Università degli Studi di Salerno
Corso di Ingegneria del Software

MyHotel
ODD
Versione 1.5

Partecipanti:

Nome	Matricola
Marco Avagliano	0512102880
Alessandro Criscuolo	0512103132

Scritto da:	Marco Avagliano,Alessandro Criscuolo

Revision History

Data	Versione	Descrizione	Autore
09/01/2017	1.0	Introduzione, object design trade-offs.	Marco Avagliano, Alessandro Criscuolo
10/01/2017	1.1	Prima stesura packages.	Marco Avagliano
10/01/2017	1.2	Linee guida per la documentazione dell'interfaccia.	Marco Avagliano
11/01/2017	1.3	Modifiche linee guida per la documentazione dell'interfaccia.	Marco Avagliano
11/01/2017	1.4	Design pattern.	Marco Avagliano, Alessandro Criscuolo
11/01/2017	1.5	Definizione interfacce in OCL.	Alessandro Criscuolo

Indice

1.Introduzione

1.1 Trade-off

Comprensibilità vs costi

Si preferisce aggiungere dei costi per la documentazione al fine di rendere il codice comprensibile anche alle persone non coinvolte nel progetto o le persone coinvolte che non hanno lavorato a quella parte in particolare. Commenti diffusi nel codice facilitano la comprensione, di conseguenza migliorare la comprensibilità agevola il mantenimento e anche il processo di modifica.

Interfaccia vs Easy-use

La piattaforma è molto semplice e di facile uso poichè avrà un'interfaccia chiara e intuitiva. Aumentare la facilità di utilizzo significa allargare le funzionalità della piattaforma ad un maggior numero di utenti. Per questo

il team di sviluppo, nel momento di implementare l'interfaccia grafica, non solo cercherà di dare alla pagina uno stile piacevole alla vista ma fornire una struttura di interazione tale da rendere semplice all'utente la gestione delle sue operazioni.

Tempo di risposta vs Spazio di memoria

Basandoci sui design goals descritti in precedenza (SDD), il tempo risposta deve essere minimo. Le operazioni che usano più tempo all'interno del sistema sono gli accessi al database, in quanto, oltre all'accesso al disco bisogna effettuare operazioni di unione e controllo sulle tabelle generate dalle query. Dato che le operazioni effettuate dal sistema spesso risultano nella creazione delle stesse tabelle con gli stessi dati al loro interno, abbiamo rilevato che generando le tabelle necessarie in precedenza, in modo tale che quando viene effettuata un'operazione che ha bisogno di tali dati, non si ha la necessità di generare la tabella a runtime, ma semplicemente si effettua una ricerca in una già generata. Questo porta ad un utilizzo di memoria maggiore all'interno del database ma velocizza di molto le operazioni effettuate.

1.2 Componenti off-the-shelf

Per il progetto software che si vuole realizzare facciamo uso di componenti off-the-shelf, che sono componenti software disponibili sul mercato per facilitare la creazione del progetto.
Per il sistema che si vuole realizzare ci interessa un framework per applicazioni web.

Il framework che andremo ad utilizzare è il Bootstrap, che è un framework open source che contiene una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web. Essa contiene modelli di progettazione basati su HTML e CSS, sia per la tipografia, che per le varie componenti dell'interfaccia, come moduli, bottoni e navigazione, e altri componenti dell'interfaccia, così come alcune estensioni opzionali di JavaScript.

1.3 Linee guida per la documentazione dell'interfaccia

Ogni metodo deve essere preceduto da un commento, o più precisamente da una documentazione che riporti l'obiettivo che si vuole. Inoltre, bisogna commentare, giustificare delle decisioni particolari o dei calcoli.

Tipo	Regole sui nomi	Esempi
Packages	Il prefisso di un nome di un pacchetto comincia sempre con una lettera minuscola. Se il nome del package comprende due o più parole, allora il nome completo prevederà che la lettera iniziale della parola nel mezzo sarà maiuscola.	package camera;
Classi	I nomi delle classi iniziano sempre con una lettera maiuscola. Esse devono racchiudere in breve il campo in cui operano. Se il nome è composto da più parole, allora le parole nel mezzo avranno la prima lettera maiuscola.	class InsertCamera;
Interface	I nomi dell'interfaccia sono scelti in modo tale da far capire a quale gestore si riferiscono, quindi il nome sarà composto dal nome del gestore seguito da model.	interface UtenteModel;
Metodi	I nostri metodi rappresentano la funzionalità ad essi associata. Se composti da molteplici parole, la prima lettera della prima parola sarà scritta in minuscolo, mentre la prima lettera di una parola interna sarà maiuscola.	deleteCamera(int numerocamera)
Variabili	I nomi associati alle variabili sono scritti totalmente con caratteri minuscoli. Essi sono	int numerocamera;

	scelti in modo tale da rendere semplice il loro significato ed esplicare significativamente il loro utilizzo.	

Indentazione

L'indentazione deve essere effettuata con un TAB e qualunque sia il linguaggio usato per la produzione di codice, ogni istruzione deve essere opportunamente indentata.

Es.

```
<html>
<head>
</head>
<body>
</body>
</html>
```

Deve essere sostituita da:

```
<html>
        <head>
        </head>
        <body>
        </body>
</html>
```

Questa pratica deve essere usata soprattutto per le istruzioni FOR, IF.

È buona pratica scendere di livello.

Inizializzazione

Inizializzare le variabili locali nel punto in cui sono state dichiarate a meno che il suo valore iniziale non dipenda da un calcolo che occorre eseguire prima.

Posizione

Mettere le dichiarazioni all'inizio dei blocchi. Non aspettare di dichiarare le variabile al loro primo uso: può confondere il programmatore inesperto e impedire la portabilità del codice dentro lo scope. L'unica eccezione a questa regola sono gli indici dei cicli for che in Java possono essere dichiarati nell'istruzione stessa. Evitare dichiarazioni locali che nascondono dichiarazioni a più alto livello. Ad esempio, non dichiarare una variabile con lo stesso nome in un blocco interno.

Parentesi

A prescindere dalle istruzioni che seguono un IF, è necessario, laddove ci fosse anche una sola istruzione, riportare il blocco di istruzioni tra parentesi graffe.

Ogni tag di apertura deve essere necessariamente seguito dall'apposito tag di chiusura (eccetto i tag self-closing).

Una convenzione importante, per quanto riguarda l'inserimento di numeri o di valori costanti, è quella di non usare una codifica fissa (hard coding) che è fortemente sconsigliata ma di associare sempre il valore ad una variabile o semplicemente definire una macro che può essere richiamata da eventi ed essere parametrizzata. In questo modo si facilita la modifica, sostituendo solo il valore della variabile o macro, in un unico posto.

1.4 Design Pattern

Bridge Pattern

Si tratta di un pattern strutturale basato su oggetti che viene utilizzato per disaccoppiare dei componenti software. Il pattern in questione si presta particolarmente al sistema proposto in quanto è stato progettato per garantire il disaccoppiamento tra le varie classi, nascondendo l'implementazione delle stesse, permettendo ad una classe client (che utilizza i servizi un'altra classe), di non essere legata strettamente ad essa, ma invece di utilizzarne i servizi in maniera "anonima". L'utilizzo proposto nell'implementazione del nostro sistema è quella di garantire una separazione tra i livelli di logica e di storage, cioè tra l'implementazione della logica di controllo e l'implementazione delle classi che si occuperanno di dialogare con il database. In questo modo l'implementazione del database diventa irrilevante ai fini d'uso del sistema, in quanto nascosta dietro un'astrazione generica, ed il sistema implementato si avvale dei servizi offerti dalle classi preposte alla comunicazione con il database in maniera trasparente.

Facade Pattern

Si tratta di un pattern strutturale che viene utilizzato per nascondere la complessità di un sottosistema e ridurre l'accoppiamento tra classi client e semplificare l'utilizzo del sottosistema da parte dei client stessi. L'utilizzo di questo pattern prevede di esporre una interfaccia che rappresenti l'intero sottosistema considerato, in maniera tale da fornire una visione più efficiente (dal punto di vista dello sviluppatore) di effettuare procedure possibilmente complesse. I sottosistemi più complessi che compongono la piattaforma MyHotel verranno nascosti dietro un'interfaccia facade in modo tale da semplificarne l'uso.

MVC Pattern

Essendo la piattaforma MyHotel un sistema pensato per il web, viene naturale utilizzare il pattern MVC. Il pattern permette la separazione della logica di business dallo stato e dalla visualizzazione dello stesso. Si ha così una divisione in tre settori diversi della piattaforma, che ben si adatta alla struttura del web. La view viene intesa come l'interfaccia utente, che viene eseguita sulla macchina client nel browser. La logica di business rappresenta i controller, eseguiti sul server. Dato che la piattaforma viene implementata in linguaggio Java, c'è l'ulteriore vantaggio dell'utilizzo della tecnologia Servlet/JSP, che sostanzialmente implementano il pattern MVC, rendendo molto più semplice l'implementazione del sistema.

2.Packages

Di seguito si elencano i packages presenti all'interno del sistema:

1. **connessione**: package la cui classe si occupa dell'accesso al database.
2. **camera**: package che contiene le classi che creano ed operano sulle camere.
3. **carrello**: package che contiene le classi che creano ed operano sul carrello.
4. **prenotazione**: package che contiene le classi che creano ed operano sulle prenotazioni.
5. **servizio**: package che contiene le classi che creano ed operano sui servizi.

6. **utente**: package che contiene le classi che creano ed operano sugli utenti.

3.Definizione interfacce in OCL

Nome Classe	GestoreUtenti
Descrizione	Rappresenta il gestore degli utenti e si occupa di funzionalità legate agli utenti come: login e logout, registrazione, visualizzazione, modifica ed eliminazione del profilo.
Pre-condizioni	context GestoreUtenti::login(email,password); pre:email!=null && password!=null context GestoreUtenti::insertUtente(email,password, nome,cognome,ruolo,datanascita); pre:email!=null && password!=null && nome!=null && cognome!=null && ruolo!=null && datanascita!=null context GestoreUtenti::modifiyUtente(email,password ,nome,cognome,datanascita); pre:email!=null && password!=null && nome!=null && cognome!=null && ruolo!=null && datanascita!=null context GestoreUtenti::deleteUtente(email); pre:email!=null context

	GestoreUtenti::getUtente(email); pre:email!=null context GestoreUtenti::checkEmail(email); pre:email!=null context GestoreUtenti::getUtenti();
Post-condizioni	context GestoreUtenti::getUtente(email); post:Utente!=null || Utente==null context GestoreUtenti::getUtenti(); post:Utenti!=null || Utente==null context GestoreUtenti::checkEmail(email); post:true || false
Invarianti	

Nome	GestoreCamere
Descrizione	Rappresenta il gestore delle camere e si occupa di operazioni quali l'aggiunta di una camera, la modifica e l'eliminazione di una camera.
Pre-Condizioni	context GestoreCamere::insertCamera(numeroCamera, tipologia,immagine,prezzo,descrizione);

	pre:numeroCamera!=null && tipologia!=null && immagine!=null && prezzo!=null && descrizione!=null context GestoreCamere::modifyCamera(numeroCamera,tipologia,immagine,prezzo,descrizione); pre:numeroCamera!=null && tipologia!=null && immagine!=null && prezzo!=null && descrizione!=null context GestoreCamere::deleteCamera(numeroCamera) pre:numeroCamera!=null context GestoreCamere::checkNumero(numeroCamera) pre:numeroCamera!=null
Post-Condizioni	context GestoreCamere::checkNumero(numeroCamera) post:true || false
Invarianti	

Nome	Gestore Prenotazioni
Descrizione	Questa classe gestisce la logica applicativa relativa alle prenotazioni, in particolare si occupa di aggiungere, rimuovere o visualizzare

	una prenotazione.

Pre-Condizioni	context GestorePrenotazioni::insertPrenotazione(idprenotazione,email,numeroCamera,totale,dataInizio,dataFine);
	pre:idprenotazione!=null && email!=null && numeroCamera!=null && (totale!=null && totale>0) && dataInizio!=null && dataFine!=null
	
	context GestorePrenotazioni::deletePrenotazione(idPrenotazione);
	pre:idPrenotazione!=null
	
	context GestorePrenotazioni::Prenotazioni getPrenotazione(idPrenotazione)
	pre:idPrenotazione!=null
	
	context GestorePrenotazioni::Prenotazioni getPrenotazioniUtente(email);
	pre:email!=null
	
	context GestorePrenotazioni::getPrenotazioni();
	
	context GestorePrenotazioni::checkDisponibilità(numeroCamera, dataInizio, dataFine);
	pre:numeroCamera!=null && dataInizio!=null && dataFine!=null
	
	context GestorePrenotazioni::filtraPrenotazioni(periodo, order, totalemin, totalemax);
	pre:periodo!=null && order!=null && totalemin!=null && totalemax!=null

Post-Condizioni	context GestorePrenotazioni::Prenotazioni

	getPrenotazioniUtente(email);
	post Prenotazioni==null || Prenotazioni!=null
	
	context
	GestorePrenotazioni::Prenotazioni
	getPrenotazioni(email);
	post Prenotazioni==null || Prenotazioni!=null
	
	context
	GestorePrenotazioni::Prenotazioni
	getPrenotazione(idPrenotazione);
	post: Prenotazioni==null || Prenotazioni!=null
	
	context
	GestorePrenotazioni::checkDisponibilità(numeroCamera, dataInizio, dataFine);
	post: true || false
	
	context
	GestorePrenotazioni::filtraPrenotazioni(periodo, order, totalemin, totalemax);
	post: Prenotazioni==null || Prenotazioni!=null

Invarianti	

Nome	VisualizzatoreCamere
Descrizione	Rappresenta la classe che permette la visualizzazione dei dettagli di ogni singola camera presente all'interno del database.
Pre-Condizioni	context VisualizzatoreCamere:: filtraCamere(tipologia, min, max, order); pre:(tipologia!=null ||

	tipologia==null)&&(min!=null || min==null)&&(max!=null || max==null)&&(order!=null || order==null) context VisualizzatoreCamere:: getCamere() context VisualizzatoreCamere:: getCamera(numeroCamera); pre:numeroCamera!=null
Post-Condizioni	context VisualizzatoreCamere:: Camere getCamera(numeroCamera); post: Camera!=null || Camera==null context VisualizzatoreCamere:: Camere filtraCamere(tipologia, min, max, order); post: Camera!=null | Camera==null context VisualizzatoreCamere::getCamere(); post: Camera!=null || Camera==null
Invarianti	

Nome	GestoreCarrello
Descrizione	Rappresenta la classe che permette la visualizzazione delle camere aggiunte al carrello dall'utente.
Pre-Condizioni	context GestoreCarrello::insertCamera(email, numeroCamera, dataInizio, dataFine, totale);

	pre:email!=null &&
	numeroCamera!=null &&
	dataInizio!=null && dataFine!=null
	&& totale!=null
	
	context
	GestoreCarrello::deleteCamera(e
	mail, numeroCamera);
	pre: email!=null &&
	numeroCamera!=null
	
	context
	GestoreCarrello::getIdCamereCar
	rello(email);
	pre:email!=null
	
	context
	GestoreCarrello::getCarrelloUtent
	e(email);
	pre:email!=null
	
	context
	GestoreCarrello::Camere
	emptyCarrello(email)
	pre:email!=null

Post-Condizioni	context
	GestoreCarrello::Camere
	emptyCarrello(email)
	post:Prenotazione==null
	
	context
	GestoreCarrello::getIdCamereCar
	rello(email);
	post: id==null || id!=null
	
	context
	GestoreCarrello::getCarrelloUtent
	e(email);
	post: Prenotazioni!=null ||
	Prenotazioni==null

Invarianti	

Nome	GestoreServizi
Descrizione	Rappresenta la classe che permette di gestire tutti i servizi relativi alle camere dell'hotel.

Pre-Condizioni	context GestoreServizi::getServizi();
	
	context GestoreServizi::getServiziCamera (numeroCamera);
	pre:numeroCamera!=null
	
	context GestoreServizi::insertServizioCam era(nomeServizio, numeroCamera);
	pre:nomeServizio!=null && numeroCamera!=null

Post-Condizioni	context GestoreServizi::getServizi(); post: Servizi==null || Servizi!=null context GestoreServizi::getServiziCamera (numeroCamera); post Servizi==nul || Servizi!=null
Invarianti	