

POLITECNICO DI MILANO
SCUOLA DI INGEGNERIA DELL'INFORMAZIONE
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA



Progetto di Ingegneria del Software 2
TravelDream
Parte II: DD
(Design Document)

Responsabile:
Prof. Raffaella Mirandola

Progetto di:	
Piazza Enrico	Matricola n.760240
Polvara Riccardo	Matricola n.817572
Zamperetti Niccolò	Matricola n.814656

ANNO ACCADEMICO 2013-2014

Indice

1	Descrizione Generale	3
1.1	Panoramica	3
1.2	Descrizione dell'architettura	4
2	Progetto dei dati	6
2.1	Modello Entity-Relationship	6
2.2	Ristrutturazione del modello ER	8
2.3	Traduzione verso il modello logico	10
3	Progetto del business tier	13
3.1	Logica applicativa (EJB)	14
4	Progetto del client tier	16
4.1	Progetto della navigazione	16
4.2	Sequence Diagramm	19
4.3	Page Sketch	21
5	Rettifiche	25
5.1	Funzionalità mancanti	25
5.2	Struttura DB	25
5.3	Navigazione delle pagine	26

Capitolo 1

Descrizione Generale

Questo documento fornisce una descrizione della struttura concettuale del sistema che implementeremo, TravelDream, basata sulle specifiche descritte nel documento di analisi dei requisiti, il Requirement Analysis and Specification Document (RASD). Si rivolge principalmente al gruppo di lavoro che sarà incaricato dell'implementazione e della manutenzione del software. Verrà quindi presentata dapprima una panoramica del sistema e della sua architettura, partendo dal modello concettuale dei dati, passando a quello logico su cui si basa lo schema della base di dati che conterrà le informazioni dei clienti e le offerte proposte. Nei capitoli successivi, analizzeremo in dettaglio la progettazione dei vari elementi che compongono l'applicazione, quali le funzionalità, la struttura a livelli, le tecnologie scelte per l'implementazione e infine l'interfaccia utente. Per quanto si è cercato di rendere il documento indipendente dalle scelte implementative, abbiamo ritenuto necessario fare riferimento a tali scelte durante la fase di progettazione. Per questo motivo, dove necessario, saranno discussi aspetti implementativi rilevanti per la fase di progettazione.

1.1 Panoramica

Il sistema TravelDream è stato ideato per aiutare le persone nella ricerca di pacchetti viaggio completi e pronti per essere acquistati, senza tralasciare la possibilità di eventuali modifiche agli stessi. Si è deciso di progettare un'interfaccia web di accesso per rendere il servizio disponibile ad un maggior numero di persone, optando per un design minimale e classico secondo i gusti dell'epoca tecnologica corrente. Per raggiungere il suo scopo, il sistema mette a disposizione strumenti di ricerca arricchiti di filtri per destinazione. Gli utenti registrati al sistema ne sfruttano a pieno le potenzialità, attraverso la gestione di un carrello contenente i pacchetti in attesa di conferma dell'ordine e di uno storico contenente quelli già acquistati. Le funzionalità accessibili agli utenti non registrati sono limitate alla sola visualizzazione per invogliare questi ad effettuare la registrazione e l'autenticazione. Agli utenti non autenticati non è permesso l'ac-

quisto dei pacchetti, ma è comunque possibile per loro regalare un componente viaggio di una giftlist il cui codice è in loro possesso.

1.2 Descrizione dell'architettura

Lo stile architetturale che abbiamo adottato, multi-tier, è tipico di applicazioni di tipo enterprise, in cui i vari livelli logici (tier) sono distribuiti su diverse macchine fisiche (layer) che comunicano tra loro via rete. In un'architettura di questo tipo, inoltre, ogni tier comunica solo con il livello immediatamente precedente e con quello immediatamente successivo, rendendo l'applicazione il più possibile modulare. L'architettura di TravelDream è suddivisa in quattro livelli logici: il client tier, il web tier, il business tier e il data tier. Il client tier è allocato sul terminale da cui l'utente accede al sistema, il web tier e il business tier risiedono sull'Application Server, mentre il data tier risiede sul DBMS.

Client tier È il livello da cui gli utenti accedono all'applicazione tramite un comune browser web. Questo tier comunica mediante il protocollo HTTP con il web tier, inviando al server i dati inseriti dall'utente e visualizzando le pagine HTML ricevute. Inoltre, il browser interpreta l'eventuale codice Javascript presente, che realizza alcune funzionalità per migliorare le prestazioni del servizio e l'esperienza dell'utente (ad esempio richieste asincrone al server oppure validazione di form).

Web tier Questo strato riceve le richieste HTTP dal client tier e risponde inviando pagine HTML. Le pagine vengono generate in base ai dati ricevuti dall'utente e all'interazione con il business tier. Il web tier sarà implementato con tecnologia JEE, e funzionerà all'interno di un application server compatibile.

Business tier Questo livello incapsula la logica applicativa e comunica direttamente con il database. I dati dell'applicazione, ovvero il modello all'interno di un pattern di tipo Model – View – Controller (MVC), sono rappresentati da oggetti di tipo Entity Beans. La logica applicativa e l'interazione con il database (il controller dell'applicazione) sono invece realizzati mediante componenti EJB. Per accedere al database, questo tier fa uso della specifica JPA, che svolge anche funzionalità di Object-Relational Mapping (ORM), astruendo il modello di tipo relazionale - implementato dalla base di dati - in un modello di dati a oggetti, con cui l'applicazione interagisce.

Data tier È costituito da un DBMS, che realizza la persistenza dei dati. L'application server comunica con il database utilizzando le tecnologie standard di Java (JDBC). Poiché la comunicazione tra DBMS e application server avviene via rete, il database può risiedere indifferentemente sulla stessa macchina fisica del server o su un'altra.

È previsto che il business tier interagisca con un server di posta elettronica, utilizzando il protocollo Simple Mail Transfer Protocol (SMTP), per inviare agli

utenti i messaggi contenenti il codice di conferma della registrazione. Questo server è esterno al sistema e può essere implementato anche da un servizio esterno (ad esempio dal servizio Gmail di Google).

Capitolo 2

Progetto dei dati

La struttura del livello di persistenza dei dati di interesse per l'applicazione è basata su una base di dati relazionale, implementata sul server MySQL a cui si interfaccia la logica applicativa.

2.1 Modello Entity-Relationship

Le entità principali che abbiamo modellato nel sistema sono Utente, Pacchetto e ComponenteViaggio, ciascuno dei quali specializzato in diverse categorie. Utente viene generalizzato da Impiegato e Cliente. L'Impiegato non presenta particolari attributi specifici, in quanto abbiamo ipotizzato che eventuali dati riguardanti stipendio, ferie e altre caratteristiche amministrative sarebbero state memorizzare in un sistema differente. Non presenta nemmeno relazioni con altre entità perchè abbiamo ritenuto non fosse rilevante tenere traccia di quale particolare Impiegato abbia effettuato una determinata modifica nel sistema. Il Cliente invece, presenta la relazione Carrello per tenere traccia dei pacchetti personalizzati che saranno presenti nella rispettiva sezione, e l'attributo booleano "Acquistato" sulla relazione permette anche la ricostruzione dello Storico. Inoltre possiede degli attributi specifici legati alle modalità di pagamento.

ComponenteViaggio è generalizzato nei singoli componenti base che possono comporre un Pacchetto, ovvero Volo, Hotel o Escursione. Di queste entità, l'unica precisazione che è doveroso segnalare è inerente all'Hotel, poichè quest'ultimo non presenta un attributo "Data". Ciò è dovuto al fatto che un Hotel, teoricamente, è disponibile tutto l'anno e non solo in determinati periodi. Quindi l'unico vincolo sulle date di un Hotel potrebbe essere dovuto alla totale occupazione dello stesso in certi intervalli di tempo. Poichè però non ci è sembrato fattibile modellare anche un fittizio sistema di gestione delle prenotazioni per ogni Hotel, abbiamo ipotizzato che ciascuno di essi abbia una capacità infinita o che, per meglio dire, abbia sempre un posto disponibile. In conclusione, in conseguenza a queste assunzioni, non essendoci termini temporali per gli Hotels, il numero di notti prenotate all'interno di un pacchetto verrà calcolato

automaticamente dal sistema come differenza tra la data di arrivo e la data di ritorno, sapendo che su di esse verrà già effettuato un controllo di consistenza.

Arriviamo infine all'entità *Pacchetto*. Oltre a un ovvio *Identificativo* dispone di un attributo "Luogo". Questo perchè come affermato nel RASD, dal punto di vista del cliente il pacchetto avrà una struttura "rigida" e legata alla specifica destinazione che rende tematico quel determinato pacchetto. Attraverso di esso, quindi, durante la personalizzazione sarà possibile mostrare automaticamente al Cliente, i soli componenti che può scegliere (in sostituzione a quelli già presenti) che rispettino il "tema" del pacchetto. Questo può venire anche considerato come un controllo di consistenza "preventivo" sui luoghi dei componenti di un pacchetto (per evitare situazioni del tipo: volo per Parigi, Hotel a Istanbul, ritorno da Barcellona). Tale Attributo del Pacchetto sarà ovviamente modificabile dall'Impiegato all'atto di creazione di un *PacchettiPredefinito*. Quando il cliente deciderà di aggiungere un Pacchetto al carrello, verrà creata una tupla all'interno di *PacchettoPersonalizzato* contenente una copia dei dati presenti all'interno di quello predefinito. Per tenere traccia dei singoli componenti interni a un pacchetto generico, sono presenti le relazioni *ContienePers* e *ContienePred*. La relazione per i Personalizzati ha degli attributi booleani che segnalano, per ogni componente, se è stato aggiunto alla *GiftList* del Cliente, se è stato Acquistato dallo stesso, o se gli è stato Regalato da un amico (ovviamente con i dovuti check sul fatto che un componente non possa essere contemporaneamente Acquistato e Regalato).

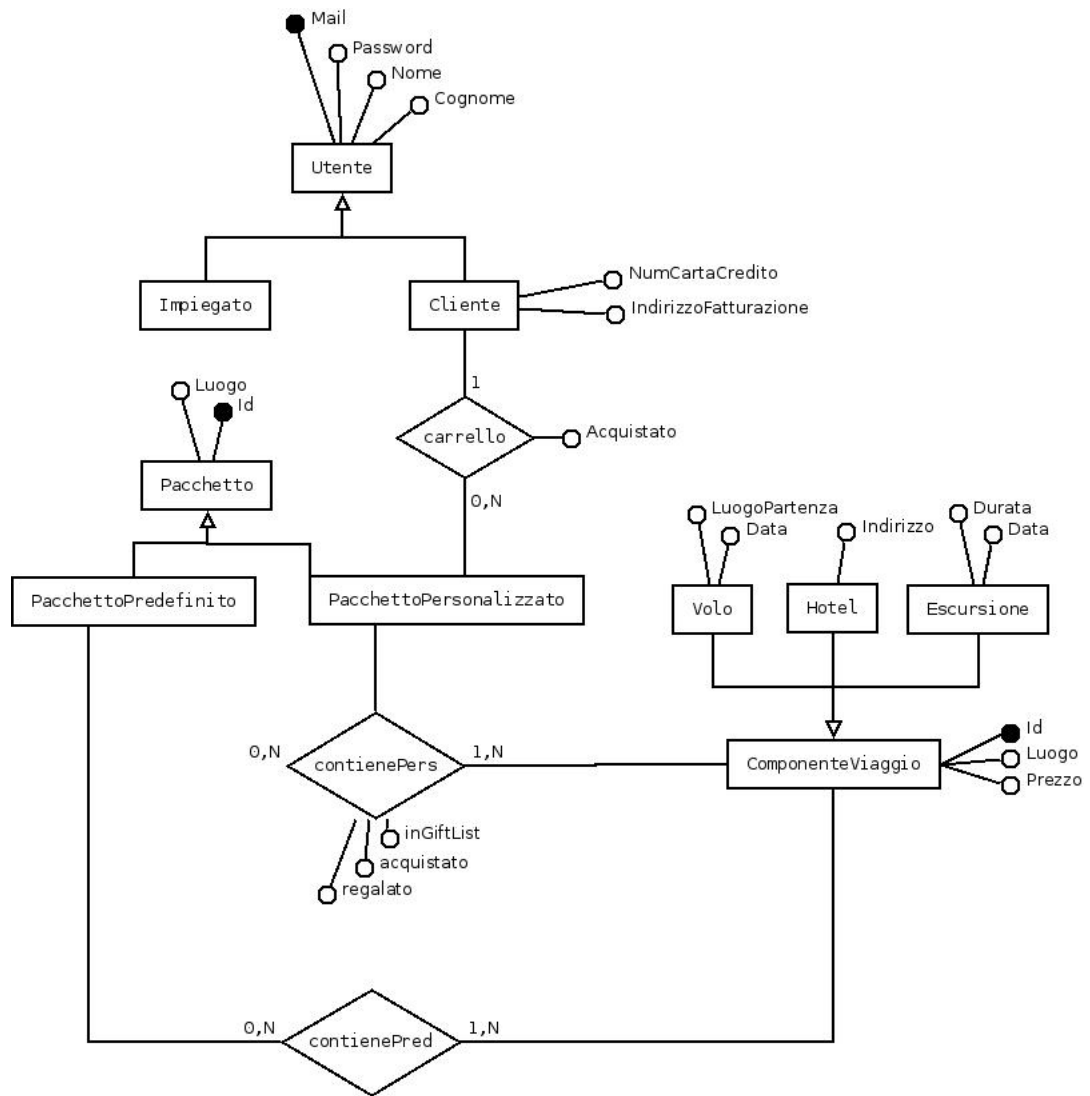


Figure 2.1: Modello ER

2.2 Ristrutturazione del modello ER

Affinchè il modello ER possa venire tradotto in uno Schema Logico, è necessario un processo di normalizzazione che ne risolva le generalizzazioni e attributi composti (questi ultimi non presenti nel nostro modello). Di conseguenza si sono apportate le seguenti trasformazioni:

- Generalizzazione Utente: E' stata risolta con uno spostamento dell'entità

padre verso i figli, copiando quindi gli attributi comuni in Impiegato e Cliente. Nonostante così l'entità Impiegato rimanga isolata dal resto del modello, ha senso separarlo dai Clienti per aumentare l'efficienza del sistema nell'effettuare il Login di uno dei due tipi di utenti (e marginalmente evitare anche i valori NULL in corrispondenza degli attributi specifici di Cliente). Dopotutto scandire due tabelle o una sola di cardinalità pari alla somma delle due più piccole non comporta una grande differenza di complessità.

- Generalizzazione Pacchetto: Risolta anch'essa con uno spostamento verso i figli. Questo per lasciare nettamente separati i Pacchetti Predefiniti e Personalizzati che hanno natura e operazioni effettuabili sugli stessi totalmente diverse (i Predefiniti vengono creati dagli impiegati, possono venire ricercati e visualizzati dai Clienti ma nel momento stesso in cui vengono aggiunti al Carrello diventano in tutto e per tutto Personalizzati). Questo ovviamente comporta la copiatura degli attributi di Pacchetto su entrambi i figli.
- Generalizzazione ComponenteViaggio: Risolta anche quest'ultima con lo spostamento verso i figli. Anche qui, per aumentare l'efficienza della ricerca quando vanno mostrati ai clienti risultati di selezioni che rispettino il tema dei pacchetti personalizzati. Questa soluzione ovviamente comporta anche la duplicazione su ciascun figlio, delle relazioni di contenimento che erano presenti sul padre.

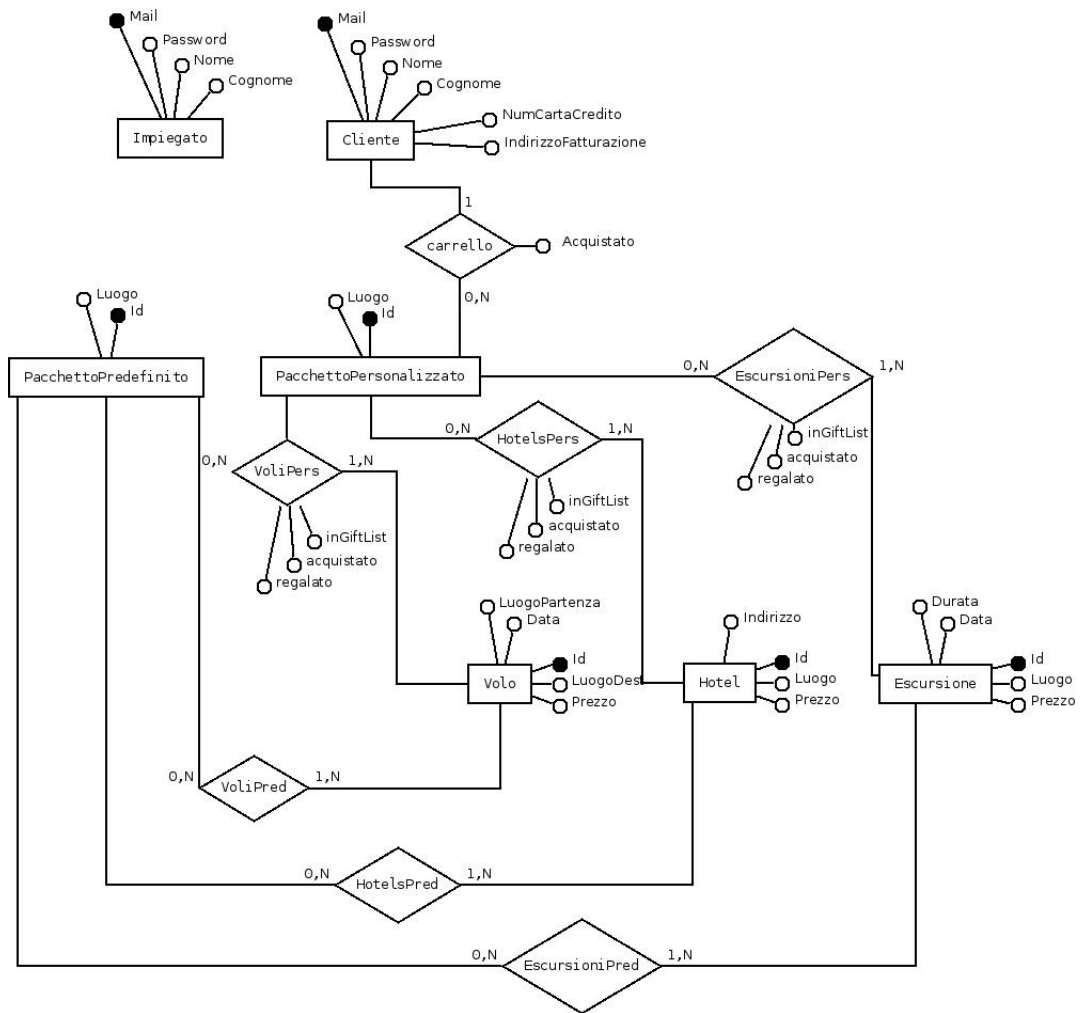


Figure 2.2: Modello ER normalizzato

2.3 Traduzione verso il modello logico

Infine, per la traduzione del modello ER nel Modello Logico che costituirà lo schema fisico della Base di Dati, bisogna scegliere come risolvere le relazioni. Nel nostro caso:

- **Carrello**: Essendo una relazione del tipo uno-a-molti (1-N), si può risolvere portando la chiave primaria dell'entità che partecipa con cardinalità 1, all'interno dell'entità che partecipa con cardinalità multipla, quindi la

Mail del Cliente nel PacchettoPersonalizzato. Poichè la relazione Carrello presentava anche l'attributo Acquistato per la ricostruzione dello storico, anche quest'ultimo viene trasferito in PacchettoPersonalizzato.

- Relazioni di contenimento: Tutte le relazioni di contenimento sono del tipo multi-a-molti (N-N). Questo ha reso quindi necessario introdurre per ciascuna di esse una “tabella ponte”, che contenga le chiavi primarie delle due entità che vi partecipano, oltre agli attributi sulla relazione stessa nel caso dei Pacchetti Personalizzati.

In definitiva, quindi, il Modello Logico risultante è il seguente (dove abbiamo identificato i nomi delle tabelle con il grassetto e le chiavi primarie con una sottolineatura):

Impiegato (Mail, Password, Nome, Cognome)

Cliente (Mail, Password, Nome, Cognome, NumCartaCredito, IndirizzoFatturazione)

PacchettoPredefinito (Id, Luogo)

Pacchetto Personalizzato (Id, Luogo, MailClienteCarrello, Acquistato)

Volo (IdVolo, LuogoPartenza, LuogoDestinazione, Data, Prezzo)

Hotel (IdHotel, Luogo, Prezzo, Indirizzo)

Escursione (IdEscursione, Luogo, Prezzo, Data, Durata)

VoliPers (Pacchetto, IdVolo, InGiftList, Acquistato, Regalato)

HotelsPers (Pacchetto, IdHotel, InGiftList, Acquistato, Regalato)

EscursioniPers (Pacchetto, IdEscursione, InGiftList, Acquistato, Regalato)

VoliPred (Pacchetto, IdVolo)

HotelsPred (Pacchetto, IdHotel)

EscursioniPred (Pacchetto, IdEscursioni)

Come già segnalato, l'attributo Acquistato in PacchettoPersonalizzato e gli attributi InGiftList, Acquistato e Regalato in VoliPers, HotelsPers e EscursioniPers sono valori booleani usati per la costruzioni di particolari liste da mostrare agli utenti.

Alla fine, il Modello Logico importato in MySQL WorkBench produce il seguente schema:

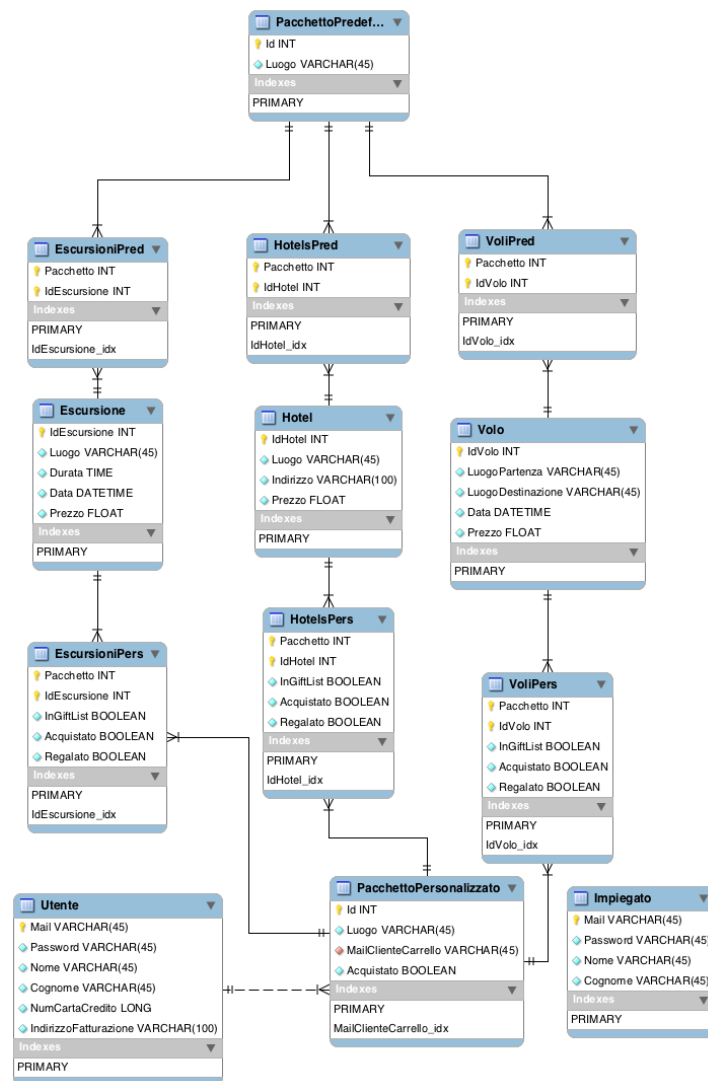


Figure 2.3: Diagramma dello schema logico

Capitolo 3

Progetto del business tier

Una volta individuati i dati di interesse per il nostro dominio applicativo, abbiamo progettato l'applicazione secondo l'architettura introdotta nel paragrafo 1.3, individuando in primo luogo le componenti software da realizzare per implementare la logica applicativa. Il business tier contiene sia le componenti software che descrivono il modello dei dati (entità di JPA o Entity Beans) che quelle che implementano la logica applicativa, implementate da componenti sviluppati secondo le specifiche EJB.

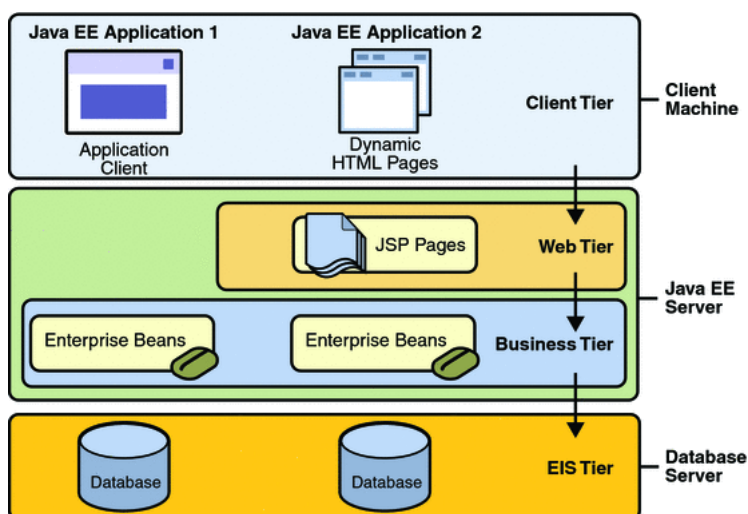


Tabella 3.1: Architetture implementativa

L'infrastruttura di supporto da utilizzare per realizzare il sistema è Java Enterprise Edition, in particolare verrà utilizzata una sua implementazione, Glassfish che contiene il web server Apache, il motore di servlet Tomcat, e la gestione della logica di business. Dal lato utente non si prevede l'installazione di

un'applicazione client specializzata, ma l'accesso avverrà attraverso un browser e richieste http al nostro server. Sulla stessa macchina il server Glassfish contiene gli Enterprise Java Beans, in particolare i Session Beans per la logica di business e gli Entity Beans per il collegamento con i dati. Infine i dati saranno memorizzati in un database gestito da un server MySQL, anch'esso inizialmente installato sulla stessa macchina del web server ma eventualmente i dati e il DBMS potrebbero essere distribuiti su macchine diverse. La piattaforma che si andrà a realizzare si configura quindi come una applicazione a più livelli: Client Tier, Web Tier, Business Tier, Data Tier.

3.1 Logica applicativa (EJB)

Per l'analisi dei componenti da implementare per realizzare il sistema ci siamo basati sul pattern Model-View-Controller. Questa strategia si adatta perfettamente alla separazione nei diversi livelli dell'architettura illustrata nella prima sezione relativa al design dell'applicazione. Tra i vantaggi principali di questa scelta c'è l'indipendenza dell'interfaccia utente dalla logica del sistema, permettendo la modifica di una delle due parti senza necessità di cambiamenti nell'altra. Inoltre gli oggetti del sistema che si occupano di controllare la modifica dei dati sono separati dalle strutture che contengono i dati stessi, garantendo in questo modo un migliore isolamento per le varie operazioni.

Per sviluppare la logica applicativa utilizzeremo soltanto i Session Bean in quanto tutte o quasi le operazioni che devono essere implementate restituiscono all'utente dei dati (entità o collezioni di entità) oppure una conferma, e devono farlo in modo sincrono (non tramite eventi o messaggi asincroni) a causa del modello di funzionamento delle applicazioni web standard e del protocollo HTTP (richiesta - risposta).

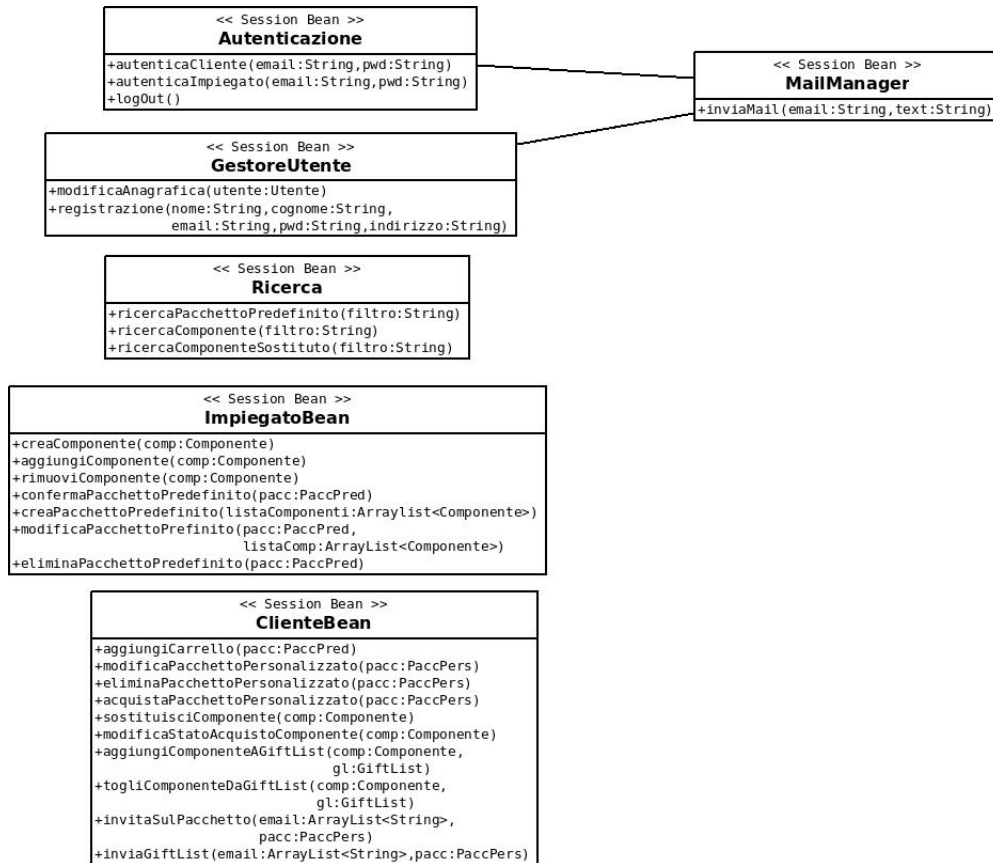


Figura 3.1: Modello dei Session Bean

Capitolo 4

Progetto del client tier

4.1 Progetto della navigazione

Per la progettazione e rappresentazione dei percorsi di navigazione relativi all'applicazione web, abbiamo seguito la convenzione definita dai diagrammi UX. Abbiamo suddiviso i diagrammi in riferimento a specifici utenti o funzionalità, per evitare di presentarne alcuni di dimensioni eccessive. Seguendo le convenzioni dei diagrammi UX, gli screen denotati dal simbolo “\$” rappresentano pagine raggiungibili da tutte le altre, mentre col simbolo “+” si rappresentano delle schermate con possibili risultati paginati (e i conseguenti metodi `prev()` e `next()` per navigare tra queste). Classi senza stereotipo rappresentano solo dati comuni ad altre screen/form. Gli attributi rappresentati fanno riferimento ai soli dati dinamici delle pagine.

Il primo diagramma rappresenta i percorsi di navigazione relativi a un utente che non ha effettuato il login. Come già ipotizzato un utente non registrato (o semplicemente se non ha ancora effettuato il login) può visualizzare `GiftList` e pacchetti inviati da amici. Mentre abbiamo ipotizzato che si possano regalare elementi di una `GiftList` anche senza effettuare il login, per partecipare a un pacchetto inviato da un amico è invece necessario (infatti il metodo di partecipazione non è presente in questo diagramma).

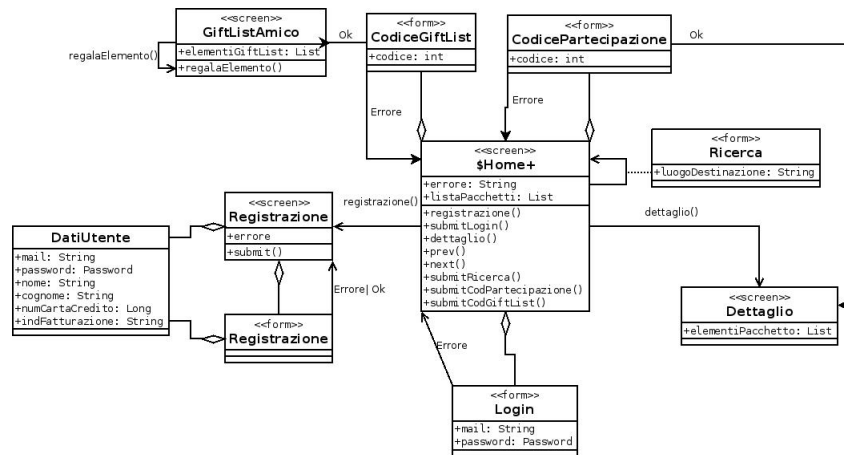


Figura 4.1: Diagramma UX dell'Utente non registrato

Di seguito riportiamo invece il diagramma relativo ai percorsi accessibili previa autenticazione. Ovviamente, a seguito della procedura di login, l'utente non potrà accedere ulteriormente alla form di login o al link per la registrazione, bensì al solo link di logout. Al momento del login, se questo va a buon fine, l'utente viene reindirizzato alla sua pagina personale, contenente l'elenco relativo al suo storico acquisti e l'elenco degli elementi attualmente nella sua gift list (ipotizziamo che accanto agli elementi in lista vi sia una spunta qualora siano stati già regalati da un amico). Inoltre in essa sono presenti i link alla classica modifica dell'anagrafica, e al carrello. All'interno del carrello vengono visualizzati i pacchetti ancora in attesa di essere acquistati, potendo anche accedere alla personalizzazione degli stessi. Come già specificato altrove, nella pagina di personalizzazione è previsto che vengano elencati automaticamente i componenti compatibili con quel pacchetto, dando all'utente la possibilità di scegliere tra questi in sostituzione a quelli esistenti. Il metodo `partecipa()`, ora visibile durante la visualizzazione di un pacchetto di un amico, permette l'acquisto di un pacchetto equivalente (con gli stessi componenti) da parte dell'utente, aggiungendolo quindi al suo storico (all'amico verrà segnalata tale partecipazione per email).



L'ultimo diagramma, infine, riguarda le attività dell'impiegato. Per evitare ulteriori ridondanze e un diagramma ancora più esteso, sono state rappresentate le sole pagine coinvolte nelle funzionalità aggiuntive previste per un impiegato. L'aggiunta di un componente in un nuovo pacchetto è previsto che passi attraverso la pagina di ricerca. I risultati presenteranno un link per l'aggiunta al pacchetto, visibile solo se si è giunti alla ricerca dei componenti dalla pagina di creazione. In caso contrario è implicito che si sia giunti alla ricerca di un componente per altri scopi (modifica, rimozione o creazione di un singolo componente) e quindi tale link non ha ragione di venire visualizzato.

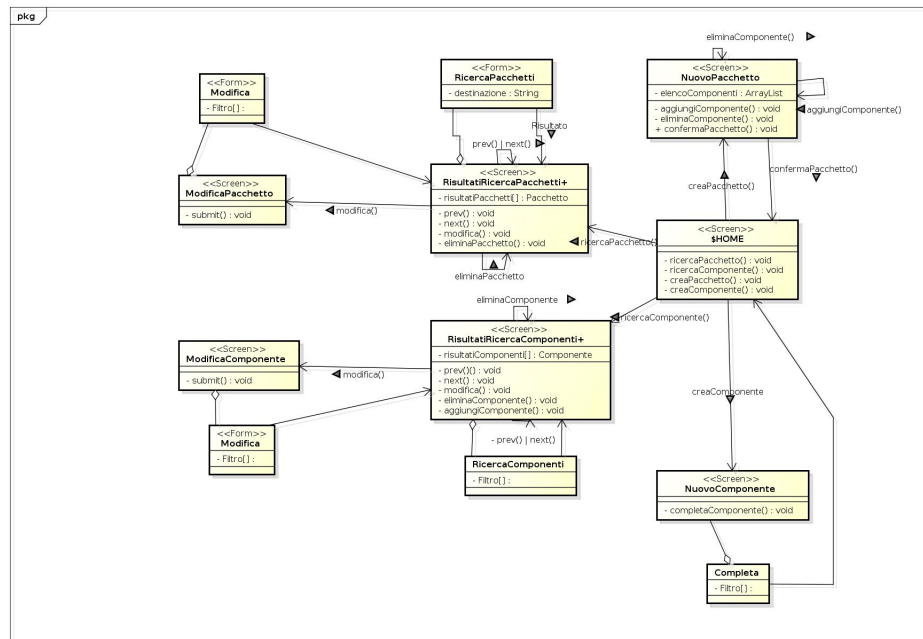


Figura 4.3: Diagramma UX dell'Impiegato

4.2 Sequence Diagramm

In questa sezione abbiamo introdotto alcuni diagrammi di sequenza UML, che riprendono quelli già studiati nella fase di analisi dei requisiti ad un livello di dettaglio maggiore e, per quanto possibile, più vicino a quello implementativo. L'intenzione è evidenziare i metodi e le classi (beans) così come pensiamo saranno realizzati all'interno del sistema, facendo riferimento a `<<screen>>` e `<<session>>` per indicare i diversi componenti attraversati dall'utente durante la navigazione. Per semplicità, non è rappresentata l'interazione tra i session bean e lo strato di persistenza dei dati, e sono omesse situazioni di errore o di funzionamento eccezionale del sistema.

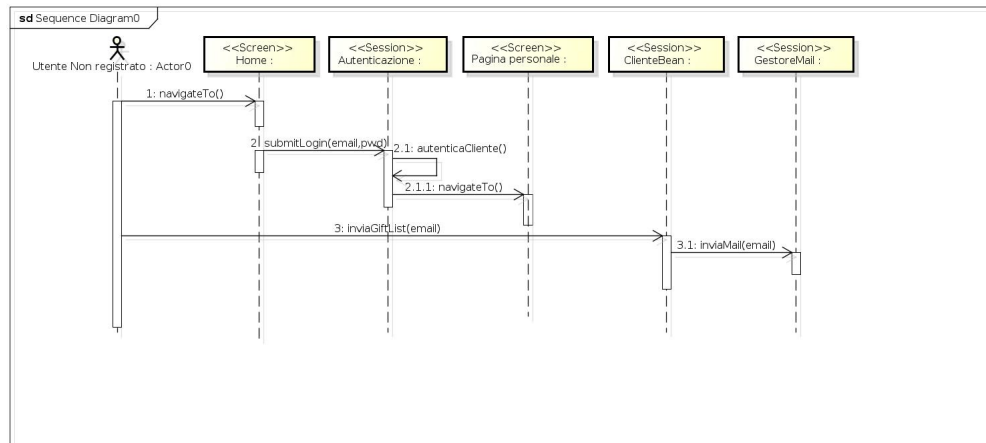


Figura 4.4: Diagramma relativo al login da parte di un utente già registrato. Questo, successivamente accede alla sua area personale e da qui invia il codice relativo alla sua gift list ad un amico, inserendone il rispettivo indirizzo email.

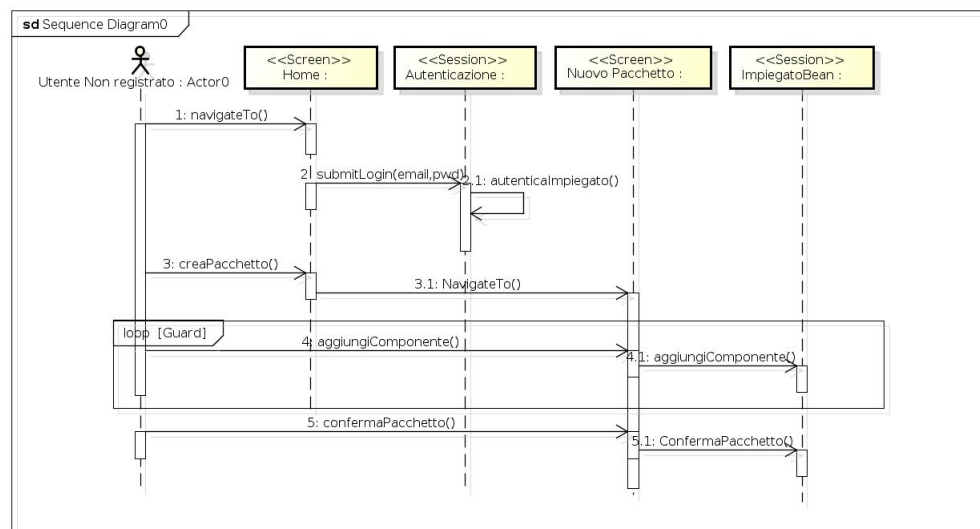


Figura 4.5: Diagramma relativo al login di un impiegato e alla creazione da zero di un pacchetto da parte dello stesso. Pertanto, l'impiegato seleziona iterativamente un insieme di componenti da aggiungere al pacchetto in via di creazione e poi procede alla sua conferma.

4.3 Page Sketch

In questa sezione presentiamo alcuni mockup relativi alle principali schermate del sistema. Le bozze sono state realizzate in parallelo alla progettazione della navigazione (diagrammi UX) e dei dati. Questi schemi hanno sia lo scopo di individuare meglio i percorsi di navigazione effettivamente di interesse per l'applicazione, che quello di supportare la fase di progetto dei dati, potendo individuare i dati da presentare all'interno delle schermate. I layout delle schermate qui presentati sono soltanto una bozza e verranno raffinati in fase di implementazione. I nomi attribuiti alle schermate nelle didascalie delle figure fanno riferimento alle <<screen>> dei diagrammi UX presenti nel documento.

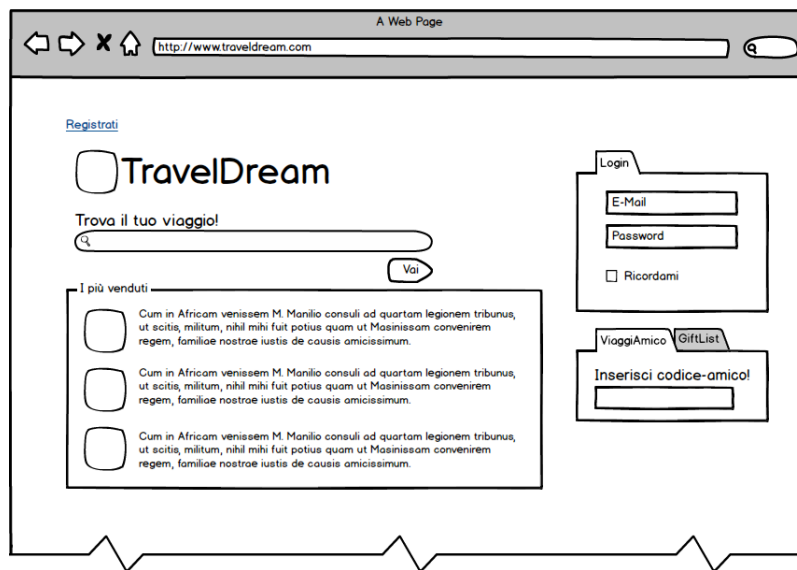



Tabella 4.1: Home page

A Web Page
http://www.traveldream.com/control_center

 **TravelDream**

Cerca un pacchetto o un componente già esistente:


☐ Pacchetto ☐ Componente

Aggiungi un pacchetto predefinito

Aggiungi un nuovo componente

Tabella 4.2: Home page dell'impiegato

A Web Page
http://www.traveldream.com/registrazione

 **TravelDream**

Registrazione

Inserisci il tuo nome:

Inserisci il tuo cognome:

Inserisci il tuo indirizzo email:

Inserisci una password:

Conferma la tua password:

Inserisci il codice della tua carta di credito:

Inserisci un indirizzo di fatturazione:

Login

☐ Ricordami

ViaggiAmico **GiftList**

Inserisci codice-amico!

Tabella 4.3: Form di registrazione

A Web Page

http://www.traveldream.com/my_giftlist

[Registrali](#)

TravelDream

La tua giftList

- ☐ Volo a AAA
- ☐ Volo a BBB
- ☐ Soggiorno a CCC
- ☐ Escursione DDD

Login

Bentornato Luca!

ViaggiAmico GiftList

Inserisci codice-amico!

Invia GiftList ad un amico!

Inserisci il suo indirizzo email:

Invia

Tabella 4.4: Screen dedicata all'invio della gift list

A Web Page

http://www.traveldream.com/modifica_pacch

TravelDream

Modifica Pacchetto

Seleziona un hotel tra quelli disponibili:

ComboBox

Seleziona una settimana tra quelle disponibili:

/ /

Seleziona nessuna o più escursioni tra quelle disponibili:

Escursione	Selezionata
Traghetto	<input type="checkbox"/>
Egogastronomico	<input checked="" type="checkbox"/>

Conferma

Tabella 4.5: Screen dedicata alla modifica di un pacchetto

The diagram shows a web browser window with the address `http://www.traveldream.com/richiesta_pacch`. The page content is as follows:

- Header:** "A Web Page" and navigation icons.
- Left Sidebar:** A link labeled "Registrati".
- Main Content Area:**
 - TravelDream:** The site logo.
 - Pacchetto Amico:** A section for a friend's package.
 - Description:** "Gallia est omnis divisa in partes tres, quarum unam incolunt Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra Galli appellantur."
 - Dates:** Two date pickers showing "01 / 01 / 12" and "07 / 01 / 12".
 - Options:** Two radio buttons labeled "Escursione AAA" and "Escursione BBB".
 - Action:** A button labeled "Vai" preceded by the text "Conferma la tua partecipazione:".
- Right Sidebar:**
 - Login:** Fields for "E-mail" and "Password", and a checkbox for "Ricordami".
 - ViaggiAmico / GiftList:** A section with the text "Inserisci codice-amico!" and an input field.

Tabella 4.6: Screen dedicata alla decisione di partecipare ad un pacchetto ricevuto da un amico

Capitolo 5

Rettifiche

Durante l'effettivo sviluppo dell'applicazione, fattori imprevisi inerenti le tecnologie utilizzate hanno portato alla modifica di alcuni dettagli rispetto a quanto specificato da questo documento. Inoltre, la mancanza di tempo ha anche comportato la decisione di non sviluppare alcune funzionalità secondarie non espressamente richieste, ma che nonostante questo erano state ottimisticamente aggiunte.

5.1 Funzionalità mancanti

1. Non è stato effettivamente sviluppato il servizio Javamail per l'invio di mail al momento della registrazione, dell'invito di partecipazione o dell'invito a visualizzare una giftlist. Nell'applicazione quindi, in corrispondenza dei moduli che si occupano di queste funzionalità, sono stati inseriti fittizi messaggi di conferma di invio del messaggio. In tali punti sarà comunque possibile aggiungere in futuro, senza troppa fatica, l'invio effettivo di una mail.
2. La modifica dell'anagrafica dell'utente non è stata sviluppata. Comunque si tratta anche in questo caso di una funzionalità non richiesta (nè particolarmente necessaria) di cui non è troppo difficile pensare una possibile aggiunta futura.
3. La modifica dei componenti. Nonostante sia effettivamente laborioso, lo stesso risultato si può ottenere con una cancellazione e un re-inserimento successivo. Quindi di fatto non si tratta di una reale funzionalità mancante, quanto più di una utile scorciatoia.

5.2 Struttura DB

La struttura descritta per il DB ha subito alcune lievi modifiche, che di fatto non comportano grandi cambiamenti strutturali. Si tratta di:

1. Gli utenti sono stati gestiti in un'unica tabella, con la presenza di una tabella ponte con i gruppi che prevede quindi il possibile un virtuale assegnamento di più gruppi a un singolo utente. Di fatto questa possibilità non viene utilizzata dall'applicazione (che prevede, come è implicito dalle scelte descritte, che a ogni utente corrisponda un solo tipo di gruppo). Tuttavia è stata così implementata per riuscire a utilizzare l'autenticazione di Glassfish. Difatti, dopo svariati (vani) tentativi di sviluppo utilizzando solo una unica tabella Utente contenente l'attributo "Tipo" inerente al gruppo, si è deciso di modificare la scelta architetturale con una che sicuramente sarebbe riuscita a funzionare con l'autenticazione di Glassfish.
2. Ai singoli componenti è stato aggiunto l'attributo booleano "Invalido". Questo ha permesso una gestione più agevole per la cancellazione dei componenti, che di fatto non vengono effettivamente rimossi dal DB ma solo contrassegnati come "invalidi". Questo permette di mantenere un'informazione utile di carattere "storico", e di gestire ugualmente la visualizzazione di tali componenti in pacchetti Personalizzati (nel carrello o nello storico) che li contengono, poichè inseriti prima della loro cancellazione. Da questo cambiamento si è aggiunta anche la decisione di cancellare tutti i pacchetti predefiniti che contengono il componente invalidato, per evitare che da quel momento venissero selezionati ulteriormente.

Infine un'ultima precisazione, che non ha comportato reali modifiche dello schema del DB. Le relazioni tra i pacchetti e i componenti sono rimaste del tipo multi-a-molti, ma nell'utilizzo dell'applicazione, di fatto, possono essere creati solo componenti con esattamente un volo di andata, un hotel e un volo di ritorno (e un numero arbitrario di escursioni). Questo rispecchia quanto descritto da RASD e DD (anche se l'impiegato poteva sembrare aver più libertà). Però potrebbe allora sorgere spontanea la domanda sul perché sono state lasciate come multi-a-molti mentre, prevedendo questa implementazione, potevano essere mappate come una relazione uno-a-uno per ciascun campo obbligatorio e unico, e quindi essere risolte come un semplice attributo all'interno dell'entità pacchetto. Ebbene, abbiamo ritenuto fosse una scelta implementativa saggia lasciare questa struttura "più generale", in modo da migliorare la scalabilità dell'applicazione, in modo da rendere più semplici eventuali espansioni future a pacchetti contenuti genericamente N componenti per tipo. Infatti questa struttura rigida è stata mantenuta (convertendola e deconvertendola di volta in volta) nei DTO presenti nell'interfaccia dell'EJB Client, rendendo molto agevole una modifica futura in questo senso.

5.3 Navigazione delle pagine

La struttura della navigazione tra le pagine web descritta dallo UX non ha subito particolarmente cambiamenti, se non quelli derivanti dalle modifiche appena citate. Infatti non è più presente la pagina di modifica dell'anagrafica. L'unica ulteriore piccola modifica, riguarda la modifica dei pacchetti (personalizzati e non): non

è previsto che per ogni componente aggiunto venga effettuata la ricerca in una pagina distinta, ma viene effettuato tutto nella stessa pagina di modifica del pacchetto.