessa.

14: DatiStanza

DatiStanza
- numPostiLetto : int - servizi : String[ ] - hotelAppartenenza : Hotel - prezzo : float
+ getDatiStanza()

La classe *DatiStanza* è una classe utile alla visualizzazione di dati aggiuntivi di una stanza. Essa è infatti associata a *Stanza*.

15: Hotel

Hotel
- IDhotel : int - nomeHotel : String - datiHotel : DatiHotel - IDgestore: int
+ getIDhotel()

La classe *Hotel* è una classe che implementa un'interfaccia utile a *Cliente* per visualizzare un hotel fra quelli presenti sul sito. Essa è inoltre associata a *DatiHotel* per la visualizzazione di dati aggiuntivi riguardanti l'hotel stesso.

16: DatiHotel

DatiHotel
- indirizzo : Indirizzo - provincia : String - numStelle : int - numStanze : int - tipoStanza : String
+ getDatiHotel()

La classe *DatiHotel* è una classe utile per raccogliere quelli che sono dati aggiuntivi riguardanti l'hotel selezionato.

17: Indirizzo

Indirizzo
- numeroCivico : int - nomeVia : String - nomePaese : String - nomeProvincia : String
+ getIndirizzo()

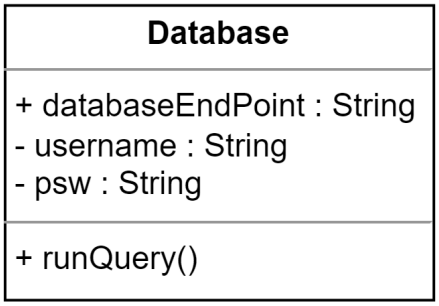
La classe *Indirizzo* è una classe utile alla specifica dell'indirizzo di un determinato hotel all'interno di una provincia.

18: Periodo

Periodo
- inizioSoggiorno: Date - fineSoggiorno : Date
+ getPeriodo()

La classe *Periodo* è una classe utile a raccogliere le date di inizio e fine soggiorno di un cliente, periodo in cui *Stanza* sarà quindi occupata.

19: Database



La classe *Database* è una classe utile ad interfacciarsi con il database contenente dati di utenti, stanze e hotel. Essa permette di inviare query al database, raccogliendo i risultati ricevuti dallo stesso.

Diagramma delle classi con OCL