MyHotel
Relazione per "Programmazione ad Oggetti"

Emilio Dettori, Alberto Serluca, Giacomo Pili

30 maggio 2016

Indice

Capitolo 1

Analisi

1.1 Requisiti

Il gruppo si pone come obiettivo quello di realizzare un'applicazione adatta a gestire un hotel in alcuni dei suoi aspetti principali. Gestire un hotel è un tipo di attività molto spesso complessa, data la mole di informazioni di cui tenere conto, per questo si ricorre molto spesso a software gestionali che aiutino a mantenere sotto controllo tutti gli aspetti fondamentali dell'azienda.

1.2 Funzionalità

Il software sarà in grado di permettere la creazione di un proprio Hotel personalizzato, e questo comprende: la possibilità di aggiungere,rimuovere, editare le camere dell'albergo, ed opzionalmente decidere se l'hotel possieda e possa gestire anche uno o piu bar, in modo da poter personalizzare al massimo la propria attività. Oltre a questi aspetti base, sarà fondamentale la possibilità di gestione delle prenotazioni, e di managing dello staff che compone l'attività ed utilizza l'applicazione.

Il software esporrà le seguenti funzionalità:

- Gestione delle prenotazioni
- Gestione del piano camere
- Gestione del bar e delle giacenze
- Calcolo del conto al check-out
- Gestione degli account dello staff

1.3 Modello del dominio

MyHotel dovrà essere in grado di far gestire a uno o più utenti un hotel. L'hotel sarà composto da un numero variabile di camere e un numero variabile di bar. In ogni camera sarà possibile inserire una o più prenotazioni, che a loro volta saranno composte da uno o più ospiti. I suddetti ospiti potranno acquistare dei prodotti dai bar. Una volta terminato il soggiorno, il sistema comunicherà agli utenti il prezzo totale dovuto dagli ospiti.

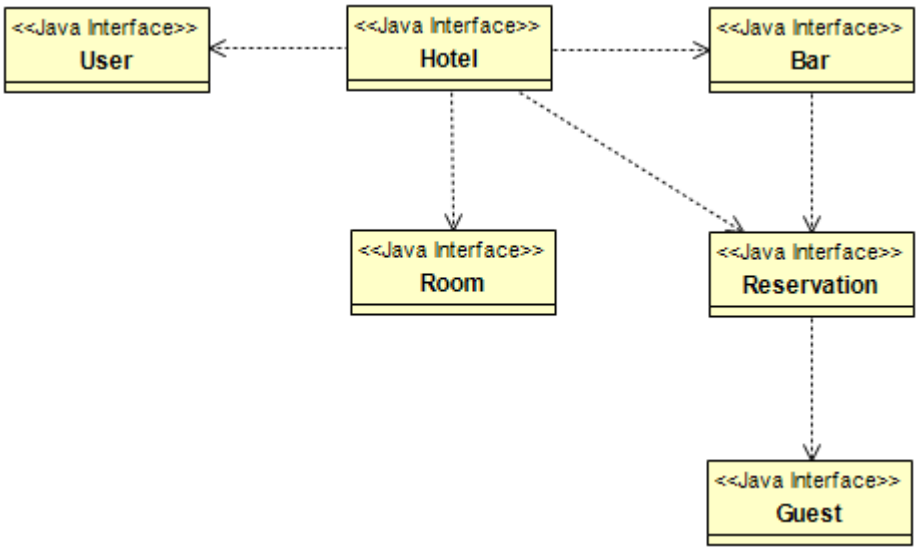

Figura 1.1: Rappresentazione del modello del dominio.

Capitolo 2

Design

2.1 Architettura

La progettazione architetturale è stata costruita tenendo in mente i seguenti obbiettivi: flessibilità, scalabilità, e facilità di manutenzione.

Per rendere possibile il conseguimento di questi obbiettivi ci siamo avvalsi del pattern MVC (Model, View, Controller). Questo pattern prevede la suddivisione del programma in tre macro parti: l'utente interagisce con la View; il controller gestisce gli input ricevuti dalla view e modifica il model di conseguenza.

Il compito di gestire la comunicazione tra view e controller è stato affidato al pattern **Observer** che permette di ottenere una separazione tra le entità che gestiscono gli aspetti di presentazione e quelli di controllo. Il controller è dotato di metodi che gli permettono di recuperare informazioni dalla View, e di metodi che gli permettono di manipolare il Model. La view fornisce tramite le sue interfacce dei metodi pratici che la modifichino. Grazie a questo pattern il programma è facilmente modificabile, poichè modifiche alla view non comportano cambiamenti necessari alla parte di controllo.

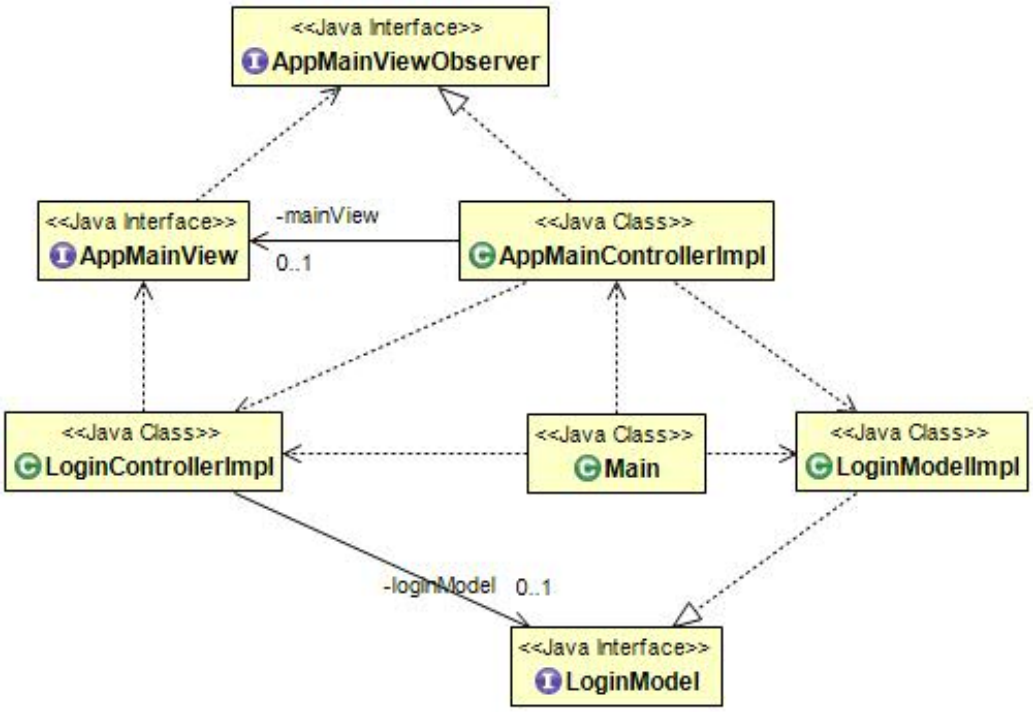

Figura 2.1: Questa è l'architettura MVC utilizzata nell'applicazione. In particolare viene mostrato l'utilizzo di una view ed un controller principale. Quest'ultimo viene utilizzato da parte degli altri controller per accedere e modificare la finestra principale del programma (A titolo d'esempio il controller del Login può impostare, nella view principale, il testo che mostra quale sia l'utente loggato al momento). Si noti che LoginModel non rappresenta tutte le parti del modello solo il suo entry point.

2.2 Design dettagliato

L'applicazione è stata suddivisa nelle tre parti principali che compongono l'MVC, ovvero Model View e Controller. Non disponendo di nessun dominio abbiamo deciso, su consiglio delle linee guida, di avvalerci del prefisso dell'Università **it.unibo** a cui abbiamo aggiunto il nome del nostro gruppo di lavoro e il nome dell'applicazione **myhoteluniboteam.myhotel**.

Il dominio così ottenuto è stato utilizzato per denominare i vari package principali, aggiungendo alla fine l'identificativo del package stesso, a titolo d'esempio: **it.unibo.myhoteluniboteam.myhotel.controller**

Il build path è stato strutturato prevedendo i sorgenti all'interno di una cartella **src** suddivisa in **src/main** che contiene i sorgenti principali, ed

src/test che contiene i file inerenti ai test JUnit dell'applicazione. Sono presenti inoltre le cartelle *lib* e *res* che contengono rispettivamente le librerie utilizzate, e le varie risorse, come per esempio il logo del programma o il file di demo dell'hotel.

2.2.1 Model - a cura di Emilio Dettori

Il Model descrive la realtà di un hotel, ed elabora i dati dello stesso. Permette quindi di: gestire uno o più utenti, i quali hanno diversi diritti di accesso; permette di aggiungere e rimuovere dall'Hotel camere e bar; permette di creare nuove prenotazioni. Ogni prenotazione, inoltre, sarà composta da un certo numero di ospiti, i quali potranno acquistare delle consumazioni in uno dei bar dell'hotel. Al momento del calcolo del conto, oltre al costo della stanza, sarà quindi aggiunto il costo degli extra acquistati. Il Model è composto da diverse interfacce. La principale è hotel; quest'ultima contiene i metodi necessari a creare ed a modificare la struttura base. Sono poi presenti le interfacce Bar, Guest, Reservation, Room, ed User, le quali modellano uno specifico aspetto implementativo dell'hotel.

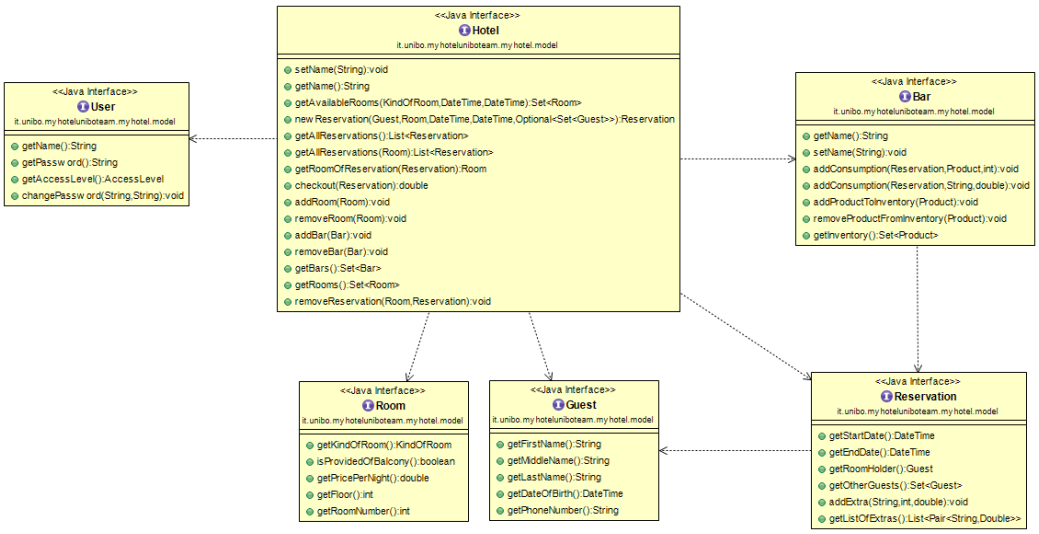

Figura 2.2: Architettura generale del Model, in cui è possibile vedere i principali metodi implentati dalle interfacce.

2.2.2 Controller - a cura di Alberto Serluca

Il controller gestisce le interazioni tra la view ed il model, si fa carico di ricevere i vari input dalla GUI e di modificare il model di conseguenza, ma anche

di gestire i cambiamenti del model comunicandoli alla view. Per permettere al controller di essere piu flessibile, il compito di mantenere memorizzati i dati importanti è stato affidato ad una classe chiamata **AppDataManager**.

Questa classe possiede dei metodi che le permettono di modificare e restituire delle strutture dati e dei riferimenti fondamentali per il programma. In particolare viene memorizzato il set di utenti e il riferimento all'istanza di HotelModel ovvero il modello principale che rappresenta l'applicazione. Questa classe espone i metodi fondamentali al programma per caricare e salvare lo stato dell'albergo e la lista di account dello staff. Per permettere una maggiore flessibilità nelle strategie correnti e future di salvataggio e caricamento, è stato utilizzato il pattern **Strategy**.

Durante lo sviluppo è venuta alla luce l'importanza di possedere una sola ed unica istanza di AppDataManager in modo che non vi fossero errori o ridondanze all'interno dell'applicazione, per questo è stato molto utile e pratico il patter **Singleton** che permette di accedere, ed ottenere, una singola istanza immutabile di un certo oggetto.

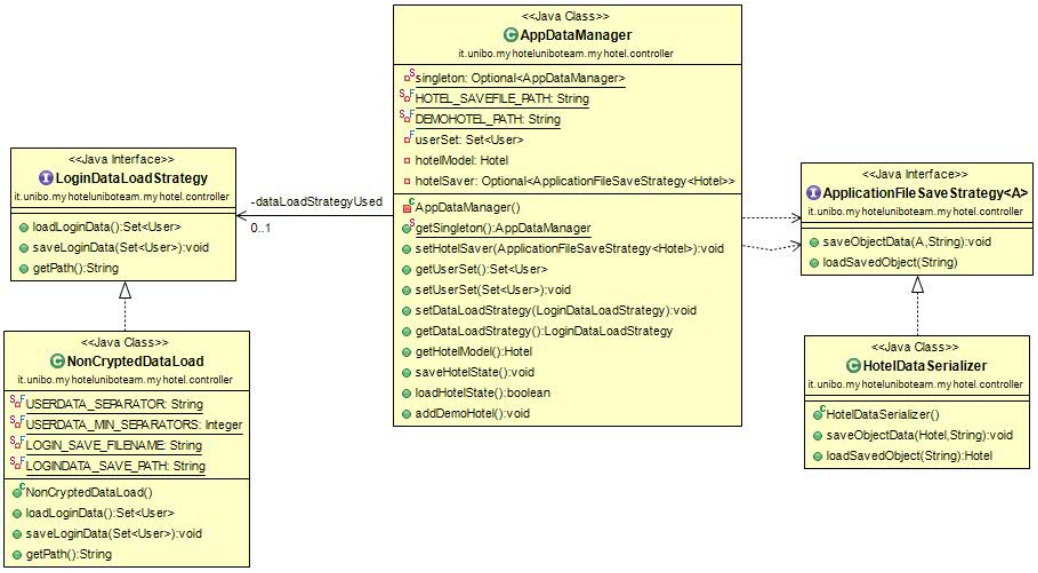

Figura 2.3: Dettaglio di AppDataManager e delle strategie di salvataggio e caricamento dell'Hotel e della lista di utenti.

Una delle parti principali del controller è quella che si occupa di gestire l'interfaccia per la registrazione, l'eliminazione o la modifica di una prenotazione. In questo caso, essendo la gui per la gestione delle prenotazioni divisa in due parti, sono stati implementati due controller separati. Il primo con-

troller **BookingsController** si fa carico di gestire la lista di prenotazioni attive al momento e, su richiesta della view di rimuovere una prenotazione, previa selezione dalla lista sopracitata. Per agevolare l'operazione ad un possibile utente, è stata implementata la funzione di ricerca, che semplicemente filtra i risultati in base alle varie keyword cercate.

Per aggiungere un po' di 'zucchero sintattico', sono state usate funzioni lambda per definire i Comparator necessari all'ordinamento delle liste di prenotazioni.

Invece di creare due controller separati che gestissero uno la parte di Aggiunta ed uno la parte di Modifica delle prenotazioni, si è preferito realizzare un singolo controller **BookingsFormController**. La distinzione tra le funzioni da eseguire (se Aggiungi o Modifica) è stata possibile grazie all'utilizzo di un Optional passato come parametro al controller.

A seguito della pressione dei tasti 'Modifica' o 'Aggiungi', il controller **BookingsController** cattura la prenotazione correntemente selezionata (se ve ne è alcuna) e, dopo aver creato l'istanza del controller **BookingsFormController** vi passa come argomento un Optional col valore della prenotazione precedentemente selezionata. A questo punto viene creato l'oggetto che rappresenta la GUI e **AppMainController** viene richiamato utilizzando il metodo **setCurrentPanel()** , settando così la GUI appena creata, come GUI principale.

In caso **BookingsFormController** rilevasse che l'Optional di prenotazione passato al costruttore è presente, predispone l'interfaccia come 'GUI di modifica', e predispone i campi dei vari componenti coi valori della prenotazione precedentemente selezionata. Se l'Optional invece risultasse vuoto significherebbe che abbiamo a che fare con la 'GUI di Aggiunta'.

Infine quando si effettuano le varie operazioni di modifica, aggiunta o rimozione, viene richiamato **AppDataManager.SaveHotelState()** che salva (per ora su disco) le operazioni effettuate.

A seguito un diagramma UML rappresentativo delle interazioni fra le classi.

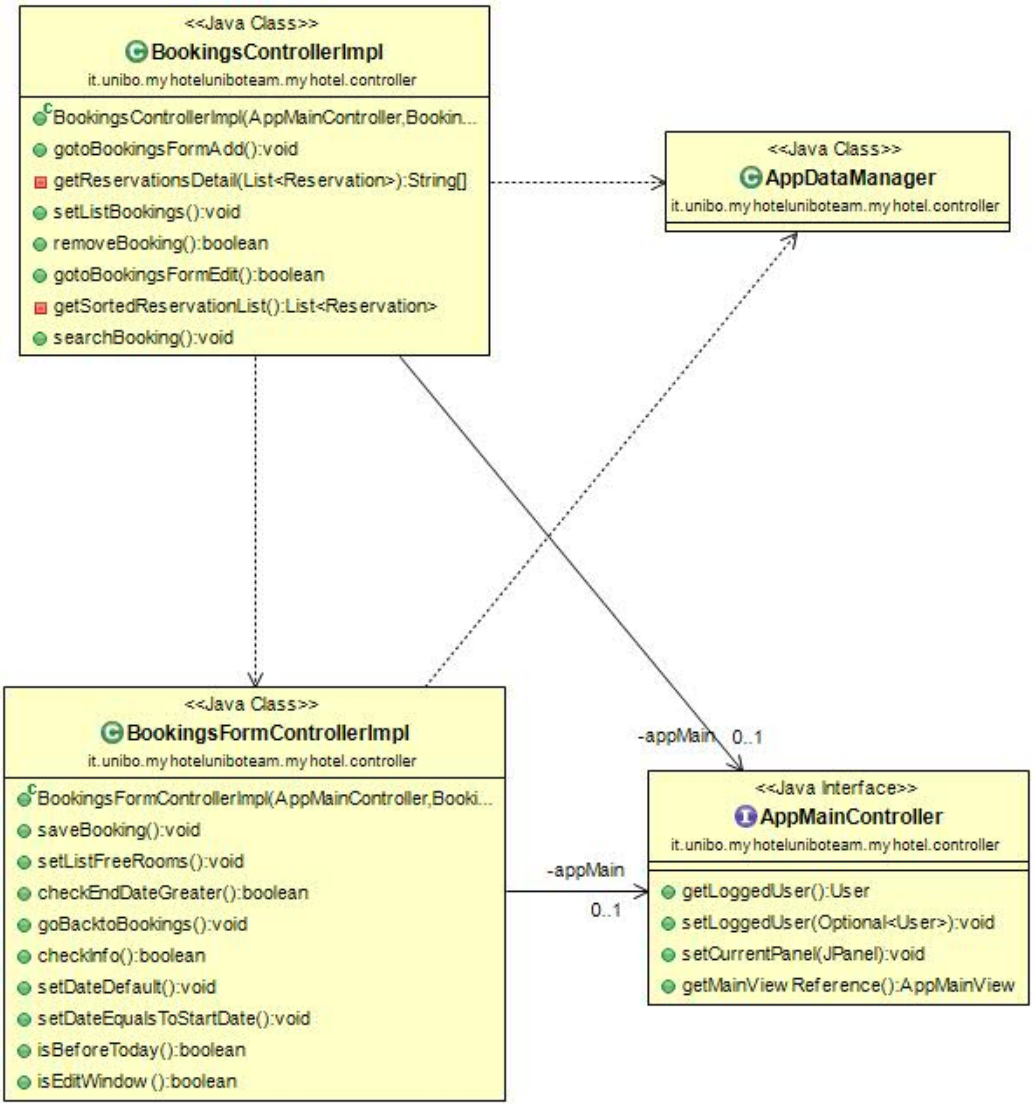

Figura 2.4: Le classi **BookingsController** e **BookingsFormController** e le loro interazioni con **AppMainController** e **AppDataManager**

2.2.3 View - a cura di Alberto Serluca

Per avere una GUI che fosse facilmente estendibile e per mostrare l'applicazione come una singola finestra, è stata creata una classe che gestisce quella che è la finestra e gli elementi principali del programma.

In particolare, oltre a mostrare il logo del programma ed impostare l'icona dell'applicazione, **AppMainView** si occupa di creare il *JFrame* principale, di conseguenza essa espone un metodo chiamato **setCurrentPanel()** che permette di impostare quale sia il pannello correntemente mostrato all'utente.

Oltre a questa funzionalità, la GUI principale si occupa dell'implementare i metodi per l'aggiunta e rimozione dei bottoni *BACK*, *LOGOUT* e della label che mostra quale sia l'utente correntemente loggato.

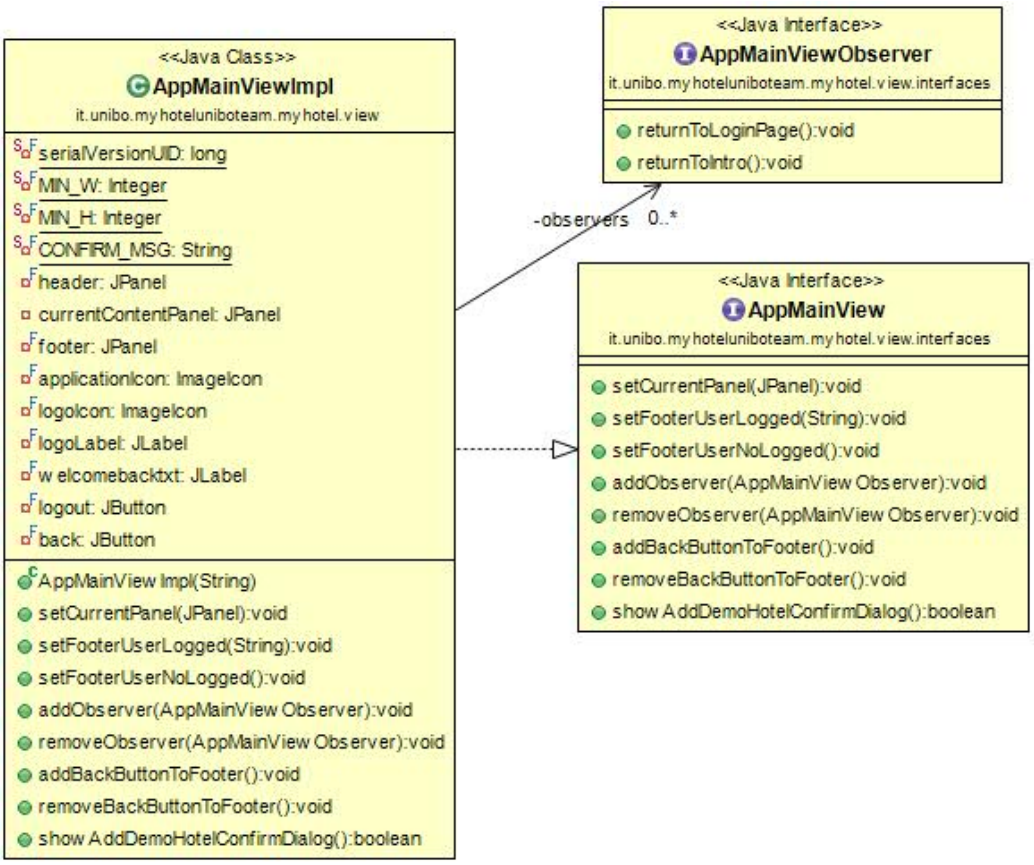

Figura 2.5: Diagramma UML della classe AppMainView che gestisce la finestra principale del programma.

Visto che la **AppMainView** rappresenta il *JFrame* principale, tutte le altre *GUI* sono state strutturate come semplici pannelli add-on.

Una delle GUI piu importanti è **HotelView**. Essa permette di modificare da zero la struttura dell'Hotel, aggiungendo, rimuovendo e modificando le varie camere che lo compongono ed inoltre permette di impostare o meno la presenza di un bar nell'Hotel. L'interfaccia in questione utilizza un *BoxLayout* al cui interno sono disposti diversi pannelli che presentano le varie funzionalità.

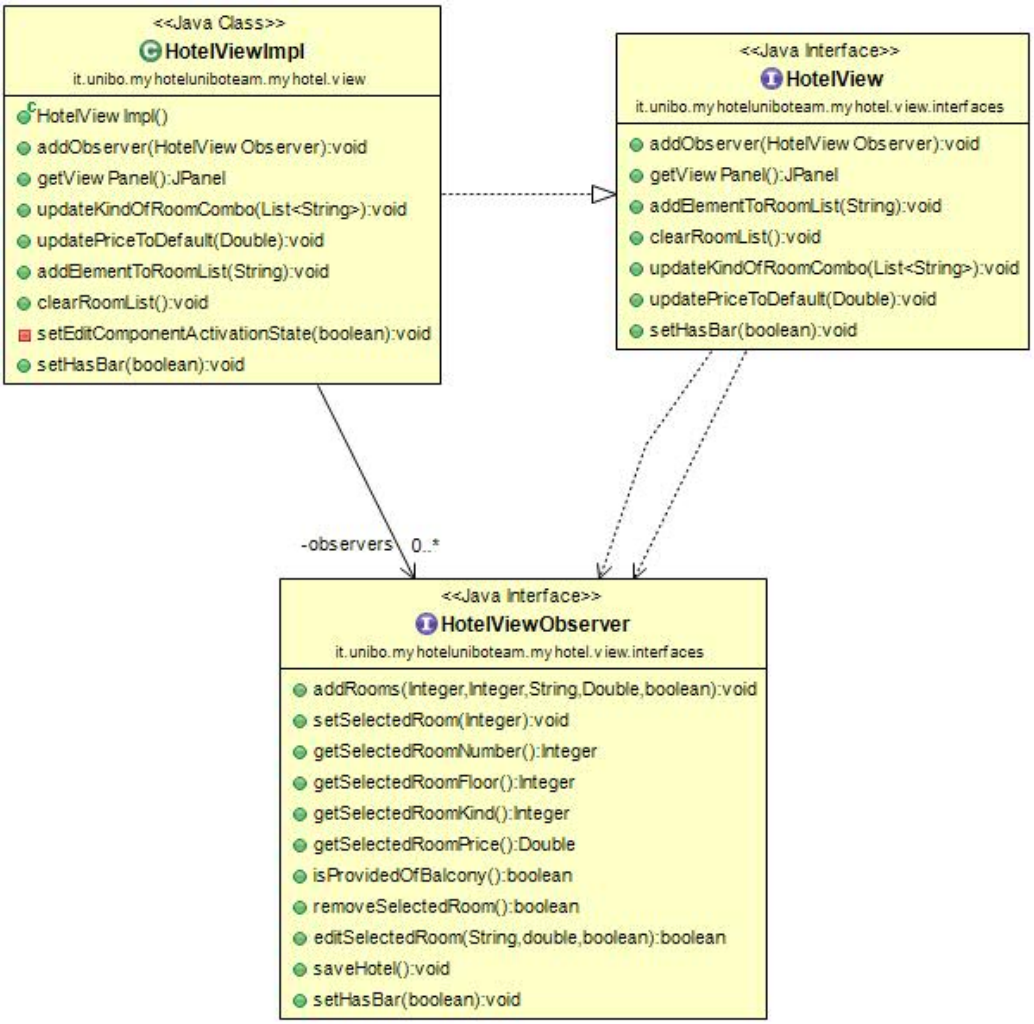

Figura 2.6: Diagramma UML della GUI per la gestione del piano camere e del bar.

2.2.4 View - a cura di Giacomo Pili

Come precedentemente citato, la view si basa sulla classe chiamata *AppMainViewImpl* che genera un frame suddiviso in 3 pannelli: header (testata), content (contenuto) e footer (piè di pagina).

Ogni classe crea un pannello "content", il cuore di ogni finestra, pannelli che durante navigazione si succedono. Come pattern creazionale è stato adottato il *Simple factory* che facilita l'aggiunta e la modifica di nuove finestre riducendo il tutto alla creazione di un singolo pannello.

Si è deciso di seguire questo design pattern dopo aver valutato che l' *Abstract factory* era troppo dispendioso, in quanto doveva rigenerare l'intero frame ogni volta che si cambiava contesto.

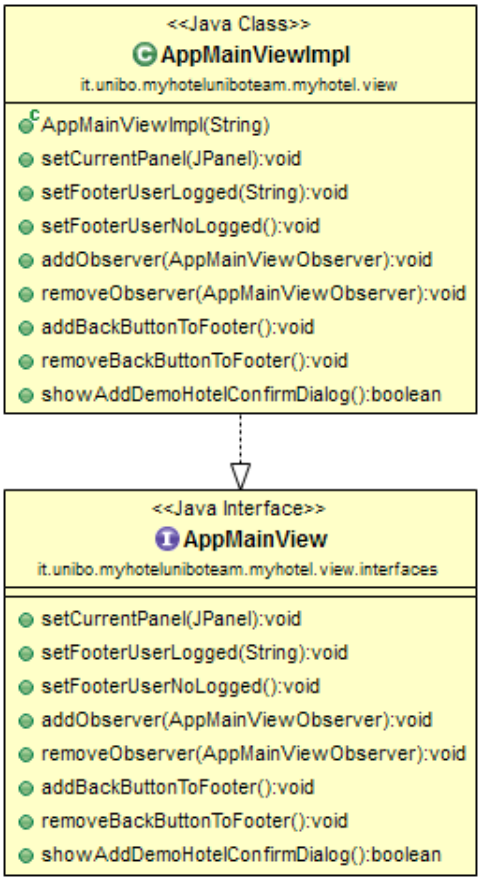

Figura 2.7: UML in particolare della classe AppMainView e della sua relativa interfaccia

Ogni classe della view ha una sua interfaccia che consente, insieme all'interfaccia Observer, al controller di far comunicare Model e View. Grazie

al pattern creazionale Observer il controller fornisce alla view i metodi per aggiornare la GUI creando un incapsulamento tra le parti e permettendo un lavoro in parallelo.

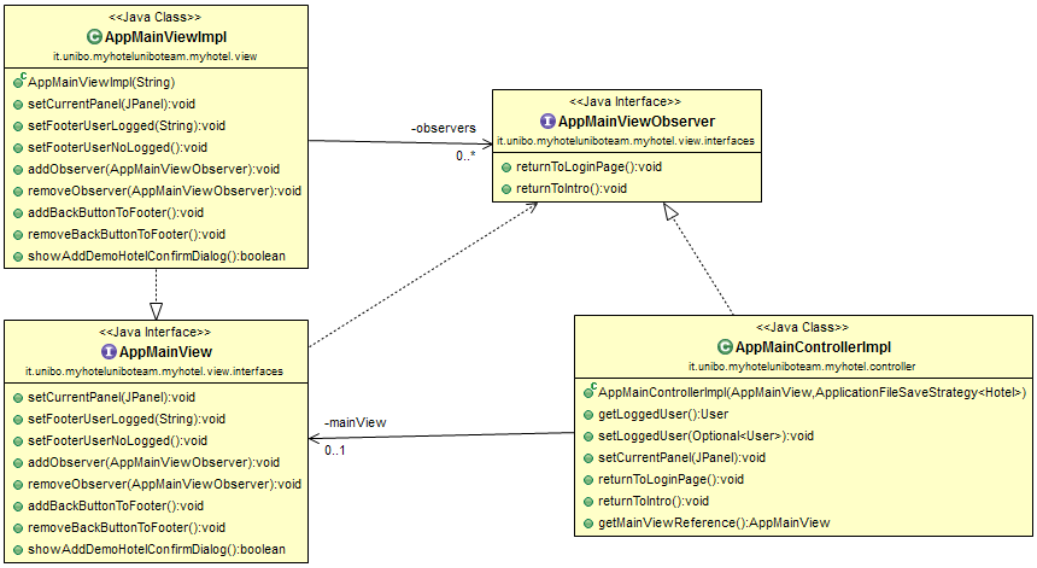

Figura 2.8: Interazione AppMainView-Controller

L'aiuto del collega Alberto Serluca nella realizzazione delle classi AppMainViewImpl, LoginViewImpl, HotelViewImpl e IntroViewImpl è stato fondamentale.

Per offrire una maggiore personalizzazione dell'interfaccia ed arricchirne il layout sono state estese le classi JButton e JPanel, andando a creare due nuove classi *MyHotelJButton* e *MyHotelJPanel*.

Grazie a queste classi è facilmente impostabile un color-scheme per la GUI del programma.

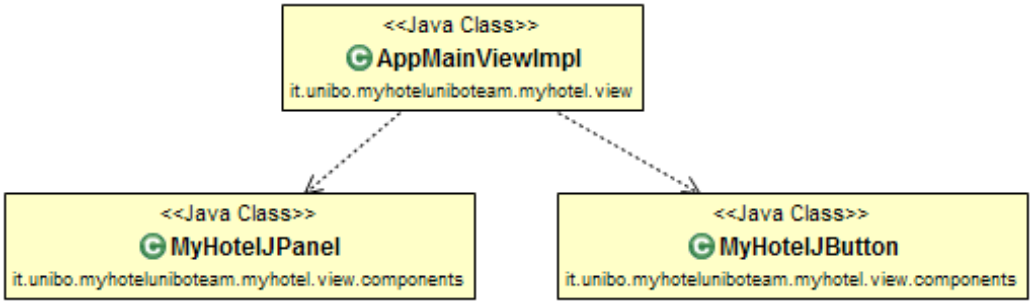

Figura 2.9: Classi estese da JPanel e JButton.

Capitolo 3

Sviluppo

3.1 Testing automatizzato

Il testing è contenuto in un package apposito, chiamato test. In questo package è presente una classe che contiene un test automatizzato su tutti i metodi del model. Il test si assicura che le istanze degli oggetti siano create correttamente, che i metodi forniscano i risultati aspettati, e che le strutture dati siano utilizzate in maniera corretta.

3.2 Metodologia di lavoro

All'inizio, una delle nostre principali preoccupazioni è stata definire un'equa suddivisione del lavoro da svolgere. Abbiamo concluso che ci saremmo divisi i compiti così:

- **Model:** Emilio Dettori
- **Controller:** Alberto Serluca
- **View:** Giacomo Pili e Alberto Serluca

In particolare Alberto Serluca si è occupato del *Controller* nella sua interezza e della *View* nelle classi che comprendono:

- AppMainView
- LoginView
- IntroView
- HotelView
- Interfacce delle GUI sopraelencate e relative Interfacce Observer.

Giacomo Pili invece si è occupato dello sviluppo delle rimanenti classi dell'interfaccia. Insieme si è concordato di utilizzare il pattern **Observer** come 'mezzo' di comunicazione tra view e controller. In fase di design si è erroneamente pensato alle varie *GUI* come *JFrame* separati. Convenendo più tardi che l'interfaccia dell'applicazione sarebbe stata molto piu chiara se presentata su finestra singola, si è integrato il codice presente per adattarlo alle nuove esigenze. Durante lo sviluppo, in caso di necessità, di modifiche minime *(quali modifiche a testi, getter, o dialoghi)* da parte di uno di noi, alle classi dell'altro, abbiamo convenuto fosse piu semplice e molto piu veloce provvedere noi stessi alle suddette modifiche, permettendo così una velocità di sviluppo maggiore delle varie GUI.

Lo sviluppo del progetto è stato supportato dall'utilizzo del *DVCS Mercurial*, utilizzando come host del repository *Bitbucket*. Serluca si è occupato del repository managing, ed insieme agli altri componenti del gruppo si è optato per lavorare su branch singolo *'default'*.

In caso di HotFix necessari o bug imprevisti che non permettevano l'esecuzione stabile del programma, sono state aperte branch temporanee per il bugfix, permettendo di continuare, in parallelo, lo sviluppo dell'applicazione, ed il bugfixing.

Una volta giunti ad una release stabile, è stato creato un branch *stable* in cui sono state caricate e taggate con apposito *version number* le versioni stabili dell'applicativo.

Capitolo 4

Commenti finali

4.1 Autovalutazione e lavori futuri

L'impegno e la collaborazione ci hanno permesso di portare a termine questo progetto. Oltre ad aver fatto tesoro di questa esperienza, pensiamo che la cosa più importante che questo progetto ci ha insegnato, e di cui sicuramente terremo conto nei progetti futuri, è l'importanza di definire ruoli e deadline da rispettare, in modo che il lavoro rimanga bilanciato e concreto.

Infine, nonostante non siamo stati in grado di sviluppate tutte le feature opzionali (quali la gestione di un ristorante o la presenza di più di un singolo bar) per mancanza di tempo, rimaniamo confidenti che la nostra metodologia di lavoro ci permetterà di implementarle in futuro senza uno sforzo eccessivo, grazie all'estendibilità e flessibilità del codice da noi prodotto.

4.2 Difficoltà incontrate

All'inizio dello sviluppo siamo rimasti un po' spiazzati dall'utilizzo del DVCS in gruppo, poichè non ci è stato subito chiaro come doverci organizzare e quali fossero le giuste azioni da intraprendere nel caso di problemi con la repository. Potrebbe risultare utile fornire piu esercitazioni sull'utilizzo in gruppo di un DVCS quale Mercurial per esempio o sulla creazione efficace di diagrammi UML rappresentativi.

Appendice A

Primo avvio ed elementi principali

All'avvio dell'applicazione per la prima volta, il programma presenterà una schermata di login, alla pressione del tasto login l'utente verrà avvisato della creazione di un file temporaneo di Login con due utenti pre-impostati:

- admin
- *PSW:* password LIVELLO D'ACCESSO: *ADMIN*
- user
- *PSW:* password LIVELLO D'ACCESSO: *USER*

Inoltre verrà chiesto se caricare un Hotel di Demo oppure avviare l'applicazione vuota. A seconda che l'utente sia Admin o User verrà predisposta una schermata principale differente.

Il menù principale è composto da massimo 5 bottoni. La loro funzione è elencata qui di seguito:

- **Bookings:** questa funzionalità permette di gestire le prenotazioni. Una volta cliccato su Bookings, si aprirà una finestra che permetterà all'utente di aggiungere, modificare, o rimuovere le prenotazioni.
- **Check out:** questa schermata permette di effettuare il check-out. Dopo aver selezionato una prenotazione, sarà sufficiente cliccare su "Check out" per rimuovere la prenotazione ed ottenere il costo totale dovuto dal cliente.
- **Hotel Setup:** qui è possibile modificare la struttura dell'hotel, aggiungendo, modificando, o rimuovendo le camere.

- **Staff Managing:** questa sezione permette di gestire lo staff che utilizza l'applicazione.
- **Bar:** la schermata bar permette, una volta selezionata una camera, di aggiungere o rimuovere delle consumazioni.

Appendice B

Guida utente

L'utilizzo di bookings è semplice, i tre bottoni Add,Edit e Remove permettono di gestire le prenotazioni, la barra di ricerca filtra le prenotazioni esistenti:

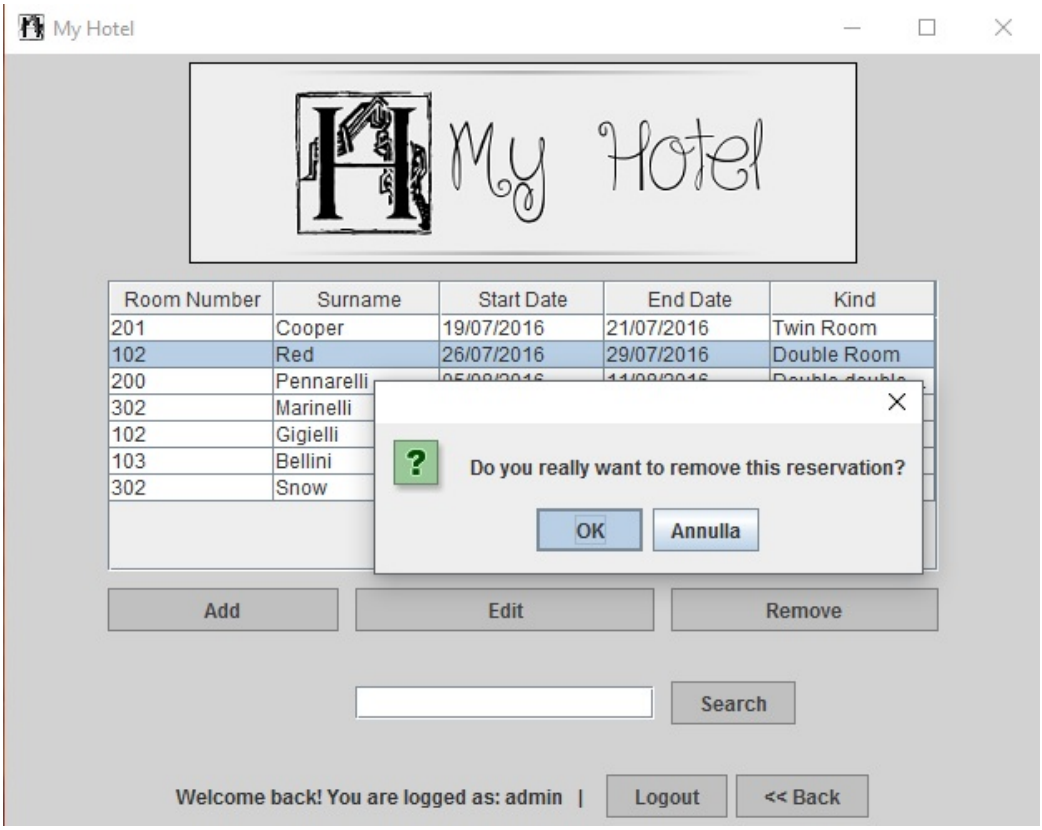

Figura B.1: La schermata principale per la gestione delle prenotazioni.

Quando si accede alla schermata Add o Edit, il programma mostra la lista di camere disponibili a seconda delle date di checkin e checkout selezionate. Se si sta modificando una prenotazione, è possibile lasciare la camera invariata o selezionarne un'altra tra quelle disponibili. Tutti gli altri dati possono essere editati a piacimento.

Le date di checkin devono ovviamente essere postume alle date di checkout ed essere successive alla data odierna.

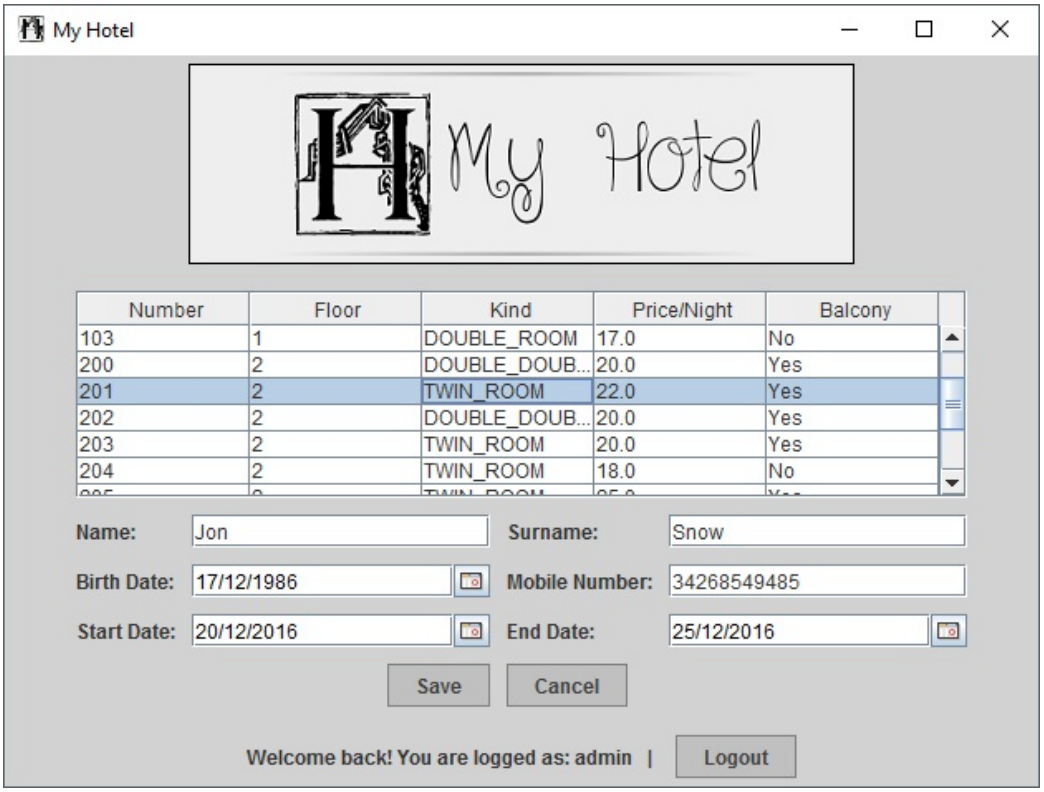

Figura B.2: Gestione di una prenotazione.

La gestione del piano camere è stata strutturata per essere altrettanto semplice, in alto la lista di tutte le camere che compongono l'Hotel. Sotto nei vari box si possono aggiungere in gruppo N camere con certe caratteristiche, e successivamente modificarle, anche una per una, in caso di necessità.

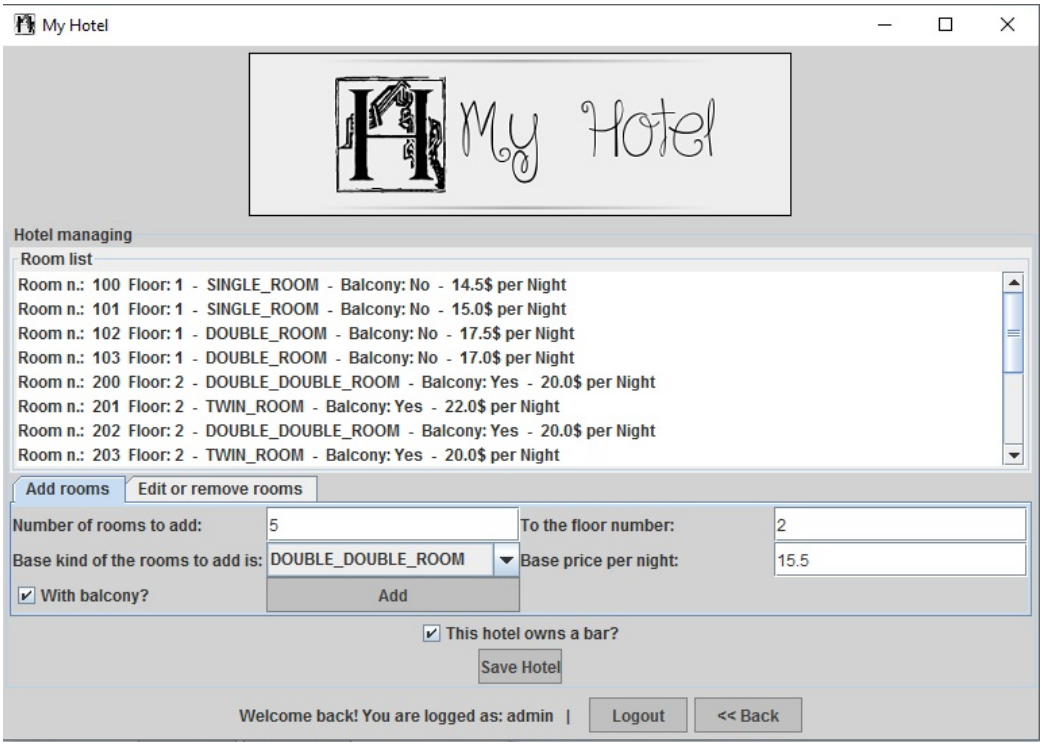

Figura B.3: Gestione del piano camere.

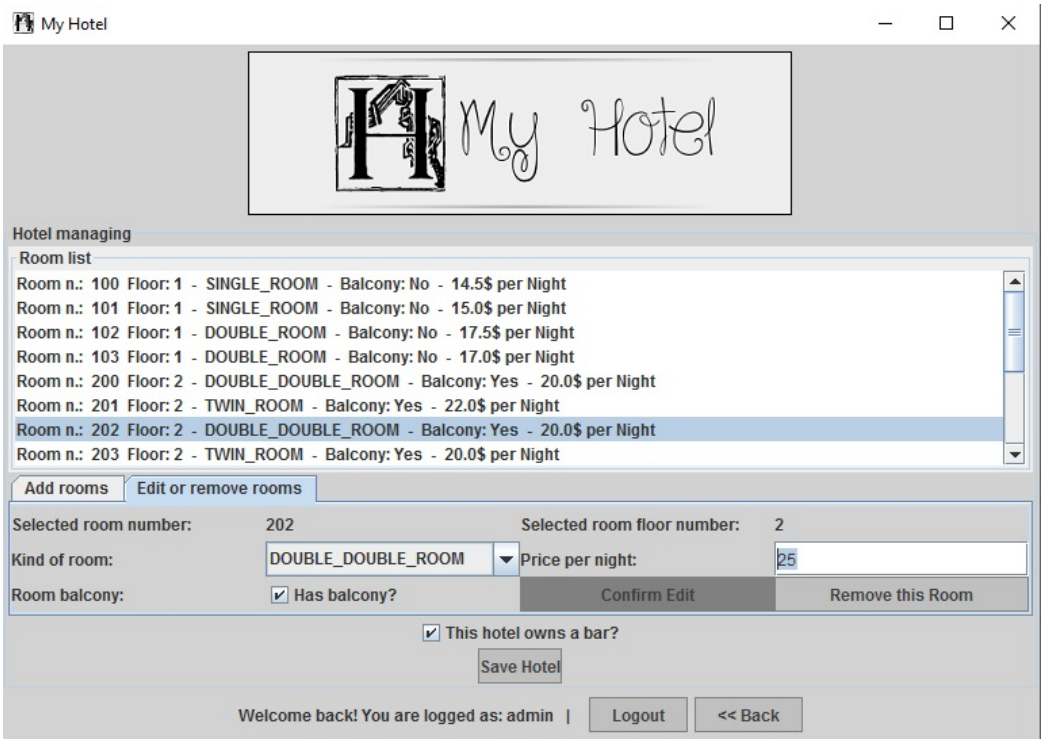

Figura B.4: Modifica delle camere.

Per aggiungere consumazioni extra ad un cliente si può ricorrere alla GUI 'Bar' che sarà disponibile solo se in fase di setup dell'Hotel l'utente confermi la presenza di un Bar nell' Hotel. Per aggiungere un conto basta specificare il numero di camera di una prenotazione attiva. La camera, nel giorno corrente, deve avere una prenotazione attiva perchè sia possibile l'aggiunta di extra.

Questa GUI mostra i vari conti aperti del bar, è possibile aggiungere rimuovere o segnare un conto come 'pagato', una volta pagato un conto, questo verrà aggiunto agli extra della camera in questione.

Si possono visualizzare in dettaglio tutti i prodotti di ogni conto, ed è possibile aggiungerne o rimuoverne dall'inventario del Bar.

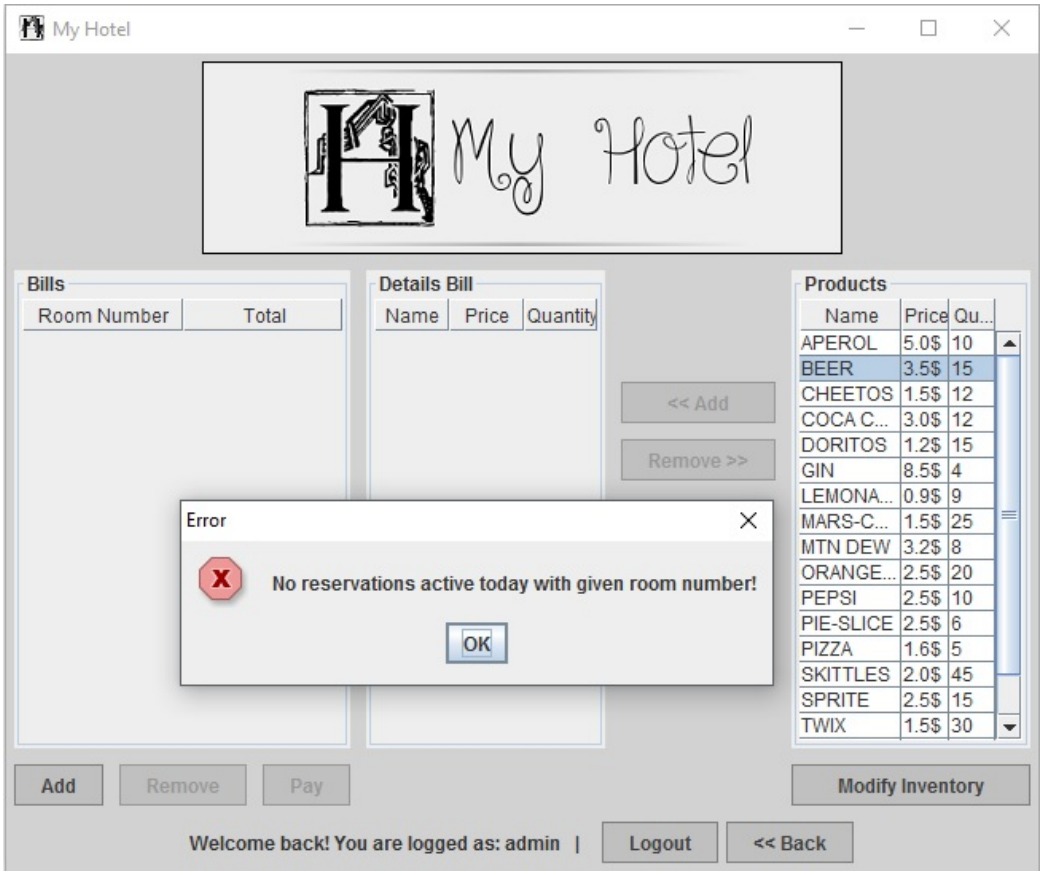

Figura B.5: Interfaccia per la gestione del Bar dell'Hotel.

In fine, si possono gestire i vari account che possono accedere all'applicazione, (questa funzione è ovviamente disponibile sono per chi ha accesso ADMIN). L'aggiunta e modifica sono molto semplici ed immediate. Non si possono aggiungere due utenti uguali ed Username e Password devono avere minimo 3 caratteri di lunghezza.

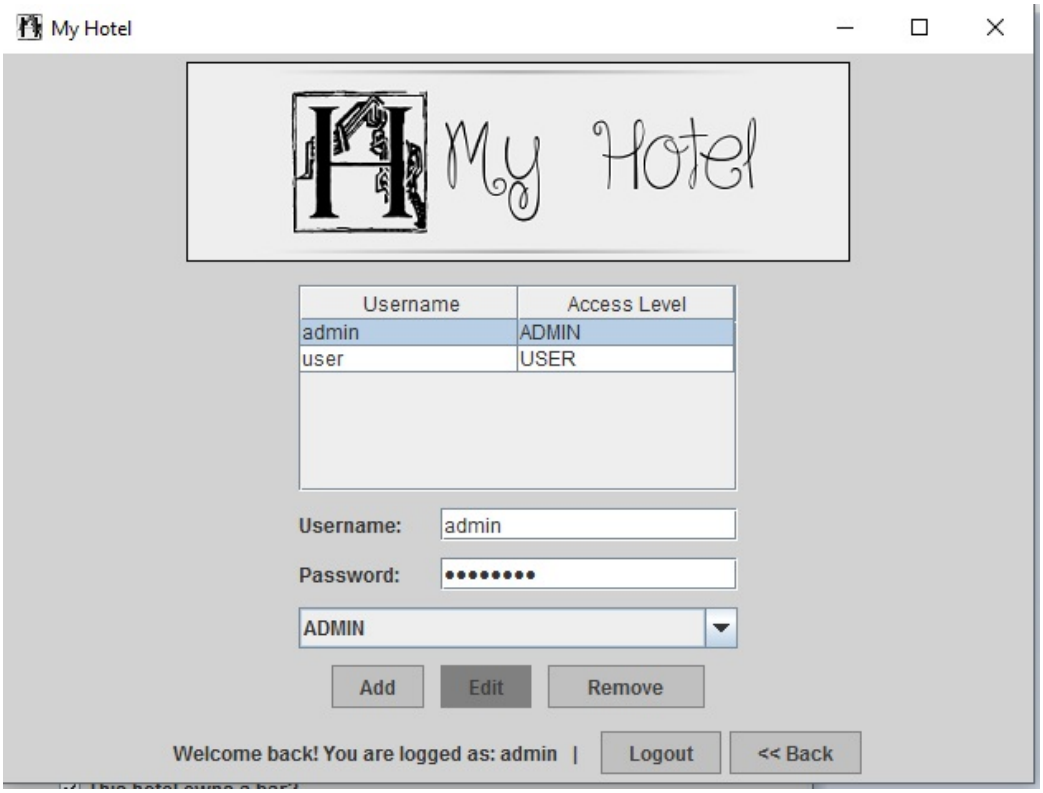

Figura B.6: Gestione dello staff dell'hotel.