Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di laurea magistrale in Ingegneria Informatica

PROGETTO DEL CORSO DI
INGEGNERIA DEL SOFTWARE 2

TRAVELDREAM
DESIGN DOCUMENT
Versione 1.1

Team
Alessandro Brunitti Matr. 817378
Andrea Corna Matr. 816737

Anno Accademico 2013-2014

Indice

Elenco delle figure

1 Modifiche

Di seguito vengono riportate le modifiche che sono state effettuate.

Modifiche Rasd		
Versione Modificata	Data	Modifica
1.0	26 Gennaio 2014	Nel BCE è stata aggiunta la descrizione dei componenti che permettono l'inserimento e la modifica di aerei, hotel ed escursioni.
1.0	27 Gennaio 2014	Apportate modifiche alla descrizione del database, aggiornando l'elenco delle tabelle e del modello logico.
1.0	29 Gennaio 2014	Corretti i navigation model e aggiornati i BCE.

2 Introduzione

Il documento di design vuole descrivere in modo sufficientemente dettagliato gli aspetti implementativi individuati durante la fase di specifica; in particolare, il team si è concentrato sull'organizzazione dei dati, sul modello architetturale da utilizzare per l'implementazione e sui vari modelli di navigazione. Tutti questi aspetti vengono presentati all'interno del documento, che viene suddiviso in opportune sezioni.
Dopo questa breve introduzione, nella sezione 3 viene trattato il design del database, partendo dal modello concettuale ER, fino ad arrivare alla descrizione dettagliata del modello logico, a cui segue lo schema della base di dati.
Nella sezione 4 vengono discussi l'organizzazione architturale, i vari modelli di navigazione, i componenti e le loro relazioni espresse tramite dei diagrammi di sequenza.

3 Design dei dati

3.1 Modello Entity Relationship

In questa sezione viene presentato il modello ER del database che verrà utilizzato per la memorizzazione dei dati utili all'applicazione. In particolare, si evidenziano tre entità principali:

- Utenti;
- Pacchetti;
- Prenotazioni.

3.1.1 Modello ER - Utenti

Gli utenti, come indicato all'interno del documento *RASD*, sono suddivisi in diverse tipologie:

- Utente registrato;
- Dipendente;
- Amministratore.

Si è deciso di proseguire con l'intenzione di mantenere un unico amministratore all'interno del sistema; inoltre, non viene mantenuta alcuna informazione riguardo agli utenti non registrati. Per quanto concerne i dati di registrazione dell'utente, si è scelto di inserirli all'interno dell'entità Anagrafica, in modo da alleggerire le entità Utente Registrato, Dipendente ed Amministratore. Si è preferito infatti mantenere direttamente accessibili i campi utili per il login e la comunicazione con l'utente, in modo da rendere le query più frequenti più performanti.

3.1.2 Modello ER - Prenotazione

L'entità *Prenotazione* si specializza in due entità:

- *Prenotazione Pacchetto*: tale prenotazione viene riferita ad un pacchetto offerto dall'agenzia e contiene un volo di andata e ritorno, un hotel ed almeno un'escursione. Il contenuto della prenotazione è una selezione delle scelte possibili presenti nel pacchetto;
- *Prenotazione Viaggio*: nel documento di RASD era stata inserita l'entità Viaggio, che rappresentava una creazione libera da parte dell'utente. In fase di implementazione si è scelto di non salvare la creazione fine a se stessa, tuttavia si vuole tener traccia delle prenotazioni che riguardano queste creazioni. L'entità Prenotazione Viaggio potrà contenere un'aereo di andata e di ritorno, un hotel e diverse escursioni.

3.1.3 Modello ER - Pacchetto

L'entità *Pacchetto* rappresenta le varie offerte che vengono proposte da Travel Dream e contengono una scelta di aerei di andata e ritorno, hotel ed escursioni. Tali liste permettono l'implementazione della funzionalità di *personalizzazione* del pacchetto da parte dell'utente.

3.1.4 Altre entità

Tra le entità restanti occorre sottolineare la *Condivisione*: contrariamente a quanto specificato nel diagramma delle classi, la condivisione non conterrà direttamente l'identificativo del pacchetto, ma quello della prenotazione dell'utente che ha deciso di condividere il proprio acquisto. In questo modo non solo si ha accesso a tutte le informazioni del pacchetto (che viene referenziato nella prenotazione), ma anche alle personalizzazioni realizzate.

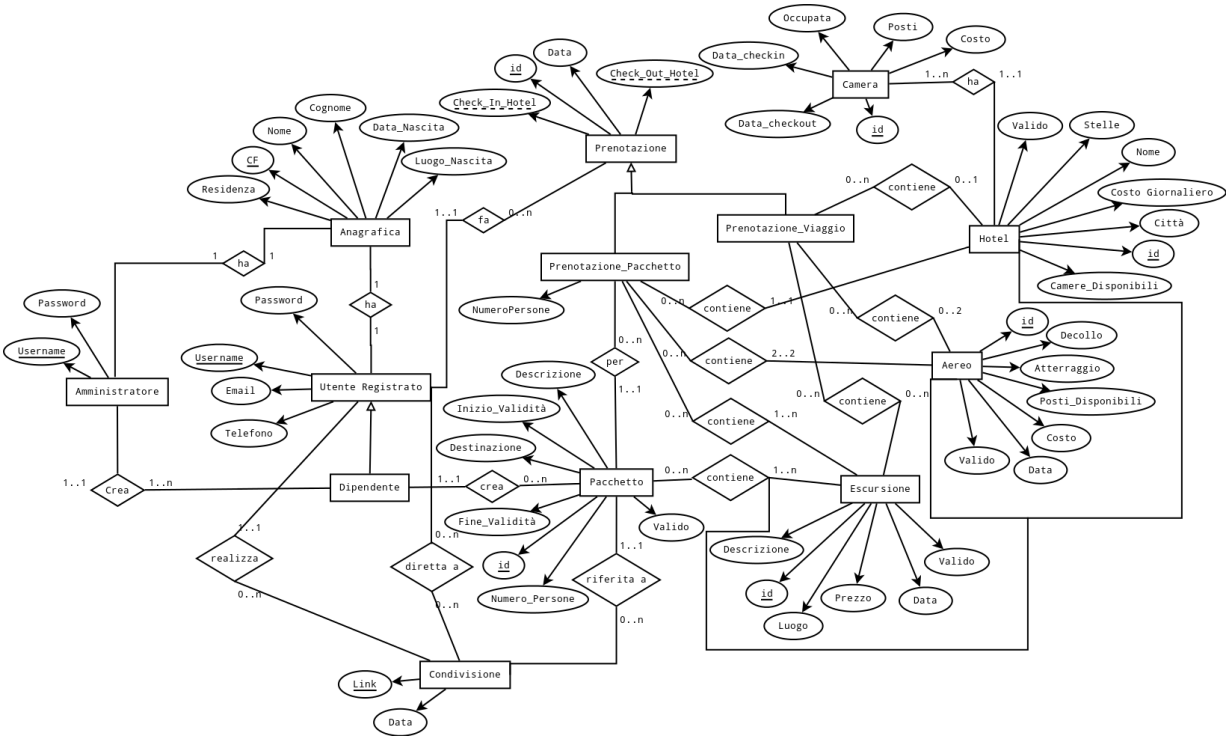

Figura 1: Modello ER

3.2 Dal modello ER al modello logico

Per poter implementare il modello ER tramite un database MySQL è stata necessaria la risoluzione delle generalizzazioni e di alcune relazioni con cardinalità molti a molti.

3.2.1 Risoluzione Generalizzazioni

Le modifiche applicate sono:

1. La generalizzazione che coinvolge *Utente Registrato* e *Dipendente* è stata risolta creando un'unica tabella *Utente*, che contiene tutti i dati dei vari utenti. Per differenziare le varie tipologie, sono state create due tabelle:
   - Gruppo: la tabella contiene un elenco dei gruppi degli utenti:
     - AMMINISTRATORE;

  - DIPENDENTE;
  - UTENTE;
- Gruppo_Utente: la tabella contiene la relazione di appartenenza di un utente ad un gruppo; ogni tupla è formata da *id_Utente* e *id_Gruppo*. Tali tabelle verranno utilizzate all'interno del reame del server glassfish per controllare l'autenticazione degli utenti.

2. La generalizzazione di *Prenotazione* è stata risolta creando due tabelle figlie, *Prenotazione_Pacchetto* e *Prenotazione_Viaggio*. Le due tabelle sono molto simili tra loro, ad eccezione di:
   (a) Un campo *id_Pacchetto* presente nella prima che si riferisce al pacchetto al quale appartengono tutti gli elementi della prenotazione.
   (b) I campi riguardanti gli aerei, gli hotel e le escursioni della tabella *Prenotazione_Viaggio* possono essere vuoti, mentre nella tabella *Prenotazione_Pacchetto* questo non è possibile.
   (c) I campi *Check_Out_Hotel* e *Check_In_Hotel* sono opzionali nella tabella *Prenotazione_Viaggio*, mentre sono obbligatori nela tabella *Prenotazione_Pacchetto*.
   (d) La tabella *Prenotazione_Pacchetto* possiede un campo *NumeroPersone* che indica per quante persone è stato prenotato il pacchetto presente in ogni tupla.

3.2.2 Risoluzione Relazioni molti a molti

Le modifiche applicate sono:

1. Le relazioni molti a molti che intercorrono tra *Pacchetto* e *Aereo, Hotel* ed *Escursione* sono state risolte inserendo delle tabelle ponte, che contengono la chiave primaria del pacchetto e dell'elemento considerato. Tali tabelle ponte sono:
   - Aereo_in_Pacchetto;
   - Hotel_in_Pacchetto;
   - Escursione_in_Pacchetto.
2. La relazione molti a molti tra le tabelle *Prenotazione_Viaggio* e *Prenotazione_Pacchetto* e la tabella *Escursione* è stata risolta tramite due tabelle ponte:
   - Escursioni_in_Prenotazione_Viaggio: la tabella contiene le chiavi primarie delle tabelle *Prenotazione_Viaggio* e *Escursione*;
   - Escursioni_in_Prenotazione_Pacchetto: la tabella contiene le chiavi primarie delle tabelle *Prenotazione_Pacchetto* e *Escursione*.

3.2.3 Modifica Entità

Prima di tradurre le relazioni che intercorrono tra le varie entità, occorre riportare alcune considerazioni:

1. Camera: l'entità camera non verrà tradotta in una tabella. L'hotel indicherà il numero di camere disponibili dedicate all'agenzia ed il controllo sulla disponibilità di camere verrà effettuato valutando prenotazioni già esistenti nel database comprese tra le date inserite dall'utente.

2. Amministratore: l'entità amministratore non verrà tradotta in una tabella, ma l'informazione verrà inserita nella tabella *Utente*.

3.2.4 Risoluzione delle relazioni

Una volta semplificato il modello ER sono state tradotte tutte le relazioni che intercorrono tra le varie entità tramite delle foreign key. All'interno del database si è scelto di utilizzare, ove possibile, la dicitura *id_NOMETABELLA* per identificare l'attributo della tabella che verrà relazionato con la chiave primaria della tabella referenziata. Riportiamo l'elenco delle foreign key:

- La relazione *ha* tra Amministratore, Dipendente ed Utente Registrato con Anagrafica è mappata in *id_Anagrafica* nelle tabella Utente.

- Le relazioni *Contiene* che interessano le entità Prenotazione_Pacchetto, Prenotazione_Prenotazione, Aereo e Hotel sono state tradotte inserendo i campi *id_Aereo_Andata*, *id_Hotel* e *id_Aereo_Ritorno* nelle tabelle Prenotazione_Viaggio e Prenotazione_Pacchetto.

- La relazione *Realizza* tra Utente Registrato e Condivisione è stata tradotta tramite un campo *id_Utente* nella tabella Condivisione, che si riferisce all'identificativo dell'utente.

- La relazione *Riferita a* tra Condivisione e Prenotazione_Pacchetto è stata tradotta con un campo *id_Prenotazione* nella tabella Condivisione, che si riferisce all'identificativo della prenotazione.

- La relazione *crea* tra Dipendente e Pacchetto è stata tradotta con un campo *id_Dipendente* nella tabella Pacchetto. In questo modo si sottolinea la necessità che il creatore di un pacchetto appartenga al gruppo dei dipendenti.

- La relazione *per* tra Prenotazione_Pacchetto e Pacchetto è stata tradotta con il campo *id_Pacchetto* nella tabella Prenotazione_Pacchetto.

- La relazione *fa* tra Utente_Registrato e Prenotazione è stata tradotta con un campo *id_Utente* nelle tabelle Prenotazione_Pacchetto e Prenotazione_Viaggio.

- La relazione *crea* tra Amministratore e Dipendente è stata tradotta con il campo opzionale *id_Amministratore* nella tabella Utente. Tale campo verrà riempito solo se l'utente in questione ricopre il ruolo di dipendente dell'agenzia.

3.2.5 Modello Logico

Di seguito viene riportata la rappresentazione del modello logico.

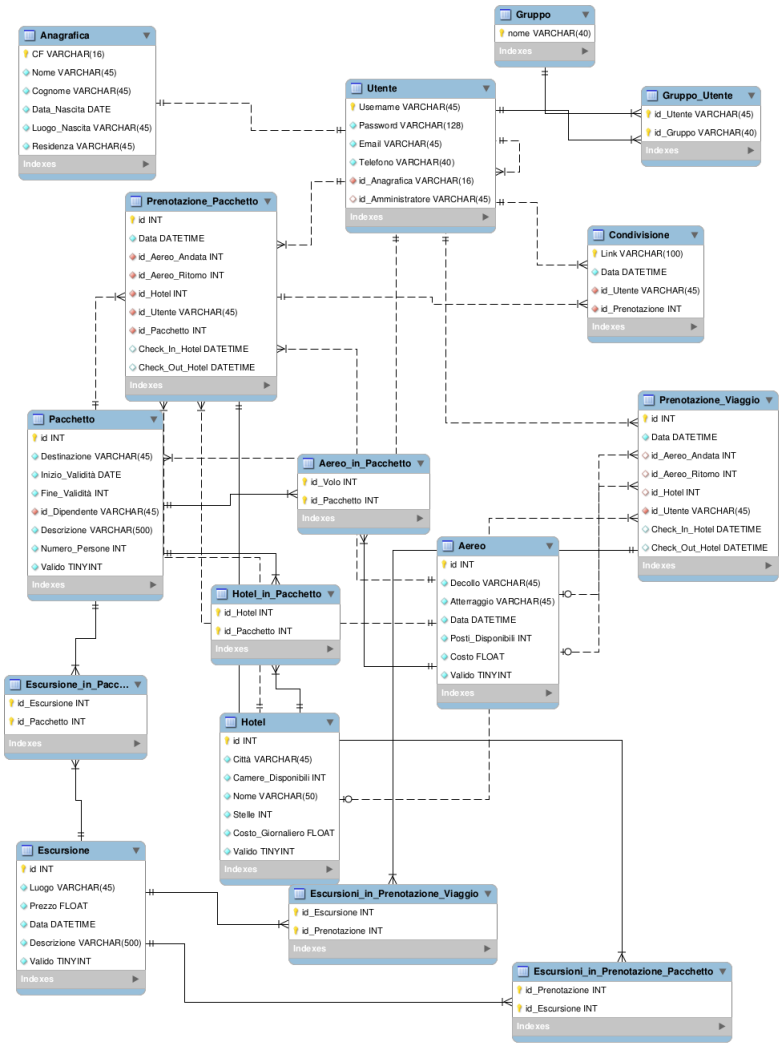

Figura 2: Modello Logico

Il risultato della traduzione fisica del database è indicato nell'elenco seguente, utilizzando la convenzione secondo la quale le primary key sono identificate con il carattere in grassetto, mentre le foreign key con il carattere corsivo.
Aereo(**id**, Decollo, Atterraggio, Data, Posti_Disponibili, Costo, Valido)
Aereo_in_Pacchetto(***id_Volo***, ***id_Pacchetto***)
Anagrafica(**CF**, Nome, Cognome, Data_Nascita, Luogo_Nascita, Residenza)
Condivisione(**Link**, Data, *id_Utente*, *id_Prenotazione*)
Escursione(**id**, Luogo, Prezzo, Data, Descrizione, Valido)
Escursione_in_Pacchetto(***id_Escursione***, ***id_Pacchetto***)
Escursioni_in_Prenotazione_Pacchetto(***id_Escursione***, ***id_Prenotazione***)

Escursioni_in_Prenotazione_Viaggio(***id_Escursione***, ***id_Prenotazione***)
Gruppo(**nome**)
Gruppo_Utente(***id_Utente***, ***id_Gruppo***)
Hotel(**id**, Città, Camere_Disponibili, Nome, Stelle, Costo_Giornaliero, Valido)
Hotel_in_Pacchetto(***id_Hotel***, ***id_Pacchetto***)
Pacchetto(**id**, Destinazione, Inizio_Validità, Fine_Validità, *id_Dipendente*, Descrizione, Numero_Persone, Valido)
Prenotazione_Pacchetto(**id**, Data, *id_Aereo_Andata*, *id_Aereo_Ritorno*, *id_Hotel*, *id_Utente*, *id_Pacchetto*, Check_In_Hotel, Check_Out_Hotel, NumeroPersone)
Prenotazione_Viaggio(**id**, Data, *id_Aereo_Andata*, *id_Aereo_Ritorno*, *id_Hotel*, *id_Utente*, Check_In_Hotel, Check_Out_Hotel)
Utente(**Username**, Password, Email, Telefono, *id_Anagrafica*, *id_Amministratore*)

3.3 Note Finali

Di seguito vengono riportate alcune note riguardati il database utilizzato:

1. La password verrà codificata tramite l'algoritmo crittografico MD5 con digest SHA-256; per tale motivo il campo che la rappresenta ha uno spazio per 128 caratteri.

2. Nel diagramma delle classi presentato nel RASD la classe *Viaggio* poteva contenere un numero illimitato di aerei, hotel ed escursioni. In fase implementativa si è scelto di limitare la cardinalità degli elementi del Viaggio:

   - Al massimo un aereo di andata e uno di ritorno;
   - Al massimo un hotel.

3. L'interfaccia *Trasporto* viene omessa all'interno del database: nel contesto attuale l'unica entità che erediterebbe da Trasporto è Aereo ed è quindi necessario inserire la suddetta tabella. Inoltre, in caso di estensioni successive, sarà necessario risolvere la generalizzazione tramite la creazione di più tabelle, in quanto i vari mezzi di trasporto saranno caratterizzati da proprietà differenti.

4. La traduzione della relazione *crea* tra Amministratore e Dipendente può risultare superflua nell'ipotesi che nel sistema sia presente un solo amministratore. Tuttavia si è deciso di mantenere il campo id_Amministratore nella tabella Utente in modo da rendere il sistema facilmente estensibile nel caso di aggiunta di ulteriori amministratori.

5. La relazione *diretta a* che unisce Condivisione con Utente Registrato non è stata tradotta poichè non si tiene alcuna informazione riguardo i destinatari delle condivisioni.

6. Il campo *valido* inserito nelle varie tabelle viene utilizzato per identificare tutte le tuple che rappresentano oggetti ancora validi e prenotabili dagli utenti. In questo modo si risolvono i problemi dell'eliminazione di componenti dovuti ai vincoli di integrità derivanti dalle foreign key tra le varie tabelle.

4 Design dell'applicazione

4.1 Livelli Architetturali

Per l'utilizzo del sistema, il client non dovrà dotarsi di alcuna applicazione specializzata, poichè l'accesso al sistema avverrà tramite un qualsiasi browser che sia in grado di inviare e riceve richieste tramite il protocollo http. Sulla stessa macchina il server Glassfish gestisce la logica del sistema; al suo interno verranno inseriti tutti i Java Entenprise Bean, organizzati in:

- Session Beans per la gestione della logica di business;
- Entity Beans per il collegamento con i dati contenuti nel database.

Infine, i dati verranno memorizzati in un database gestito da un server MySQL, che durante la fase di sviluppo risiederà sulla medesima macchina in cui l'application server è attivo; tuttavia i dati potranno successivamente essere distribuiti su macchine diverse. La piattaforma è quindi strutturata su diversi livelli:

- Client tier;
- Web tier;
- Business tier;
- Data tier.

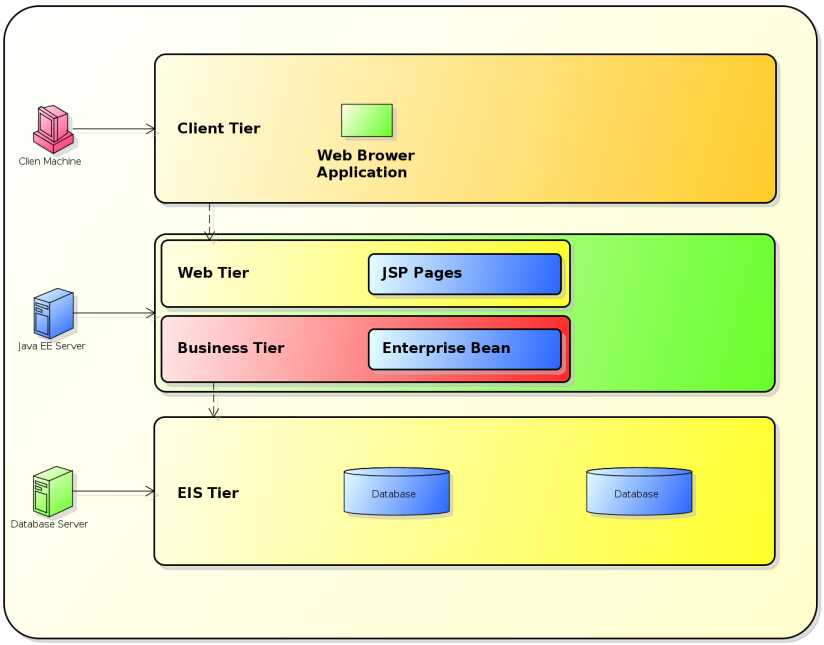

Figura 3: Suddivisione dell'architettura

4.2 Modelli di Navigazione

Per semplificare la visualizzazione, i diagrammi di navigazione sono stati suddivisi in 4 parti:

- Homepage principale;
- Homepage Utente Registrato;
- Homepage Dipendente;
- Homepage Amministratore.

Tali modelli riprendono i mockup presenti nel documento RASD; tuttavia, poichè tali anteprime fornivano solo un accenno sulla reale struttura dell'interfaccia utente, sono state inserite modifiche e aggiunte in modo da migliorare la navigazione dell'utente. Vengono riportate di seguito alcune modifiche:

- I link *Personalizza Pacchetto* e *Acquisto* nella homepage dell'utente registrato e del dipendente sono stati accorpati e inseriti nelle screen raggiungibili dopo la pressione del link *MostraOfferte*.
- Nelle homepage dei vari attori è stato inserito il link *Logout*.

4.2.1 Modello di Navigazione - Homepage

Questo diagramma mostra la navigazione di un utente generico che accede alla pagina principale del sistema. In particolare, vengono specificate le funzionalità di *Registrazione*, che permette ad un nuovo utente di inserirsi nel database dell'agenzia, e quella di *Login*. Un utente che effettua il login accederà alla propria pagina personale, tramite la quale accederà alle proprie funzionalità. Infine viene fornita *la visualizzazione di una condivisione*, ottenuta inserendo il link ricevuto tramite un servizio esterno al sistema.

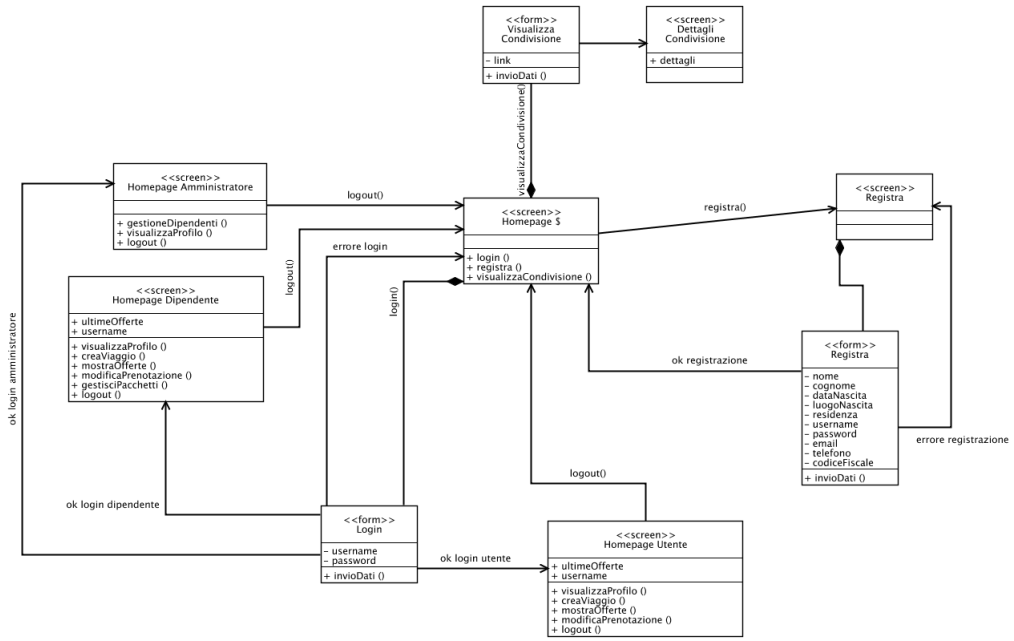

Figura 4: Modello di Navigazione - Homepage Principale

4.2.2 Modello di Navigazione - Homepage Utente Registrato

Il modello di navigazione presenta tutte le funzionalità fornite ad un utente registrato che ha effettuato con successo il login, in particolare la possibilità di acquistare pacchetti, creare i propri viaggi e navigare nello storico delle proprie prenotazioni.

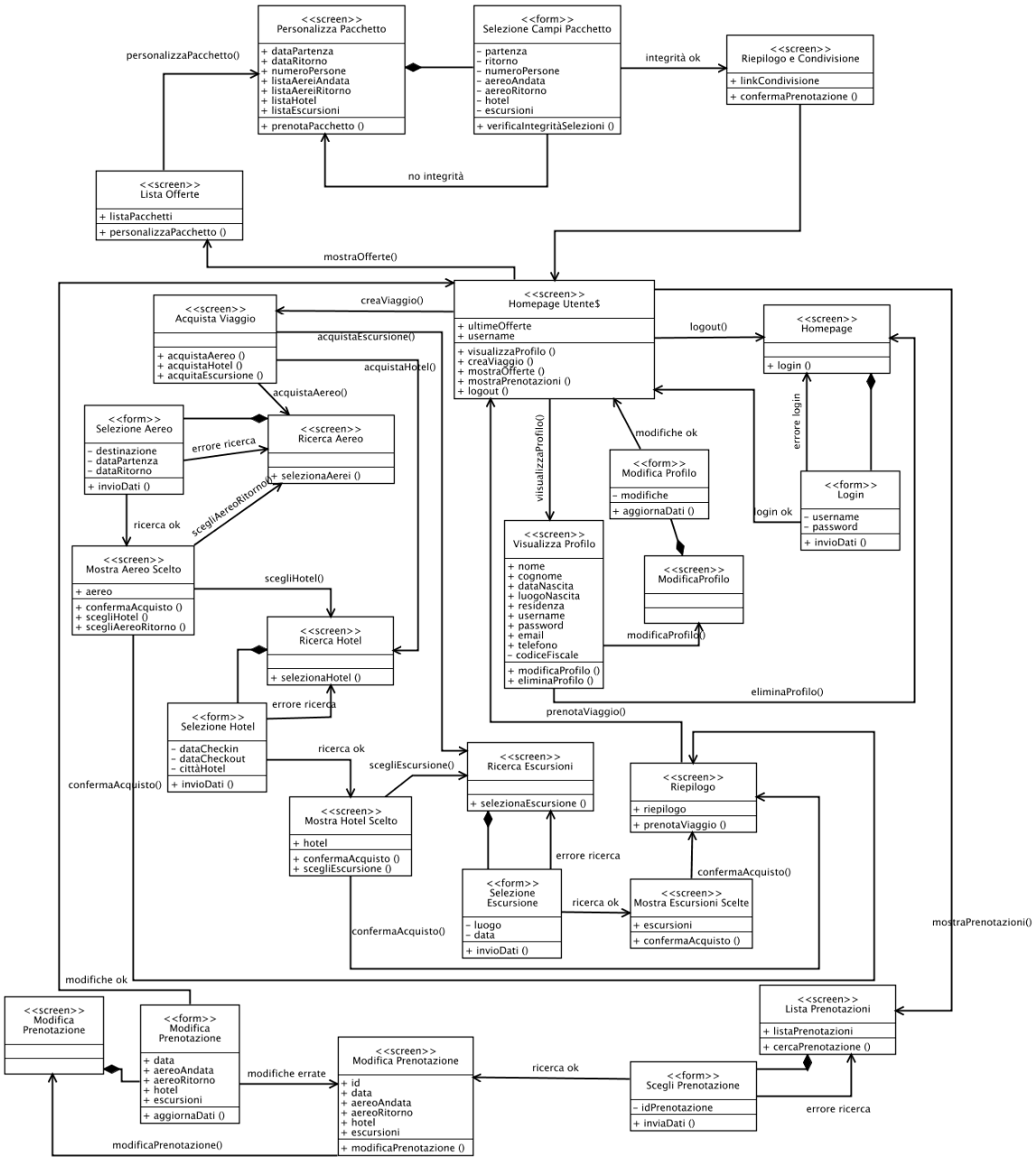

Figura 5: Modello di Navigazione - Homepage Utente Registrato

4.2.3 Modello di Navigazione - Homepage Dipendente

Il modello di navigazione del dipendente, estensione di quello dell'utente registrato, è stato ottenuto aggiungendo le funzionalità di:

- Creazione e gestione dei pacchetti;
- Creazione e modifica dei componenti base (aereo, hotel ed escursione).

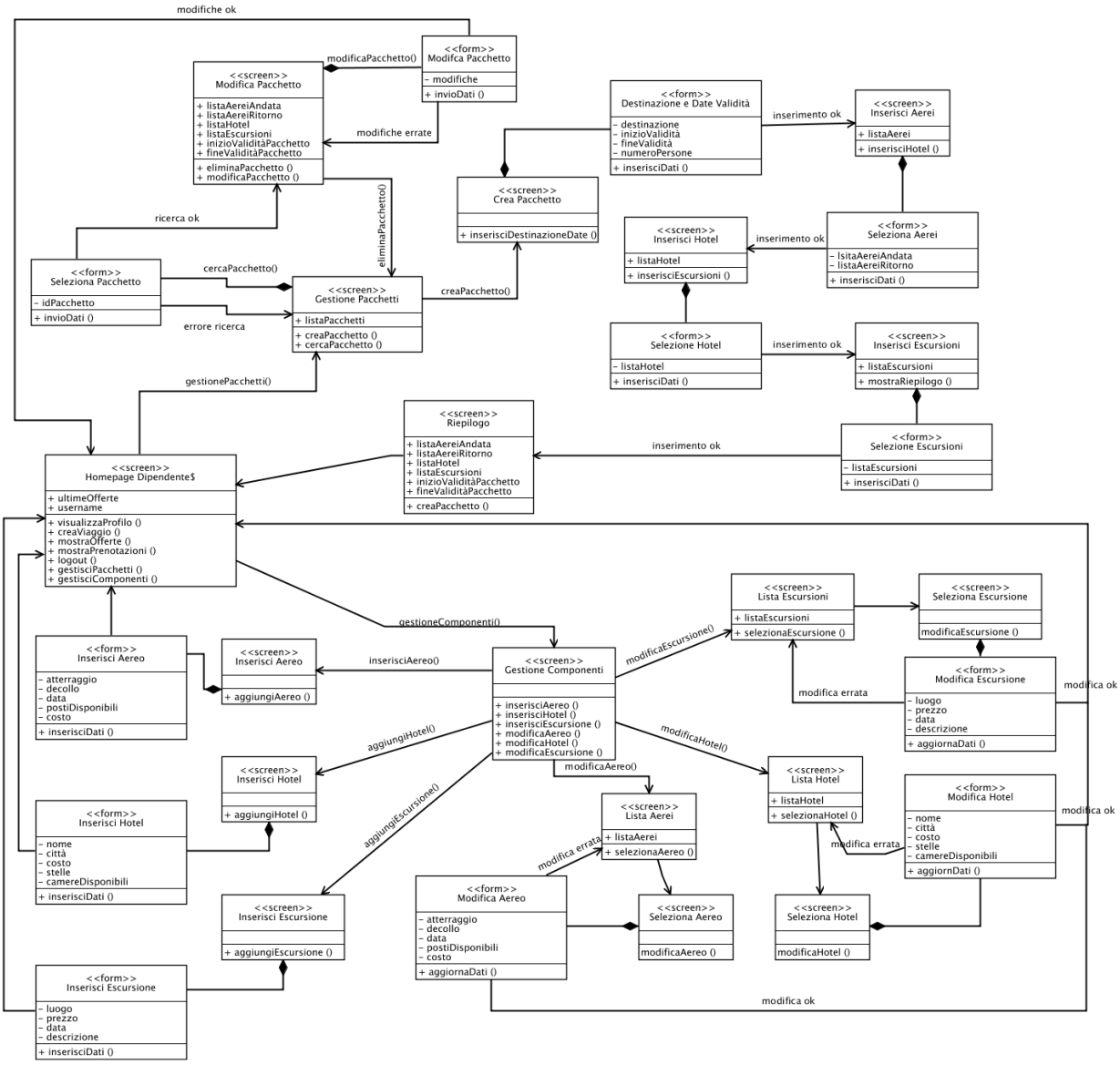

Figura 6: Modello di Navigazione - Homepage Dipendente

Per rendere più leggibile il modello, vengono solo rappresentate le funzioni tipiche del dipendente.

4.2.4 Modello di Navigazione - Homepage Amministratore

Il modello di navigazione mostra le funzioni che vengono concesse all'amministratore del sistema, in particolare la gestione dei dipendenti.

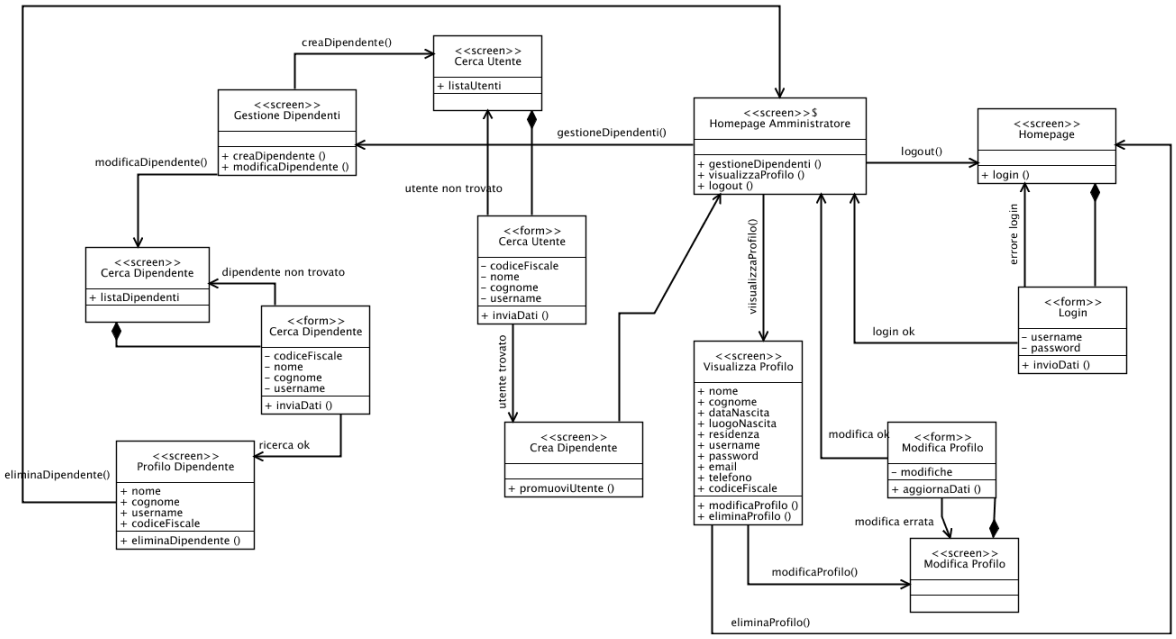

Figura 7: Modello di Navigazione - Homepage Amministratore

4.3 Componenti

L'analisi dei componenti necessari per la realizzazione del sistema è basata su pattern Model-View-Controller, che rispecchia la suddivisione dei livelli architetturali presentata nella sezione 4.1. La suddivisione netta derivata dal pattern utilizzato permette di separare la logica dall'interfaccia utente: la comunicazione avviene tramite delle interfacce che resteranno invariate anche a seguito di modifiche implementative. Un altro vantaggio deriva dal fatto che tutti gli oggetti destinati alla modifica dei dati sono separati dai dati stessi, permettendo un controllo più accurato. La rappresentazione dei vari componenti utilizza il linguaggio dei grafici UML BCE, all'interno dei quali vengono specificati:

- Boundary: raggruppano le funzionalità offerte agli utilizzatori del sistema che sono state indicate nei vari navigation model;
- Control: oggetti che consentiranno la comunicazione tra l'interfaccia utente e i dati;
- Entity: schematizzano le tabelle del database, come indicato nella sezione 3.

Le pagine HTML e JSP costituiranno l'implementazione delle boundary, mentre i SessionBean e le EntityBean costituiranno rispettivamente le implementazioni

delle componenti control ed entity. Di seguito vengono rappresentati i diagrammi BCE; per migliorare la leggibilità dei grafici, si è scelto di divididere la rappresentazione in due parti: la prima riferita alle funzionalità dell'utente registrato e del dipendente, la seconda alle azioni dell'amministratore. Nonostante questa suddivisione, si è scelto di mantenere invariate le entità *Control* all'interno delle rappresentazioni, per sottolineare l'unicità dell'entità. Vengono riportati solo i metodi che permettono di comprendere il flusso degli eventi: all'interno dell'implementazione possono variare leggermente.
All'interno delle Entity non sono stati riportati:

- Attributi: i vari attributi si mappano esattamente sui campi della tabella referenziata dalla Entity; per la lista è possibile consultare la specifica delle tabelle nella sezione 3.
- Metodi: all'interno della Entity saranno disponibili tutti i metodi getter e setter per la gestione delle informazioni; inoltre sarà possibile creare ed eliminare istanze tramite i metodi new e delete.

4.3.1 Diagramma BCE - Amministratore

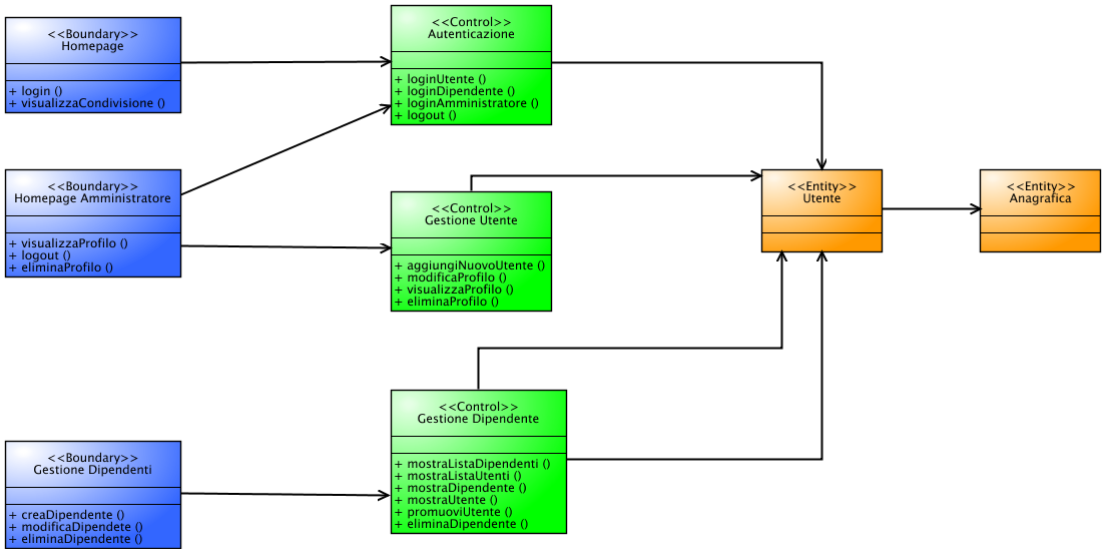

Figura 8: Diagramma BCE - Amministratore

4.3.2 Diagramma BCE - Utente e Dipendente

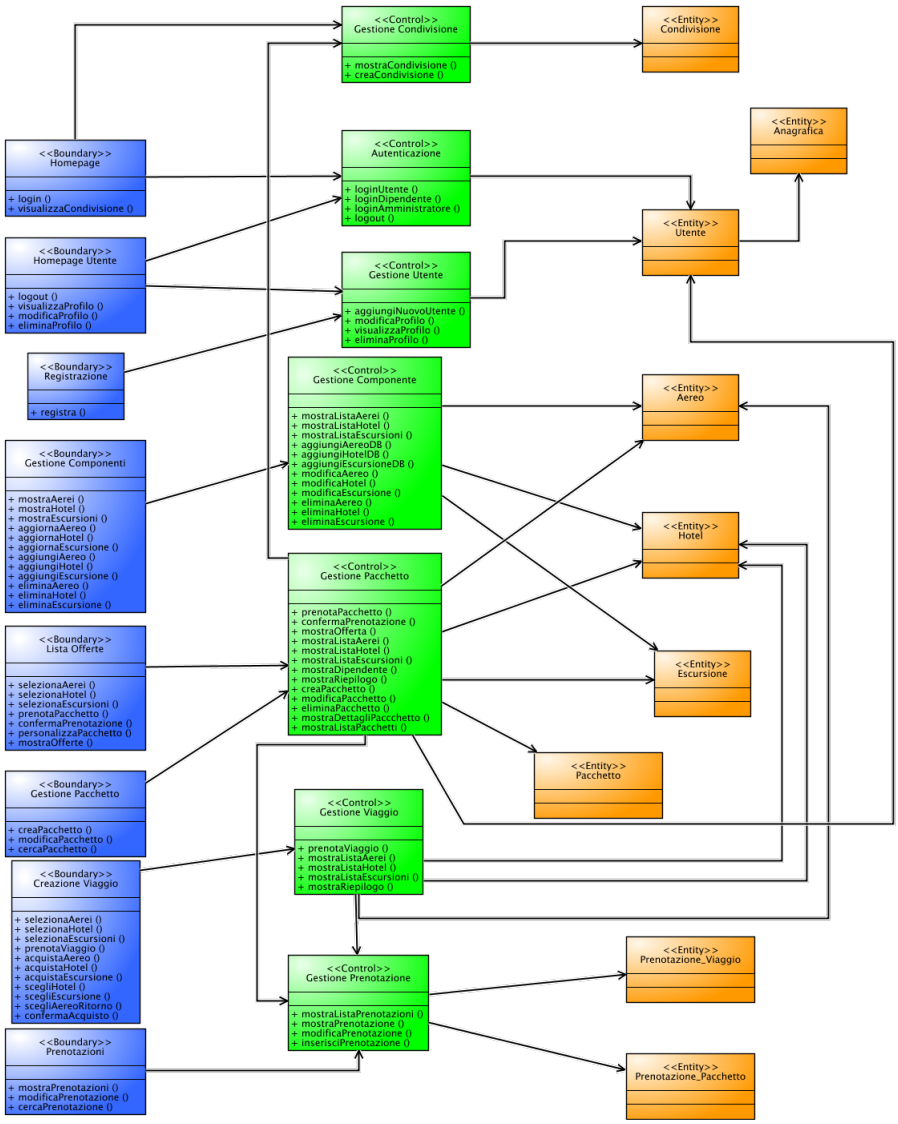

Figura 9: Diagramma BCE - Utente e Amministratore

Si vuole sottolineare che la «Boundary» *Homepage Utente* raggruppa le screen *Homepage Dipendente* e *Homepage Utente*.

4.3.3 Diagrammi di sequenza

In questa sezione vengono riportati alcuni diagrammi di sequenza per evidenziare il flusso degli eventi in alcuni casi d'uso.

Il primo diagramma presentato mostra la modifica del profilo di un utente registrato. Dalla boundary *Homepage* effettua il login con l'ausilio del control *Autenticazione* che verifica la correttezza dei dati inseriti interrogando il database. A seguito dell'accesso, l'utente visualizza il proprio profilo.

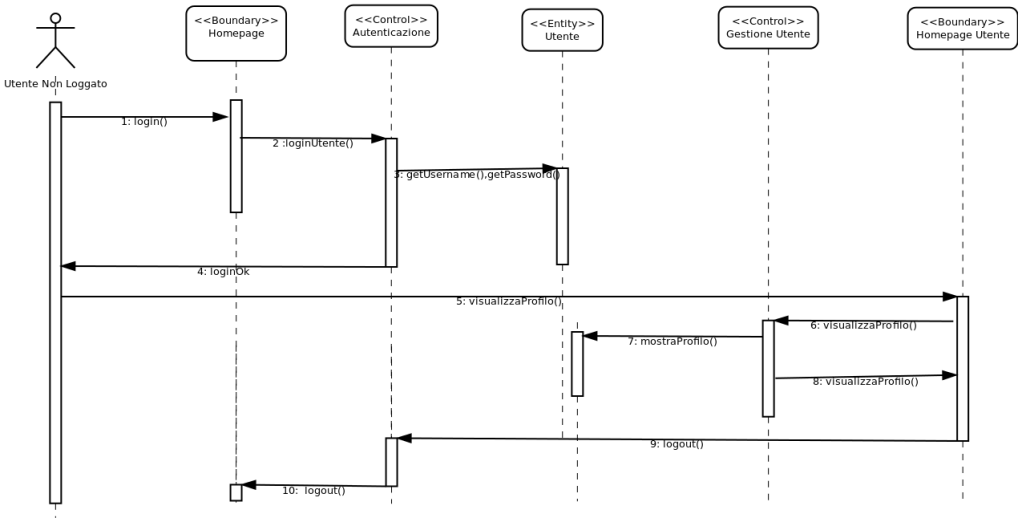

Figura 10: Diagramma di Sequenza - Utente

Il secondo diagramma mostra l'acquisto di un pacchetto da parte dell'utente. In questo sequence si è ipotizzato il fatto che il cliente non richieda la condivisione.

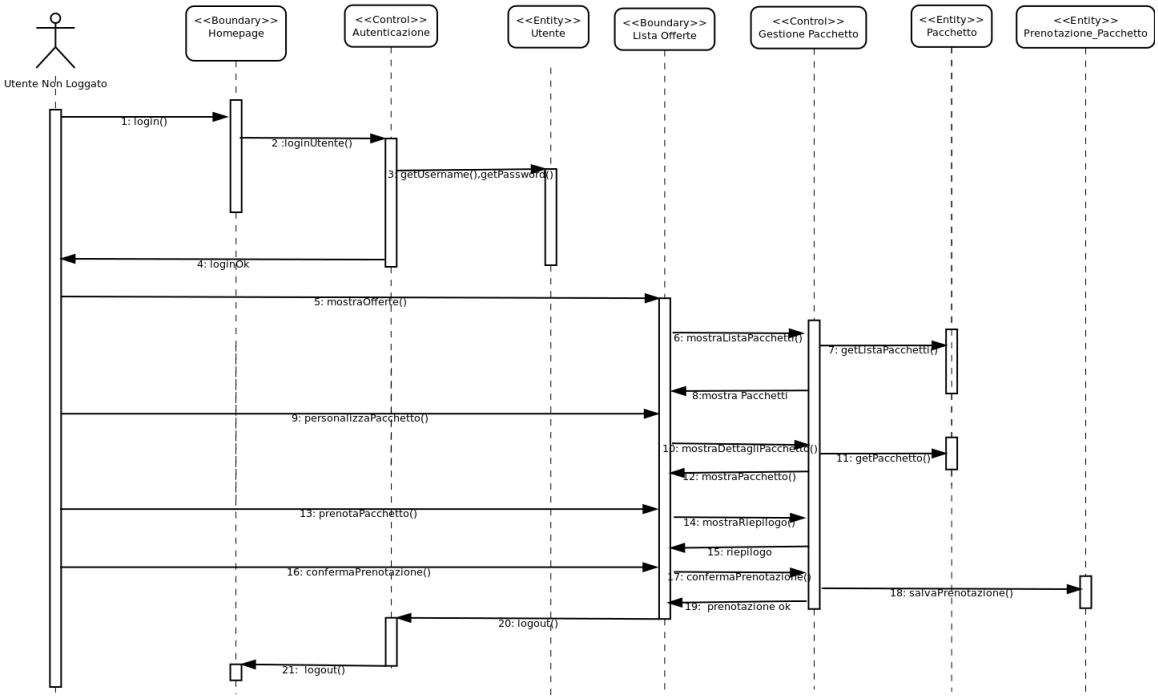

Figura 11: Diagramma di Sequenza - Utente - Acquisto Pacchetto

Il terzo sequence diagram è riferito al dipendente. Dopo aver effuato il login tramite l'ausilio dei componenti descritto nel diagramma precendente, procede con la creazione di un nuovo pacchetto da inserire nel database.

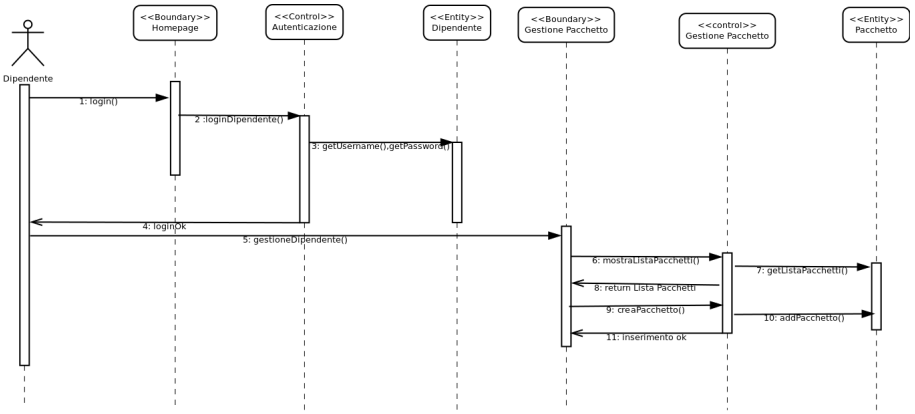

Figura 12: Diagramma di Sequenza - Dipendente

L'ultimo sequence diagram è riferito all'amministratore. Dopo aver effettuato il login, cerca un utente ed lo eleva al ruolo di dipendente.

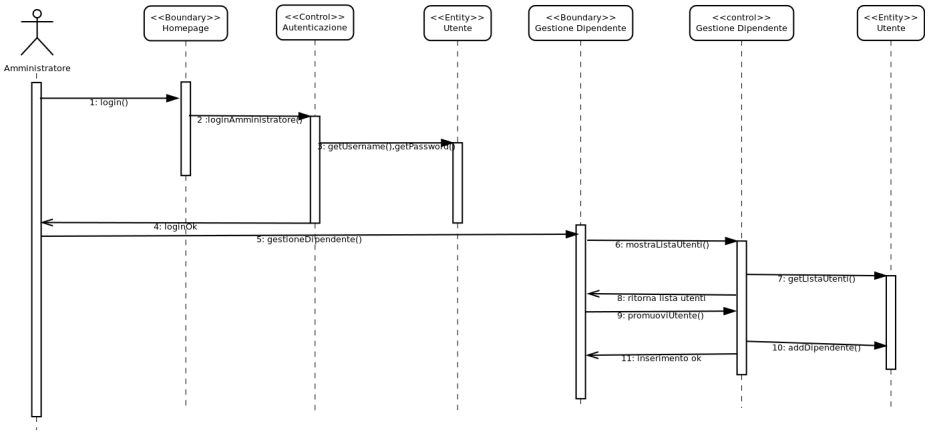

Figura 13: Diagramma di Sequenza - Amministratore