POLITECNICO
DI MILANO

Corso di Laurea Magistrale in Ingegneria Informatica

Progetto di Ingegneria del Software 2:
SWIMv2

Design Document

Autori:

Nicola Sturari
Carlo Sicignano
Simone Mazzoni

Docente:
Raffaela Mirandola

21 dicembre 2012

Indice

Introduzione

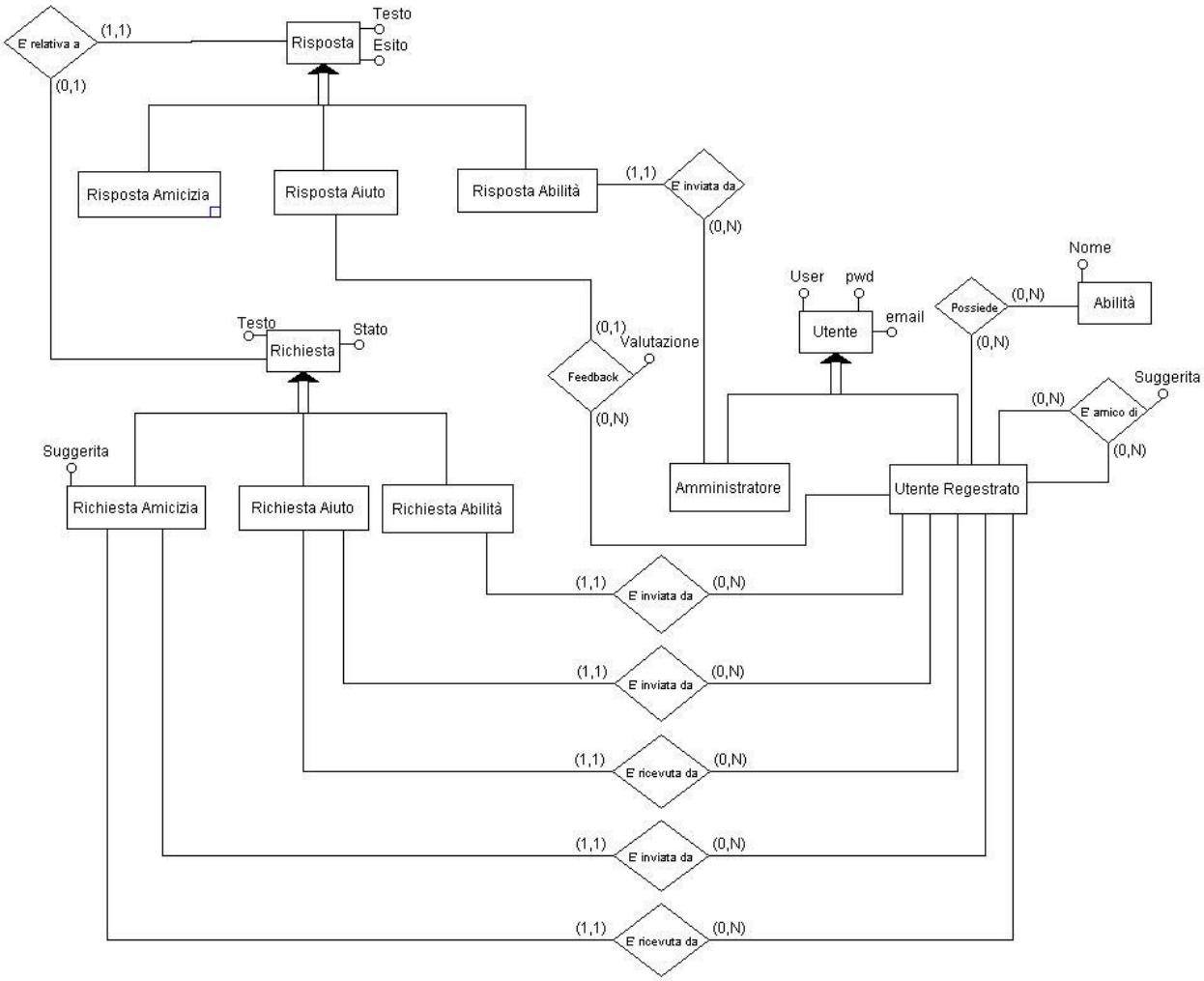
Nella fase di design del sistema SWIMv2 abbiamo evidenziato due aree di lavoro principali che si riflettono nella struttura di questo documento.
I dati relativi alle comunicazioni e richieste tra gli utenti nonché le loro informazioni personali, dalle abilità alle credenziali per l'accesso, necessitano di un database di supporto per garantirne persistenza e permetterne il recupero in modo efficiente e affidabile.
La prima sezione del documento si occupa quindi del design dei dati, partendo dal modello concettuale (diagramma ER) per arrivare al modello logico da cui segue lo schema del database.
Nella seconda parte vengono spiegate le scelte architetturali effettuate per quando riguarda la parte funzionale del sistema, in particolare la struttura a livelli e le tecnologie adottate. Sono inoltre presentati i modelli di navigazione per descrivere l'esperienza del sistema dal punto di vista dell'utente con le differenti schermate visualizzate e le funzioni che permettono di navigare da una all'altra.
Infine sono presentati i componenti che dovranno essere sviluppati in fase di implementazione, senza la pretesa di una definizione già completa, e le loro interazioni attraverso dei diagrammi di sequenza.

Design dei dati

Modello Entity Relationship

Le principali entità presenti all'interno del sistema sono gli utenti, le richieste e le risposte.
Per quanto riguarda gli utente ne distinguiamo due tipologie: normali utente registrati e amministratori. Nonostante nel RASD sia supposta la presenza di un unico amministratore, ci è sembrato ragionevole non inserire questa limitazione in fase di design perché ci potrebbe essere la necessità, in una futura espansione del sistema, di avere più amministratori che si occupino di diverse funzioni. Per gli utenti del sistema non registrati non c'è alcuna informazione importante da memorizzare come già chiarito nella fase di analisi dei requisiti.
Le entità RICHIESTA e RISPOSTA si specializzano ciascuna a seconda del tipo che può essere di aiuto, di amicizia o di aggiunta abilità, ad un dato momento nel sistema ogni richiesta può avere collegata 0 o 1 risposta.
Un vincolo non espresso dal diagramma è che la tipologia di risposta deve corrispondere a quella della richiesta, altrimenti non è possibile che siano in relazione (sarebbe stato possibile rappresentare 3 diverse relazioni nel diagramma ma si voleva evitare di appesantire troppo il modello per esprimere un vincolo che risulta .
Si è scelto inoltre di modellare le abilità dichiarate da un utente come un'entità separata collegata dalla relationship molti-a-molti POSSIEDE.
Tra le relationship risultano facilmente comprensibili quelle di invio e ricezione tra richieste e utenti registrati. L'unica nota rilevante è che la richiesta di aggiunta nuova abilità non è ricevuta in modo specifico da qualcuno ma dal sistema che secondo certi criteri permetterà ad un amministratore di inviare la relativa risposta.
L'amicizia tra due utenti registrati è modellata come una relazione riflessiva n ad n, mentre il feeback su un aiuto ricevuto è rappresentato come una relazione che collega l'utente registrato alla risposta d'aiuto.
Un altro vincolo non espresso dal diagramma è che l'utente che invia il feedback deve essere lo stesso che ha inviato la richiesta d'aiuto relativa a quella risposta.
Nel modello per semplicità non sono presenti tutti gli attributi che saranno inseriti alla fine (ad esempio tutte le informazioni riguardanti l'utente), ma solo quelli necessari per capire il ruolo delle diverse entità.
Infine gli attributi che necessitano di una spiegazione particolare sono:

- "suggerita" nell'entità "Richiesta Amicizia" e nella relazione "E' amico di" è un attributo di tipo booleano che serve a memorizzare rispettivamente se la richiesta è stata inviata su suggerimento del sistema e se l'amicizia tra due utenti deriva da un suggerimento
- "stato" nell'entità "Richiesta" è un attributo booleano che indica se la richiesta in questione ha avuto risposta o no; benché questo attributo sia ridondante si è scelto di inserirlo perché permette di individuare con una interrogazione molto semplice le richieste a cui un utente deve ancora rispondere.


E' relativa a
(1,1)
(0,1)
Risposta
Testo
Esito
Risposta Amicizia
Risposta Aiuto
Risposta Abilità
(1,1)
E' inviata da
(0,N)
Nome
User
pwd
Possiede
(0,N)
Abilità
Utente
email
(0,N)
Testo
Richiesta
Stato
(0,1)
Feedback
Valutazione
(0,N)
Suggerita
(0,N)
E' amico di
(0,N)
Suggerita
Richiesta Amicizia
Richiesta Aiuto
Richiesta Abilità
Amministratore
Utente Regestrato
(1,1)
E' inviata da
(0,N)
(1,1)
E' inviata da
(0,N)
(1,1)
E' ricevuta da
(0,N)
(1,1)
E' inviata da
(0,N)
(1,1)
E' ricevuta da
(0,N)

Ristrutturazione del modello ER

Al fine di tradurre in uno schema logico il modello concettuale si rende necessaria una ristrutturazione che prevede la risoluzione delle generalizzazioni e la trasformazione in entità (con associati collegamenti) di alcune relationships.
Le modifiche applicate allo schema ER sono le seguenti:

- L'entità RICHIESTA viene accorpata nelle tre entità figlie, stessa strategia per l'entità RISPOSTA visto che in entrambi i casi l'entità padre presenta un numero di relazioni inferiore e la separazione permette anche di specificare in modo esplicito la corrispondenza di tipologia (amicizia, aiuto, abilità) tra richiesta e relativa risposta.
- L'entità UTENTE essendo poco significativa viene eliminata e i suoi attributi inseriti in AMMINISTRATORE e UTENTE REGISTRATO. Un vantaggio di questa decisione consiste nel poter garantire una migliore separazione tra il normale utilizzo del sistema e le operazioni di amministrazione.
- La relationship ricorsiva E' AMICO DI viene reificata nell'entità AMICIZIA con l'aggiunta di 2 relazionships di collegamento verso UTENTE.
- La relationship FEEDBACK viene trasformata in un'omonima entità e non viene conservato il collegamento con UTENTE REGISTRATO ma solo l'associazione con la RISPSOTA D'AIUTO perché l'informazione relativa a chi ha inviato il feedback risulta ricavabile indirettamente dalla risposta.
- Per tutte le entità viene aggiunto un campo intero "Id" univoco che costituisce la chiave primaria. Questa decisione si rivela necessaria in alcuni casi nei quali nessuna combinazione degli attributi costituisce una chiave come nel caso dell'entità RICHIESTA, si è deciso di applicarla anche a tutte le altre entità per avere una gestione più semplice e una maggiore uniformità.

Sarebbe stato possibile accorpare le entità RISPOSTA e RICHIESTA in una sola con l'aggiunta degli attributi esito_risposta e testo_risposta nella seconda, tuttavia in un'ottica di futura espandibilità e modifica del sistema questa scelta non avrebbe permesso, ad esempio, di supportare richieste di aiuto inserite come "annunci" nel sistema, a cui cioè possono rispondere più utenti. Per questo motivo riteniamo che la rappresentazione scelta sia il giusto compromesso tra minimalità e possibilità di generalizzare.
Per brevità non riportiamo il diagramma ER ristrutturato, ma alla fine è presente una rappresentazione dello schema del database in cui sono evidenti le modifiche fatte in questa fase e nella successiva traduzione verso il modello relazionale.

Traduzione verso il modello logico

Semplificato il modello concettuale andiamo a tradurre entity e relationships in relazioni con attributi a fare da chiave esterna per rappresentare i loro collegamenti. Per il progetto abbiamo deciso di nominare tutti gli attributi in foreign key come id_<nome-relazione-referenziata> ove possibile.
Le scelte effettuate per il passaggio alle relazioni non necessitano di molte spiegazioni:

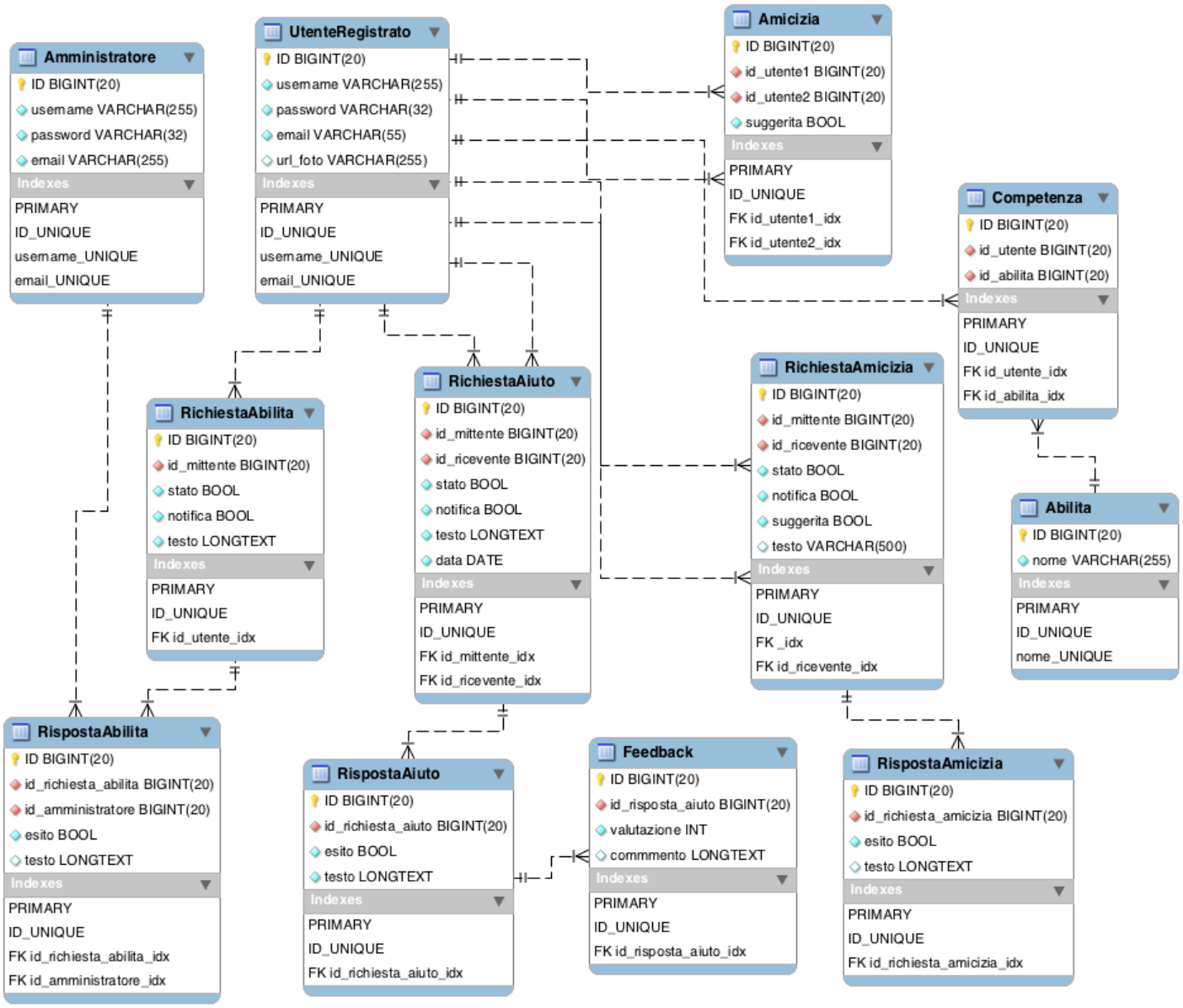
- Le relationships E' INVIATA DA e E' RICEVUTA DA sono mappate negli attributi "id_mittente" e "id_ricevente" per le relazioni che rappresentano i diversi tipi di risposte e richieste.
- La relationship E' RELATIVA A viene rappresentata dall'attributo "id_richiesta" nella relazione RISPOSTA. Questa scelta è la più adatta perché la cardinalità della relationship è (1,1) dal lato RISPOSTA, mentre dall'altra parte c'è l'opzionalità.
- La relazione AMICIZIA contiene gli attributi "id_utente1" e "id_utente2" per rappresentare i 2 amici. Per ogni amicizia saranno presenti le 2 tuple (A,B) e (B,A).
- La relationship POSSIEDE diventa la relazione di join COMPETENZA tra UTENTE REGISTRATO e ABILITA

Risultato della traduzione fisica della base di dati:

Amministratore(ID, user, pwd, email)
UtenteRegestrato(ID, user, pwd, email)
RispostaAmicizia(ID, testo, esito, id_richiesta)
RispostaAiuto(ID, testo, esito, id_richiesta)
RispostaAbilità(ID, testo, esito, id_richiesta, id_amministratore)
RichiestaAmicizia(ID, testo, suggerita, id_mittente, id_ricievente, stato, notifica)
RichiestaAiuto(ID, testo, id_mittente, id_ricievente, data, stato, notifica)
RichiestaAbilità(ID, testo, id_mittente, id_ricievente, stato, notifica)
Amicizia(ID,id_utente1, id_utente2, suggerita)
Feedback(ID, valutazione, id_risposta_aiuto, commento)
Abilità(ID, Nome)
Competenza(ID, id:utente, id_abilità)

Note finali:

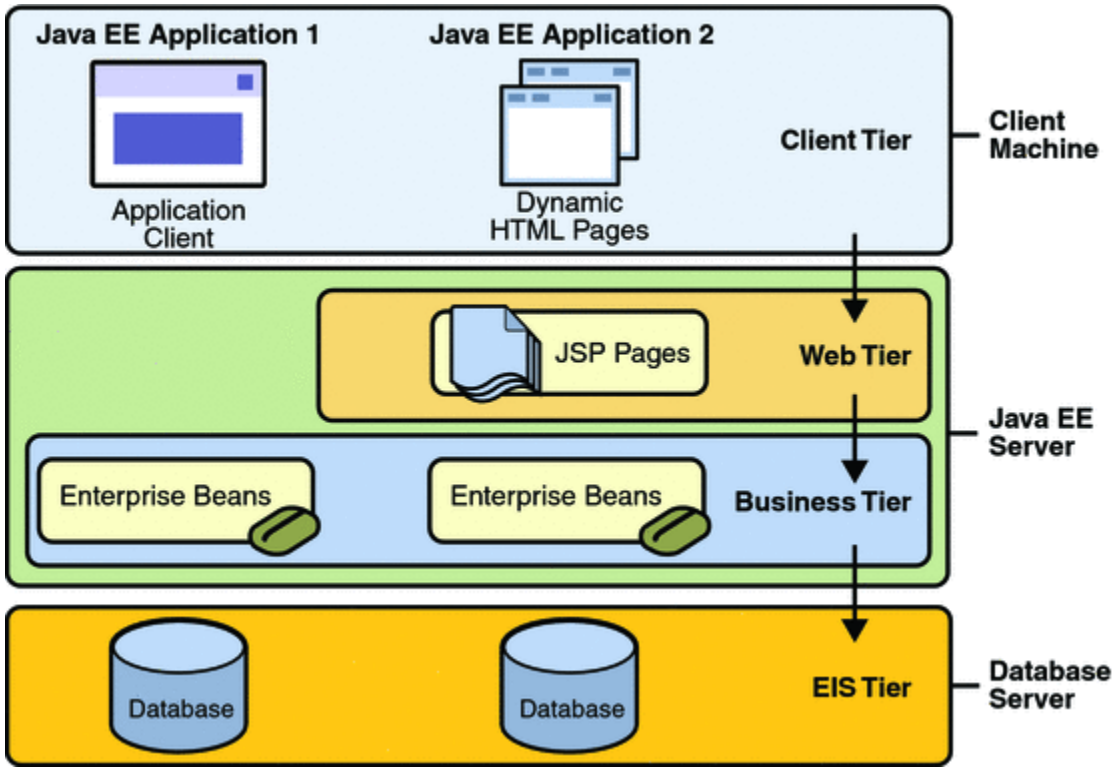
- la pwd dell'UtenteRegistrato verrà memorizzata come MD5, quindi una sequenza di 32 cifre esadecimali; perciò il tipo dell'attributo sarà una stringa di 32 caratteri
- l'attributo "notifica" nelle relazioni relative alle richieste è un booleano che indica se l'utente ha visualizzato la risposta relativa; il suo valore è quindi obbligatoriamente *false* fintanto che l'attributo "stato" è *false*, è messo a true quando l'utente visualizza la risposta.


Amministratore
ID BIGINT(20)
username VARCHAR(255)
password VARCHAR(32)
email VARCHAR(255)
Indexes
PRIMARY
ID_UNIQUE
username_UNIQUE
email_UNIQUE
UtenteRegistrato
ID BIGINT(20)
username VARCHAR(255)
password VARCHAR(32)
email VARCHAR(55)
url_foto VARCHAR(255)
Indexes
PRIMARY
ID_UNIQUE
username_UNIQUE
email_UNIQUE
Amicizia
ID BIGINT(20)
id_utente1 BIGINT(20)
id_utente2 BIGINT(20)
suggerita BOOL
Indexes
PRIMARY
ID_UNIQUE
FK id_utente1_idx
FK id_utente2_idx
Competenza
ID BIGINT(20)
id_utente BIGINT(20)
id_abilita BIGINT(20)
Indexes
PRIMARY
ID_UNIQUE
FK id_utente_idx
FK id_abilita_idx
RichiestaAbilita
ID BIGINT(20)
id_mittente BIGINT(20)
stato BOOL
notifica BOOL
testo LONGTEXT
Indexes
PRIMARY
ID_UNIQUE
FK id_utente_idx
RichiestaAiuto
ID BIGINT(20)
id_mittente BIGINT(20)
id_ricevente BIGINT(20)
stato BOOL
notifica BOOL
testo LONGTEXT
data DATE
Indexes
PRIMARY
ID_UNIQUE
FK id_mittente_idx
FK id_ricevente_idx
RichiestaAmicizia
ID BIGINT(20)
id_mittente BIGINT(20)
id_ricevente BIGINT(20)
stato BOOL
notifica BOOL
suggerita BOOL
testo VARCHAR(500)
Indexes
PRIMARY
ID_UNIQUE
FK _idx
FK id_ricevente_idx
Abilita
ID BIGINT(20)
nome VARCHAR(255)
Indexes
PRIMARY
ID_UNIQUE
nome_UNIQUE
RispostaAbilita
ID BIGINT(20)
id_richiesta_abilita BIGINT(20)
id_amministratore BIGINT(20)
esito BOOL
testo LONGTEXT
Indexes
PRIMARY
ID_UNIQUE
FK id_richiesta_abilita_idx
FK id_amministratore_idx
RispostaAiuto
ID BIGINT(20)
id_richiesta_aiuto BIGINT(20)
esito BOOL
testo LONGTEXT
Indexes
PRIMARY
ID_UNIQUE
FK id_richiesta_aiuto_idx
Feedback
ID BIGINT(20)
id_risposta_aiuto BIGINT(20)
valutazione INT
commmento LONGTEXT
Indexes
PRIMARY
ID_UNIQUE
FK id_risposta_aiuto_idx
RispostaAmicizia
ID BIGINT(20)
id_richiesta_amicizia BIGINT(20)
esito BOOL
testo LONGTEXT
Indexes
PRIMARY
ID_UNIQUE
FK id_richiesta_amicizia_idx

Design dell’applicazione

Livelli Architetturali

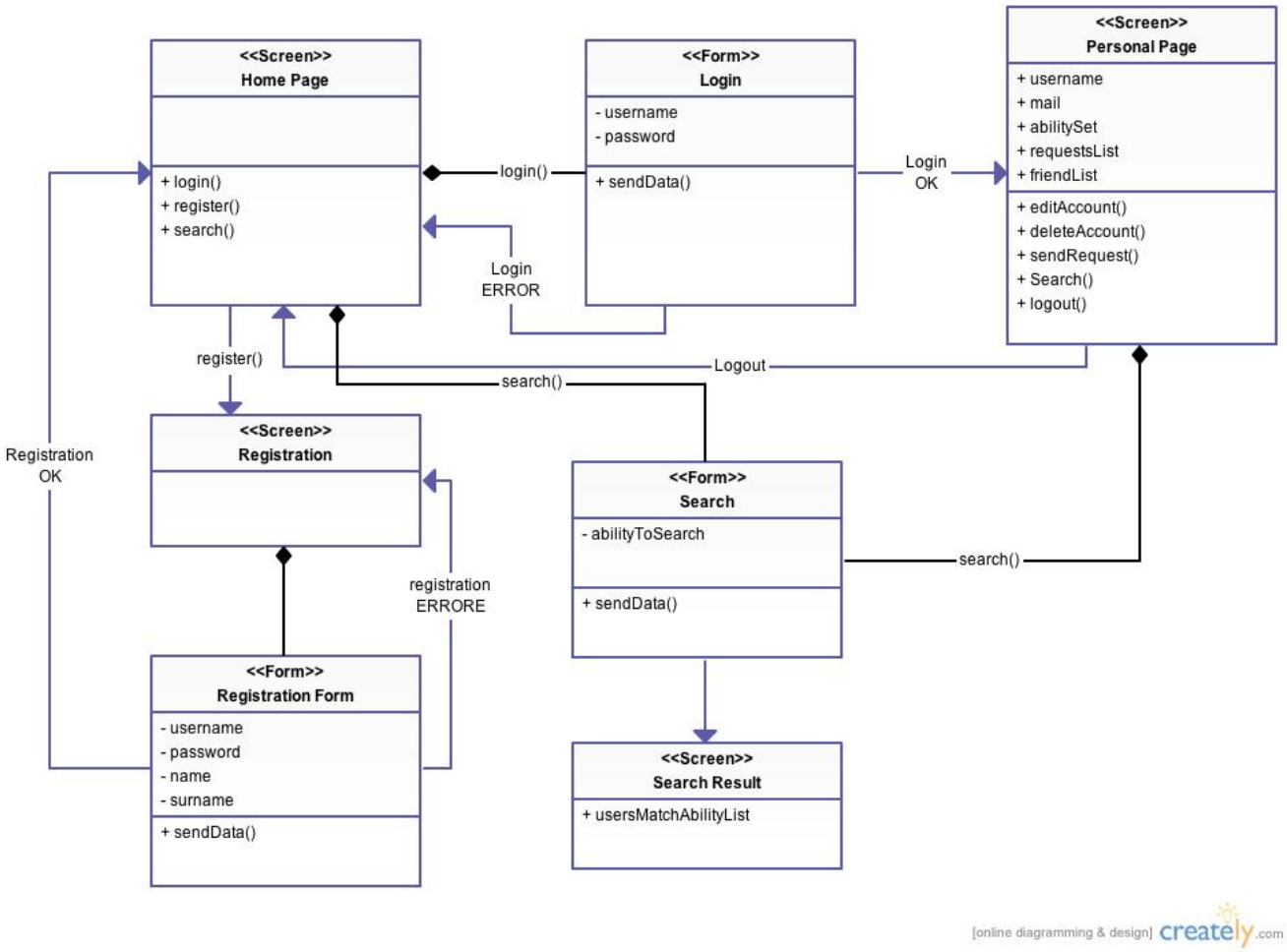
L’infrastruttura di supporto da utilizzare per realizzare il sistema è Java Enterprise Edition, in particolare verrà utilizzata una sua implementazione, JBoss che contiene il web server Apache, il motore di servlet Tomcat, e la gestione della logica di business.
Dal lato utente non si prevede l’installazione di un’applicazione client specializzata, ma l’accesso avverrà attraverso un browser e richieste http al nostro server.
Sulla stessa macchina il server JBoss 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.



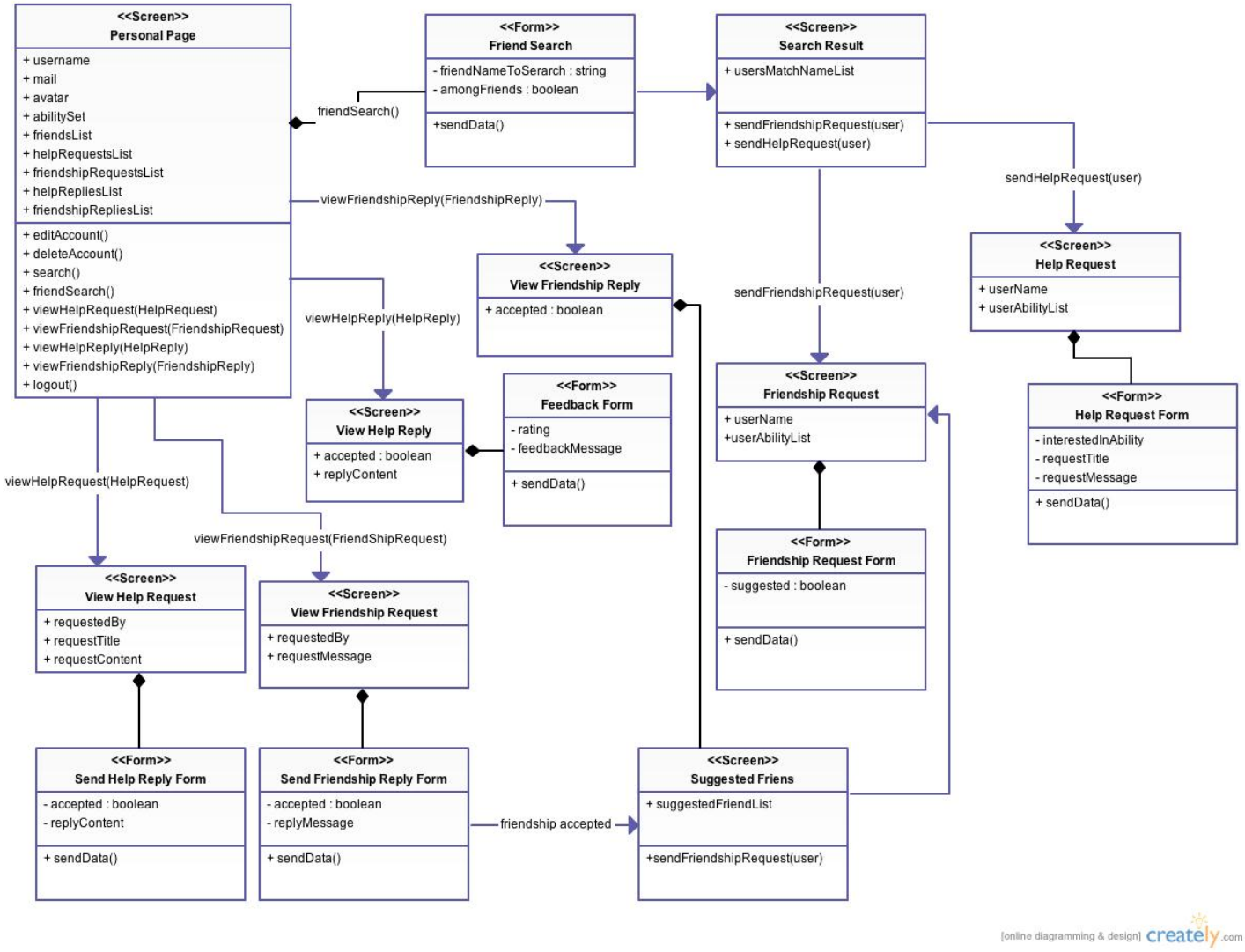
Modelli di Navigazione

Home Page: diagramma rappresentante il modello di navigazione di un utente di qualsiasi tipo nella pagina principale del sistema. Viene illustrata in particolare la possibilita di effettuare il LOGIN (se già si è registrati) o una nuova registrazione per usufruire di tutte le funzionalità del sistema.

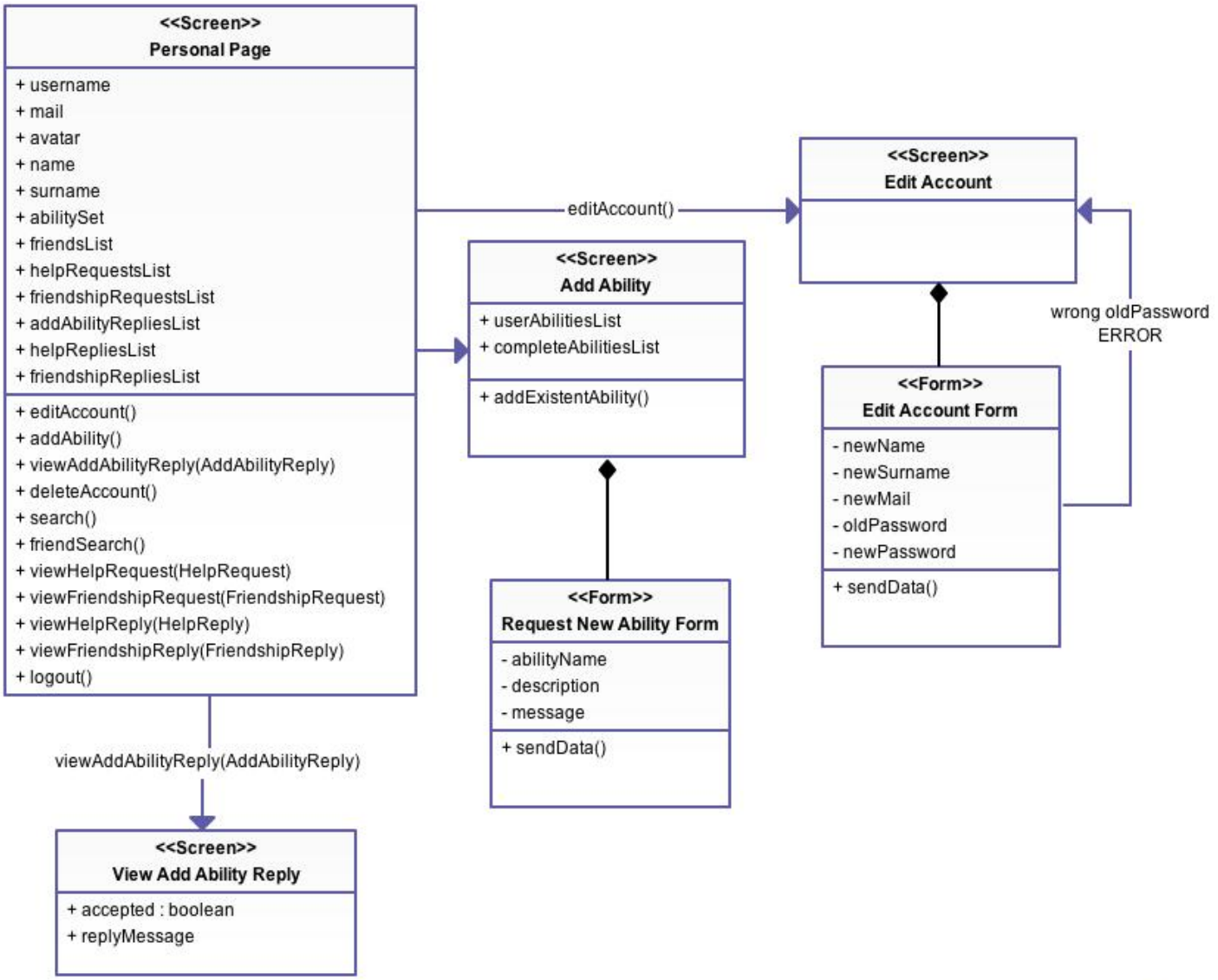
Una volta che un utente si è loggato nel sistema avrà accesso ad una sua pagina personale dalla quale potrà svolgere diverse operazioni come per esempio la modifica dei propri dati personali o l'aggiornamento del proprio set di abilità.



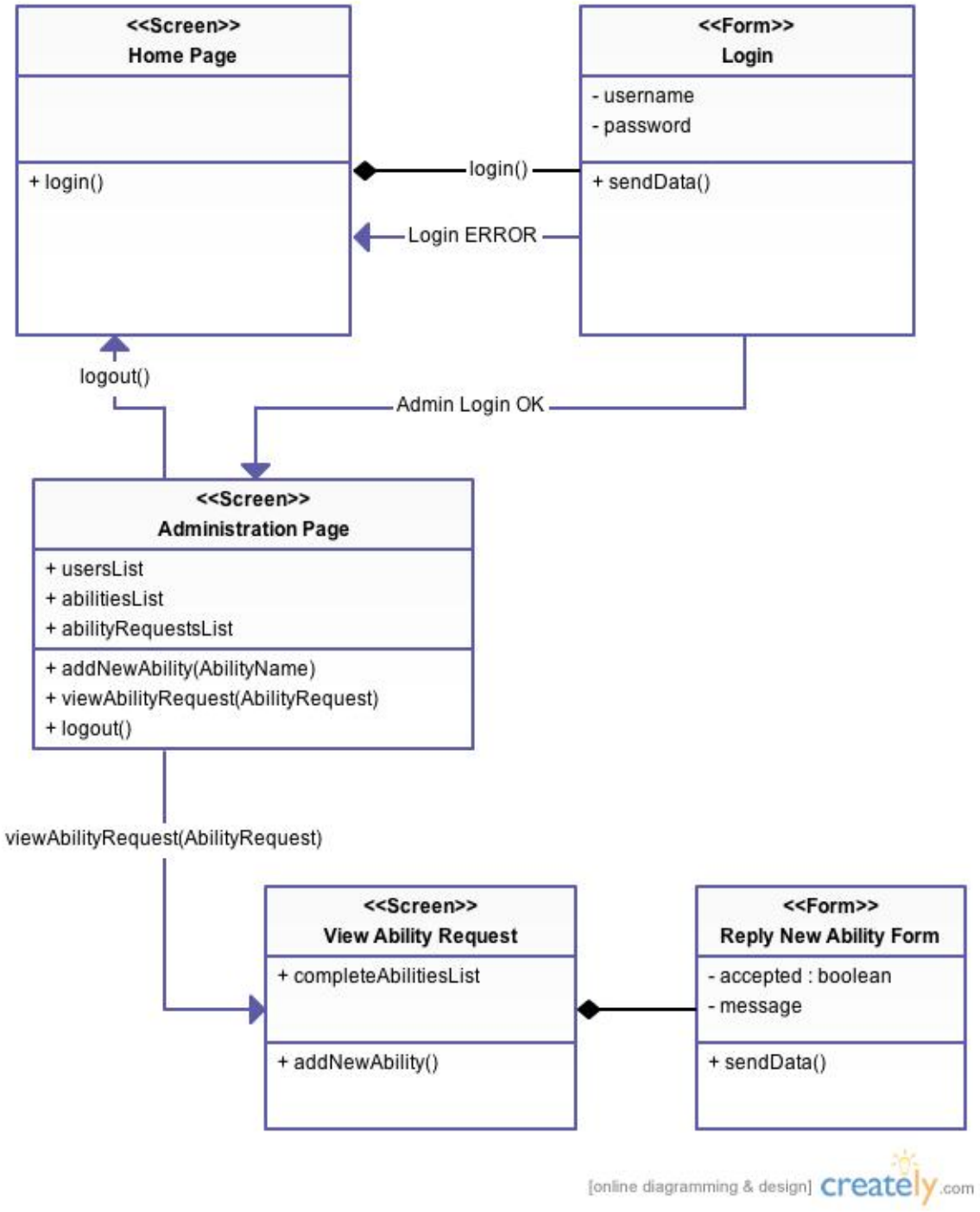
Personal Page: diagramma che illustra tutte le funzionalità a cui ha accesso un utente registrato una volta effettuata la procedura di LOGIN. Il diagramma totale è stato diviso in due diagrammi per praticità di lettura. In particolare nel primo diagramma si vuole far notare che la funzione di RICERCA DI AIUTO è una diretta conseguenza della ricerca di un UTENTE (amico o non) che abbia nel proprio set l'abilità di cui l'utente ha bisogno. Una volta effettuata la ricerca e trovati degli utenti in grado di soddisfarla, l'utente può scegliere a quale inviare la sua richiesta di aiuto ed eventualmente inviargli anche una RICHIESTA DI AMICIZIA.
Ai fini di implementare un meccanismo interno al sistema di notifiche in stile social network, si è deciso di gestire le RICHIESTE/RISPOSTE ricevute come delle liste in modo tale che possano essere presentate e gestite dall'utente ogni qualvolta entra nel sistema.



Il secondo diagramma illustra più nello specifico le funzionalità che permettono all'utente di agire sul proprio profilo per modificarlo (ed eventualmente cancellarlo), oppure aggiungere nuove abilità tra quelle presenti nel proprio set.

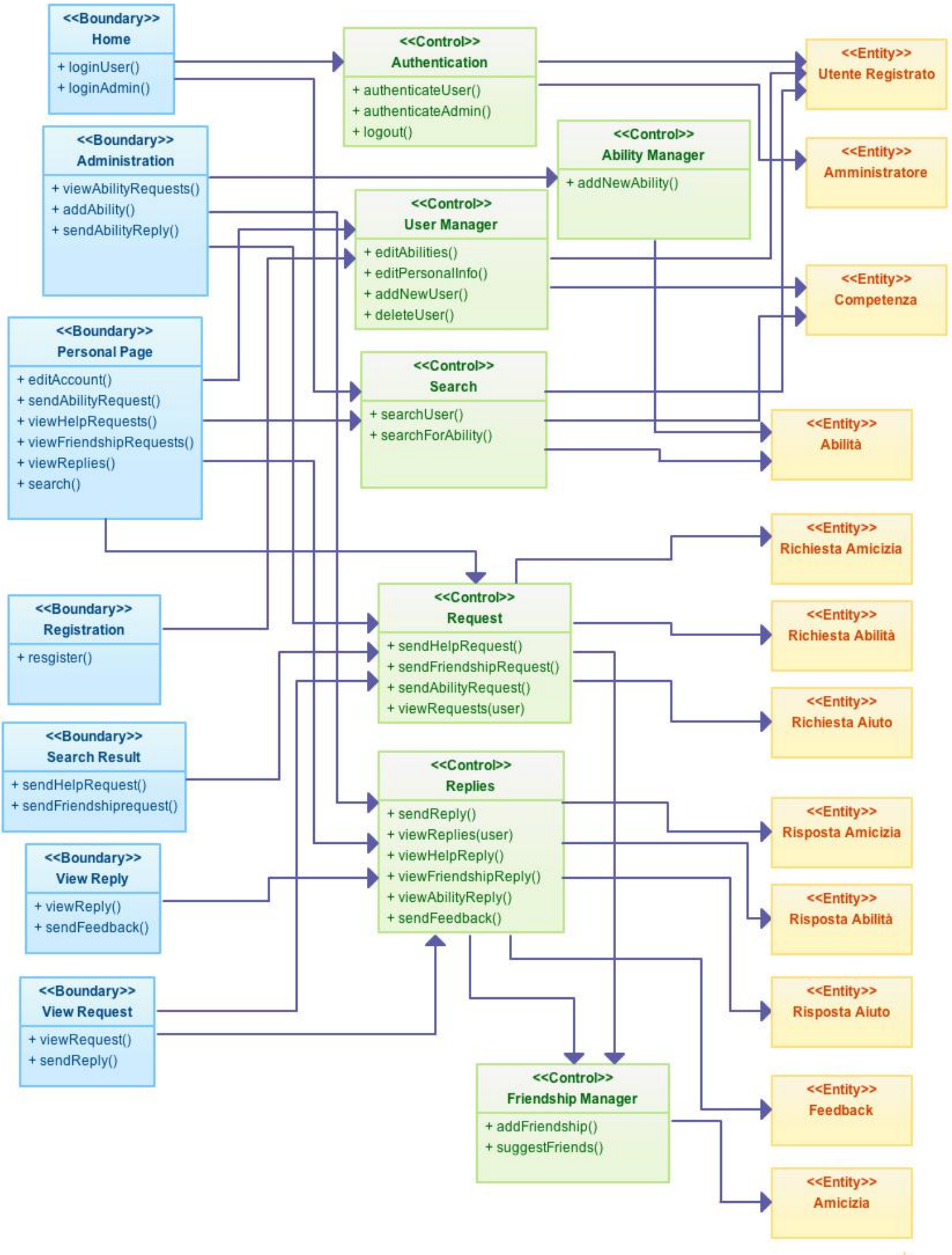


Administration: Si è infine rappresentato il diagramma di navigazione dell'utente di tipo amministratore in quanto dotato di particolari funzionalità riservate solo a lui.
Allo stesso modo delle richieste degli utenti, anche per l'amministratore si è scelto di rappresentare le RICHIESTE DI AGGIUNTA DI NUOVE ABILITA' come una lista in modo da visualizzarle in blocco ad ogni accesso al sistema.



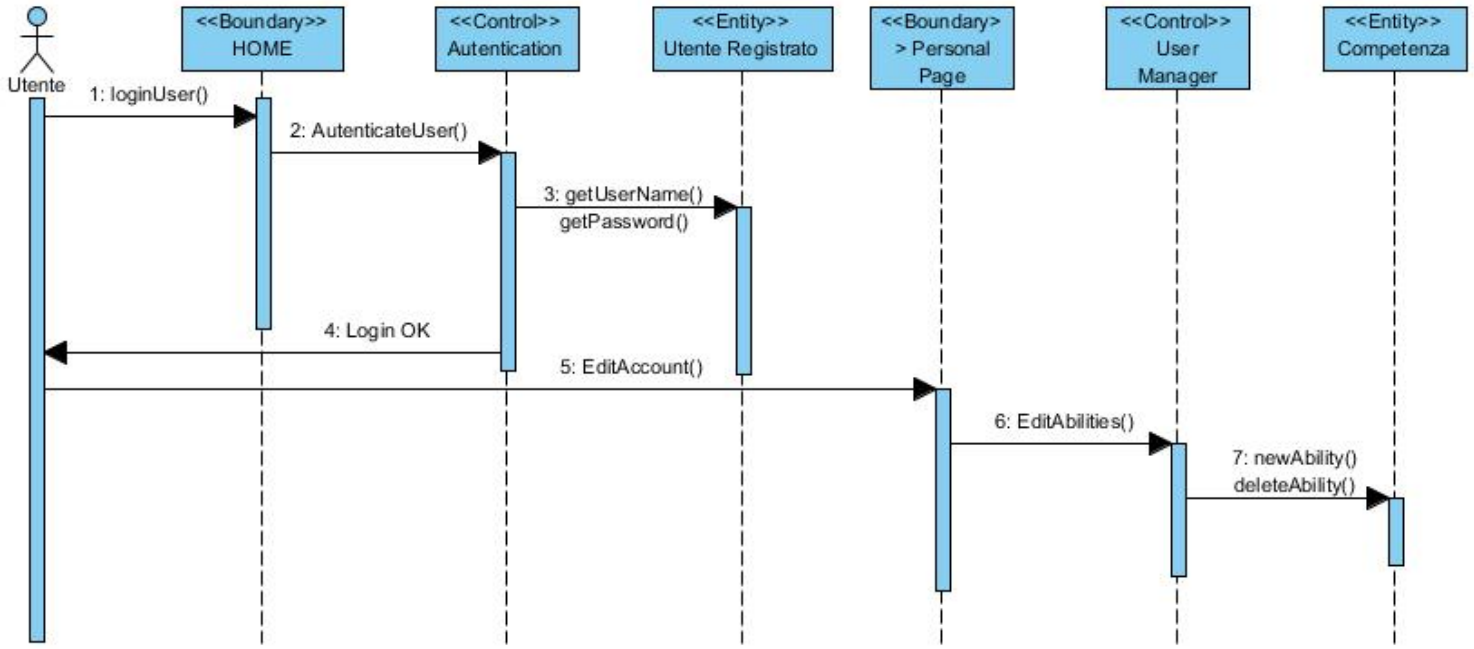
Componenti

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 nel 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 oggeti 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.
Abbiamo scelto di rappresentare la suddivisione in componenti attraverso gli stereotipi UML Boundary, Control e Entity. Le <Boundary> rappresentano gli insiemi di funzionalità offerte all’utente coerentemente ai modelli di navigazioni proposti. Gli oggetti <Control> sono i mediatori tra l’interfaccia utente e i dati che sono modellati come <Entity>.
Le interfacce presentate all’utente, quindi le pagine HTML e JSP nel progetto finale, costituiranno l’implementazione delle <Boundary>. Gli elementi <Control> e <Entity> saranno realizzati rispettivamente con SessionBeans e EntityBeans.
I campi degli oggetti <Entity> non sono presenti nello schema che segue, ma si possono considerare gli stessi dell’ultimo diagramma presentato nella sezione riguardante il design dei dati. Per i metodi, sono anch’essi omessi, ma nell’implementazione si avranno setter per i campi modificabili, getter per tutti i campi, new e delete.

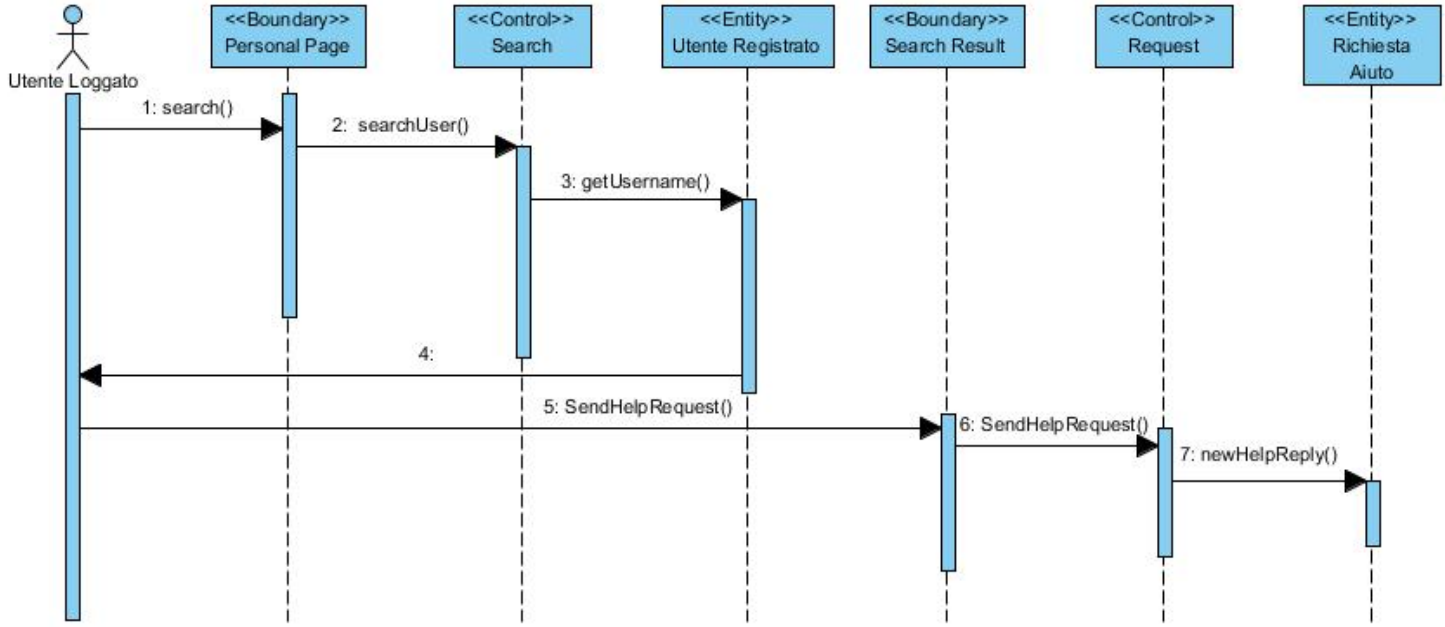

<<Boundary>>
Home
+ loginUser()
+ loginAdmin()
<<Control>>
Authentication
+ authenticateUser()
+ authenticateAdmin()
+ logout()
<<Entity>>
Utente Registrato
<<Boundary>>
Administration
+ viewAbilityRequests()
+ addAbility()
+ sendAbilityReply()
<<Control>>
Ability Manager
+ addNewAbility()
<<Entity>>
Amministratore
<<Control>>
User Manager
+ editAbilities()
+ editPersonalInfo()
+ addNewUser()
+ deleteUser()
<<Entity>>
Competenza
<<Boundary>>
Personal Page
+ editAccount()
+ sendAbilityRequest()
+ viewHelpRequests()
+ viewFriendshipRequests()
+ viewReplies()
+ search()
<<Control>>
Search
+ searchUser()
+ searchForAbility()
<<Entity>>
Abilità
<<Entity>>
Richiesta Amicizia
<<Boundary>>
Registration
+ resgister()
<<Control>>
Request
+ sendHelpRequest()
+ sendFriendshipRequest()
+ sendAbilityRequest()
+ viewRequests(user)
<<Entity>>
Richiesta Abilità
<<Entity>>
Richiesta Aiuto
<<Boundary>>
Search Result
+ sendHelpRequest()
+ sendFriendshiprequest()
<<Control>>
Replies
+ sendReply()
+ viewReplies(user)
+ viewHelpReply()
+ viewFriendshipReply()
+ viewAbilityReply()
+ sendFeedback()
<<Entity>>
Risposta Amicizia
<<Boundary>>
View Reply
+ viewReply()
+ sendFeedback()
<<Entity>>
Risposta Abilità
<<Boundary>>
View Request
+ viewRequest()
+ sendReply()
<<Entity>>
Risposta Aiuto
<<Control>>
Friendship Manager
+ addFriendship()
+ suggestFriends()
<<Entity>>
Feedback
<<Entity>>
Amicizia

Diagrammi di sequenza

Il primo diagramma riguarda il login di un utente registrato, dalla <boundary> home, coinvolgendo il <control> Authentication che per verificare le credenziali fornite chiede i dati all'<Entity> Utente Registrato. Successivamente l'utente dalla <Boundary> Personal Page modifica le sue competente aggiungendo o rimuovendo abilità.



Questo diagramma mostra l'invio di una richiesta d'aiuto da parte di un utente che abbia già effettuato il login. Partendo dalla sua pagina personale invoca la funzione di ricerca che viene trasmessa al relativo <Control> che a sua volta effettua un'interrogazione all'<Entity> per ottenere i dati richiesti.
L'utente quindi attraverso dalla <Boundary> Search Result invia la richiesta di aiuto che viene inserita nel database.



In quest’ultimo diagramma mostriamo una sessione di amministrazione: dopo avere effettuato il login con una funzione specifica che va a conivolgere l’<Entity> Amministratore, dalla <Boundary> Administration visualizza una richiesta di abilità presente, decide di aggiungere una nuova abilità e invia la risposta all’utente.