UNIVERSITÀ DEGLI STUDI DI SALERNO

Ingegneria del Software

ODD – Object Design Document

ANNO ACCADEMICO 2017/2018

Versione 1.2

Top Manager

Nome
Prof. De Lucia Andrea
Prof. Francese Rita

Partecipanti

Nome	Matricola
Casillo Francesco	0512103760
Ferrentino Alfredo	0512103714

Revision History

Data	Versione	Descrizione	Autore
27/06/2018	1.0	Stesura del documento	Membri del Team
28/06/2018	1.1	Revisione del documento	Membri del Team
29/06/2018	1.2	Revisione del documento	Membri del Team

Sommario

1. Introduzione

Dopo aver stilato il documento di Requirements Analysis e il documento di System Design in cui vi è una descrizione sommaria di ciò che sarà il nostro sistema, definendo i nostri obiettivi ma tralasciando gli aspetti implementativi, andiamo ora a stilare il documento di Object Design che ha come obiettivo quello di produrre un modello che sia in grado di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti. In particolar modo, in tale documento si definiscono le interfacce delle classi, le operazioni, i tipi, gli argomenti e le signature dei sottosistemi definiti nel System Design.

Comprensibilità vs Tempo: Il codice del sistema deve essere comprensibile il più possibile, in modo da facilitare testing e possibili modifiche da apportare in futuro. Per rispettare ciò il codice sarà ovviamente accompagnato da commenti volti a semplificare il tutto. Questo comporterà un aumento di tempo di sviluppo del nostro progetto.

Interfaccia vs Usabilità: L'interfaccia grafica è stata realizzata in maniera molto semplice, chiara e concisa, vengono utilizzati i form e pulsanti con lo scopo di rendere semplice l'utilizzo del sistema da parte dell'utente finale.

Sicurezza vs Efficienza: La sicurezza, come descritto nei requisiti non funzionali del Requirements Analysis, rappresenta uno degli aspetti importanti del sistema. Tuttavia, dati i tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su username e password degli utenti.

1.1. Linee guida per la documentazione delle interfacce

Gli sviluppatori dovranno seguire determinate linee guida per la stesura del codice:
Convenzione nominativi:
È buona norma utilizzare nominativi:
• Descrittivi
• Pronunciabili
• Di lunghezza medio-corta
• Abbreviazioni non esagerate
• Utilizzare caratteri consentiti

Variabili:
• I nomi delle variabili devono iniziare con la lettera minuscola, e le parole successive con quella maiuscola. La dichiarazione delle variabili deve essere effettuata ad inizio blocco; ogni riga presenterà una sola variabile per migliorare la comprensione e per lo stesso motivo c'è bisogno di una corretta indentazione.

Metodi:
• I nomi dei metodi devono iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola. Di solito il nome del metodo è costituito da un verbo che identifica un'azione, seguito dal nome di un oggetto.
• I nomi dei metodi per l'accesso e la modifica delle variabili dovranno essere del tipo getNome(), setNome().
• Ai metodi va aggiunta una descrizione, la quale deve essere posizionata prima della dichiarazione del metodo, e deve descriverne lo scopo.

Classi e pagine:
• I nomi delle classi e delle pagine devono iniziare con la lettera maiuscola, e anche le parole successive all'interno del nome devono iniziare con lettera maiuscola. I nomi delle classi e delle pagine devono essere evocativi, in modo da fornire informazioni sullo scopo di quest'ultime.
• La dichiarazione di una classe è caratterizzata da:
    - Dichiarazione di costanti
    - Dichiarazione di variabili di classe
    - Dichiarazione di variabili d'istanza
    - Costruttore
    - Commento e dichiarazione metodi
    - Le variabili non vedono una descrizione in quanto nessuna variabile sarà visibile all'esterno, tutte le nostre variabili sono tenute interne alla classe.

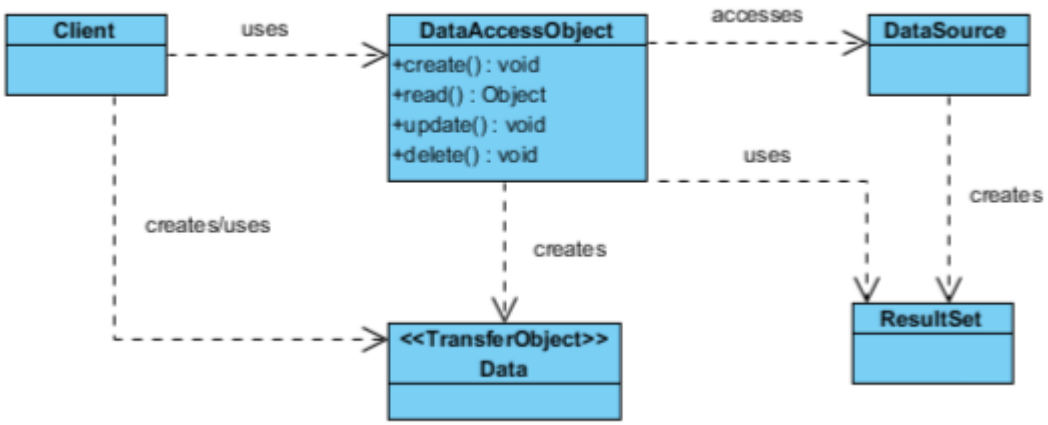
1.2. Definizioni, acronimi e abbreviazioni

Acronimi:
• RAD: Requirements Analysis Document • SDD: System Design Document
• ODD: Object Design Document
Abbreviazioni:
• DB: DataBase

1.3. Riferimenti

• B.Bruegge, A. H. Dutoit, Object Oriented Software Engineering – Using UML, Pattern and Java, Prentice Hail, 3rd edition, 2009.
• Documento RAD Fantabet del progetto Fantabet.
• Documento Gestione dati persistenti_Fantabet del progetto Fantabet.

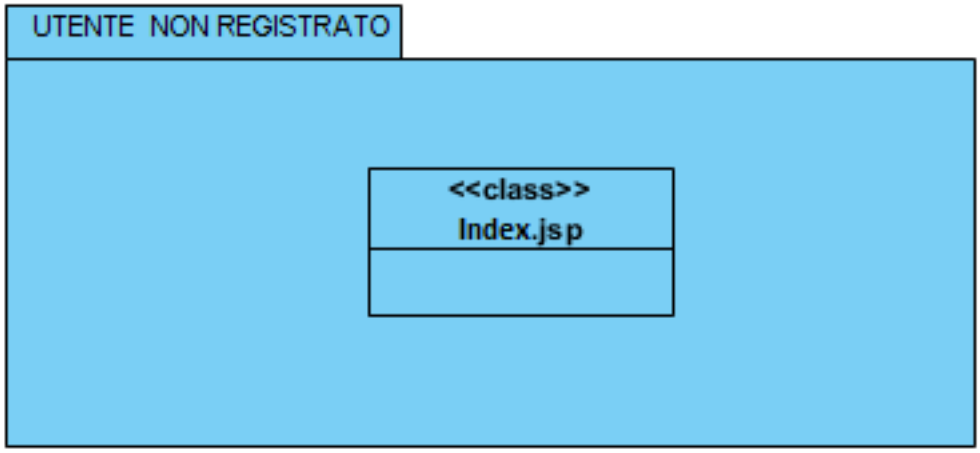
2. Design Pattern



Fantabet fa uso del DAO Pattern. Si tratta di un pattern architetturale per la gestione della persistenza: fondamentalmente, è una classe con relativi metodi che rappresenta un'entità tabellare di un RDBMS, usata per stratificare e isolare l'accesso ad una tabella tramite query (poste all'interno dei metodi della classe) ovvero al data layer da parte della business logic creando un maggiore livello di astrazione e una facile manutenibilità. I metodi del DAO con le rispettive query verranno così richiamati dalle classi della business logic. Il seguente design pattern è stato utilizzato nell'implementazione dei vari Model dove sono presenti i vari metodi con le query per interrogare e riportare informazioni presenti nel database.

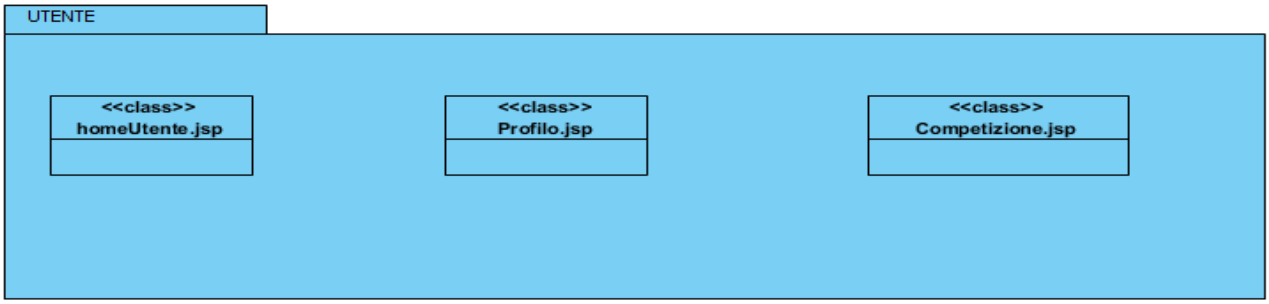
3. Package Components

3.1 Package Utente Non Registrato



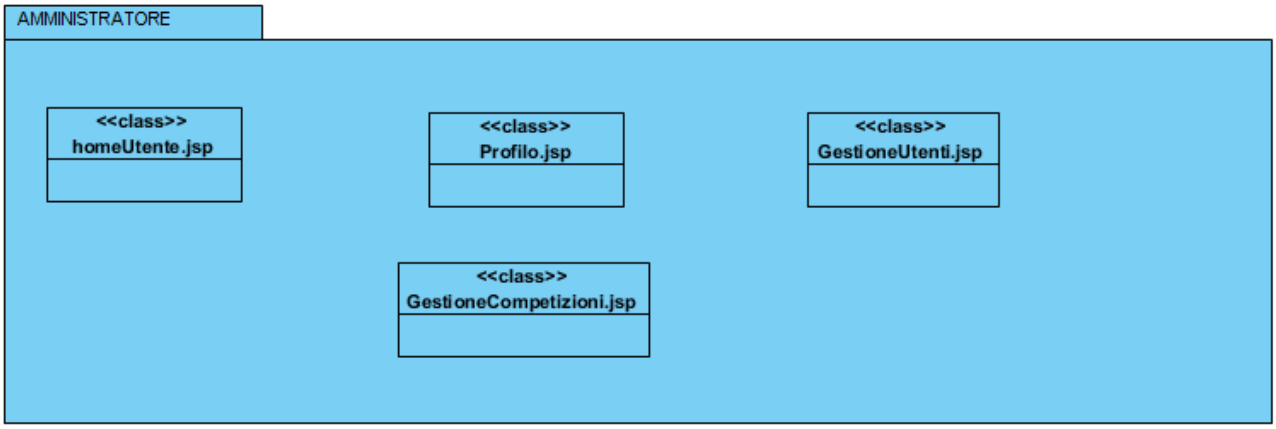
Classe:	Descrizione:
index.jsp	Si occupa di generare la pagina html che sarà la homepage di un utente non autenticato.

3.2 Package Utente



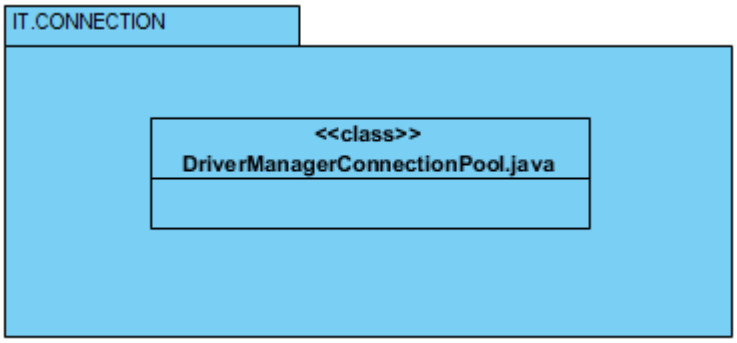
Classe:	Descrizione:
homeUtente.jsp	Si occupa di generare la pagina html che sarà la homepage di un utente autenticato.
Profilo.jsp	Si occupa di generare la pagina html che permette ad un utente di visualizzare e modificare le proprie informazioni.
Competizione.jsp	Si occupa di generare la pagina html che permette ad un utente di effettuare operazioni relative alle competizioni a cui partecipa.

3.3 Package Amministratore



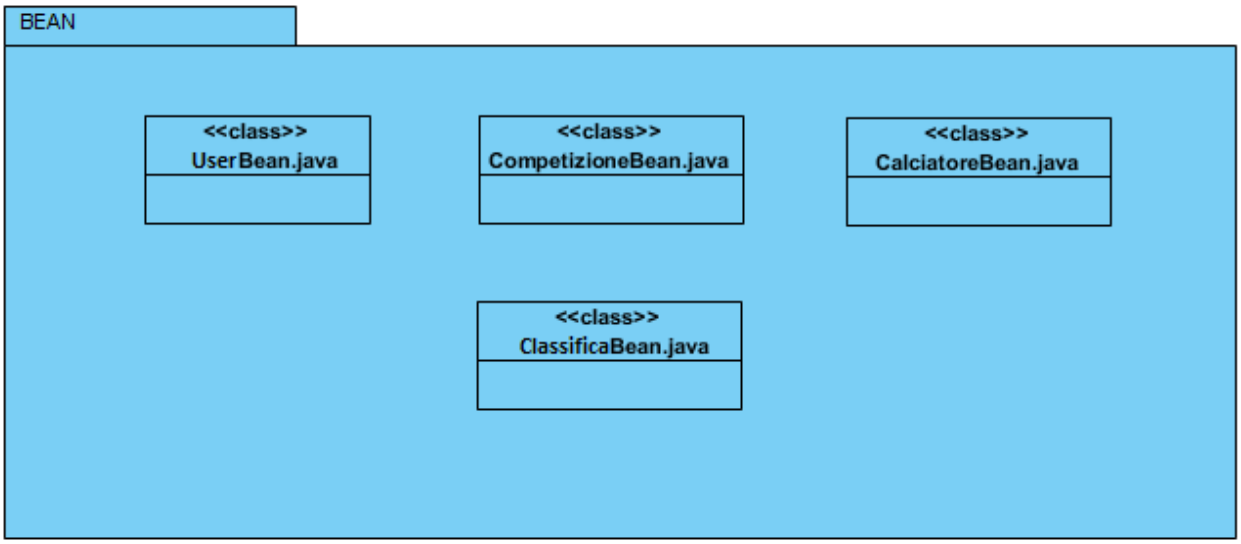
Classe:	Descrizione:
homeUtente.jsp	Si occupa di generare la pagina html che sarà la homepage dell'amministratore.
Profilo.jsp	Si occupa di generare la pagina html che permette ad un amministratore di visualizzare o modificare i propri dati.
GestioneUtenti.jsp	Si occupa di generare la pagina html che permette ad un amministratore di visualizzare o modificare i dati relativi agli utenti registrati.
GestioneCompetizioni.jsp	Si occupa di generare la pagina html per la gestione delle competizioni.

3.4 Package It.Connection



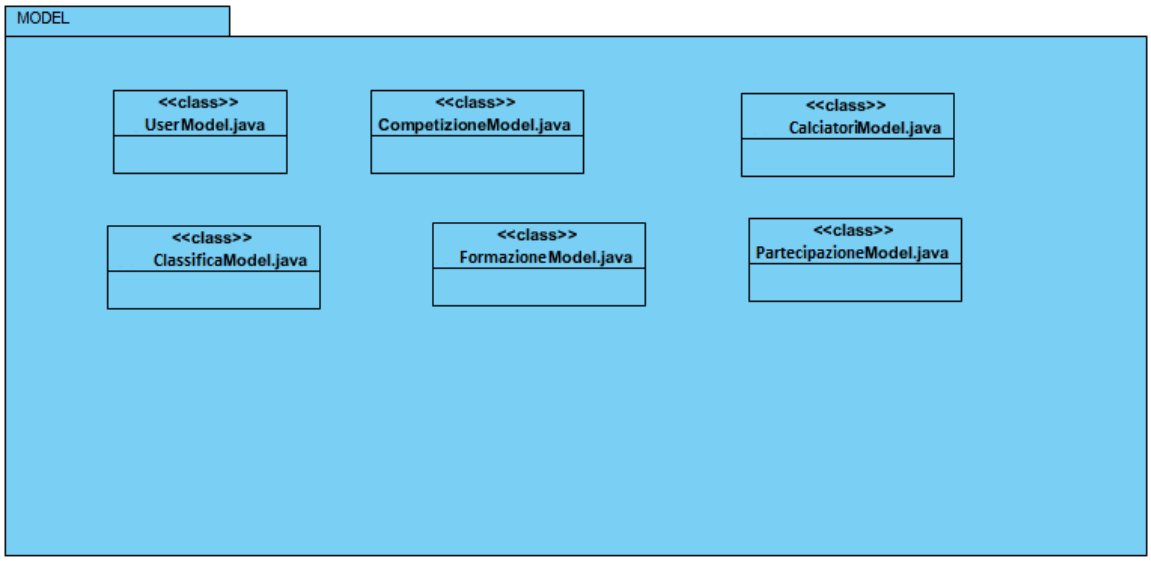
Classe:	Descrizione:
DriverManagerConnectionPool.java	Questa classe si occupa di gestire una pool di connessioni al database.

3.5 Package Bean



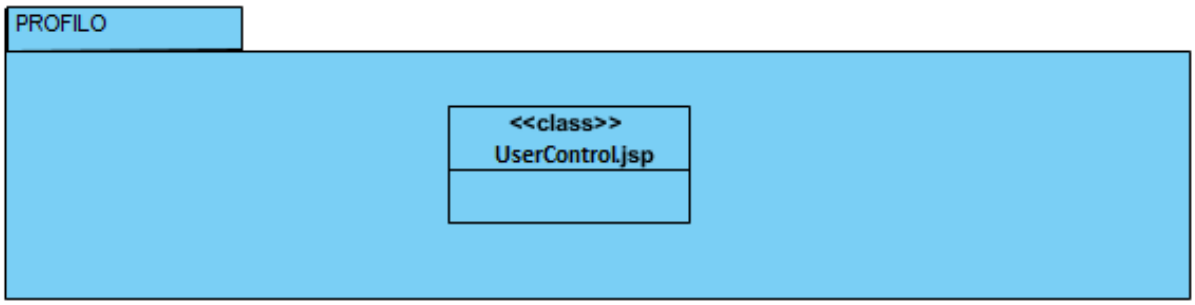
Classe:	Descrizione:
UserBean.java	Questa classe rappresenta un utente.
CompetizioneBean.java	Questa classe rappresenta una competizione.
ClassificaBean.java	Questa classe rappresenta una classifica.
CalciatoreBean.java	Questa classe rappresenta un calciatore.

3.6 Package Model



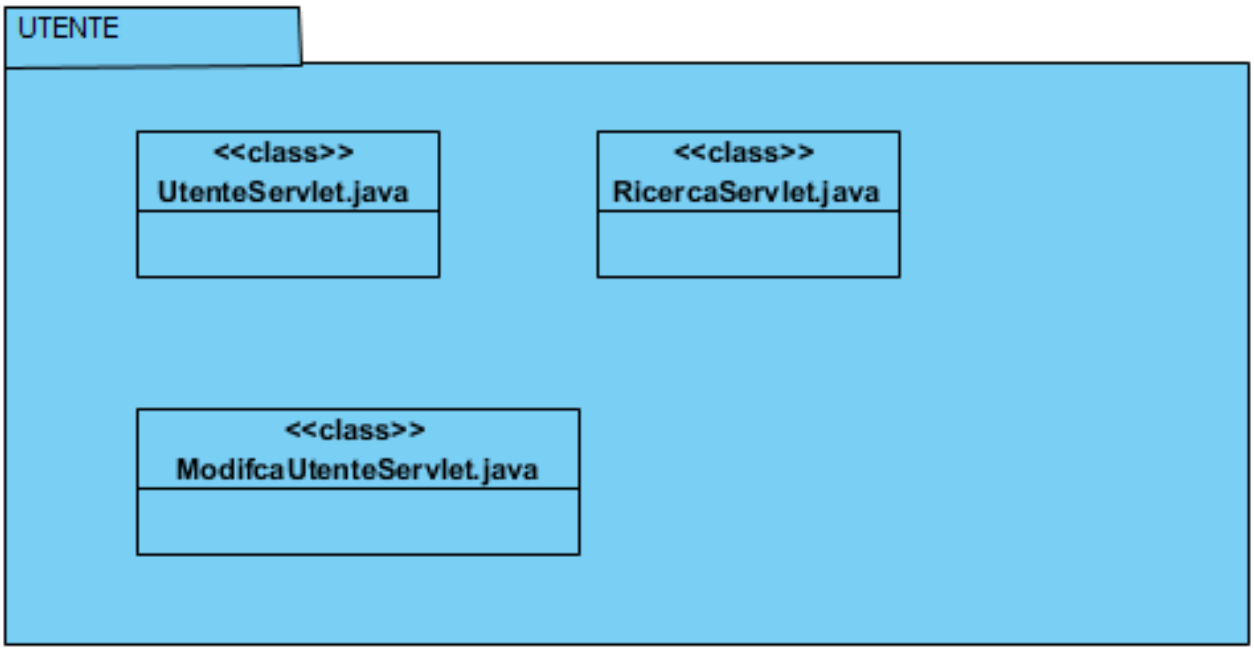
Classe:	Descrizione:
UserModel.java	Questa classe contiene metodi per la gestione degli utenti all'interno del database.
CompetizioneModel.java	Questa classe contiene i metodi per la gestione delle competizion all'interno del database.
CalciatoriModel.java	Questa classe contiene i metodi per la gestione dei calciatori all'interno del database.
ClassificaModel.java	Questa classe contiene i metodi per la gestione della classifica all'interno del database.
FormazioneModel.java	Questa classe contiene i metodi per la gestione della formazione all'interno del database.
PartecipazioneModel.java	Questa classe contiene i metodi per la gestione dellle partecipazioni relative alle competizioni all'interno del database.

3.6 Package Profilo



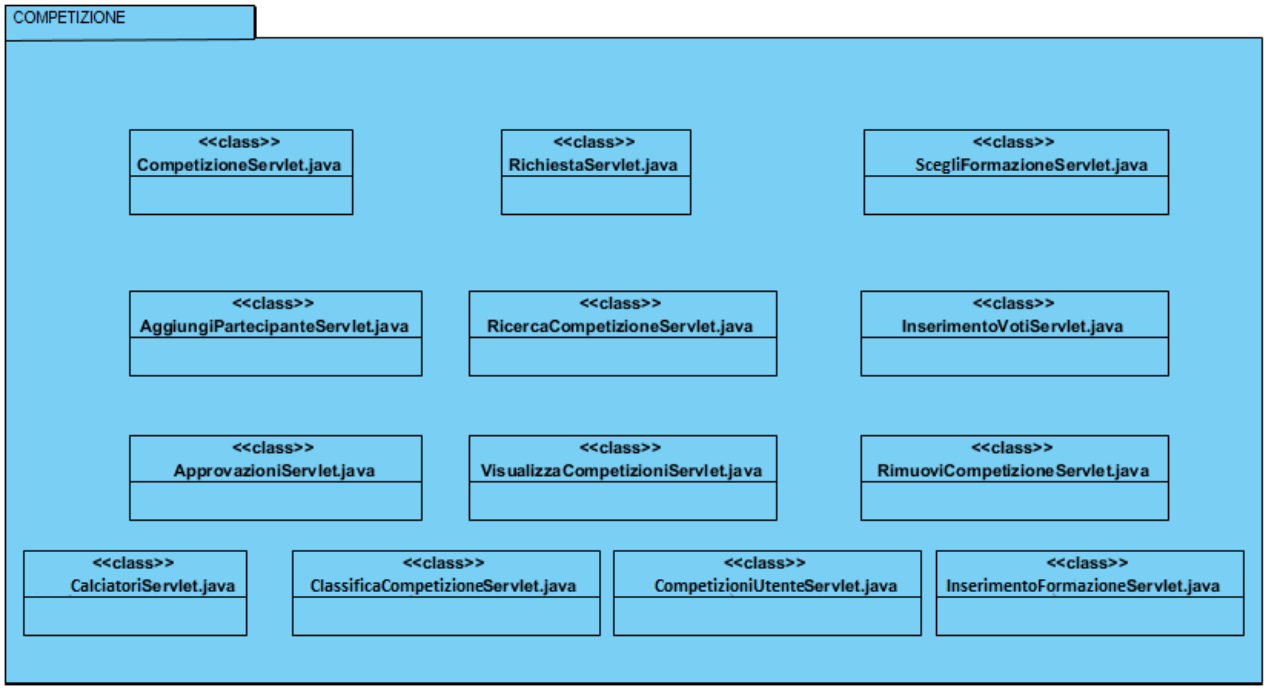
Classe:	Descrizione:
UserControl.java	Questa servlet si occupa di prendere i dati in input dalla form Modifica Profilo in profilo.jsp e passarli a UserModel.java per la modifica .

3.7 Package GestioneUtente



Classe:	Descrizione:
UtenteServlet.java	Questa servlet recupera una lista di tutti gli utenti del sistema tramite UserModel.java e la fornisce a GestioneUtenti.jsp per la visualizzazione.
RicercaServlet.java	Questa servlet invia a UtenteModel.java dati da ricercare all'interno del database.
ModificaUtenteServlet.java	Questa servlet invia dati ricevuti in input dalla form Modifica Utente in GestioneUtenti.jsp e li fornisce a UserModel.java per la modifica.

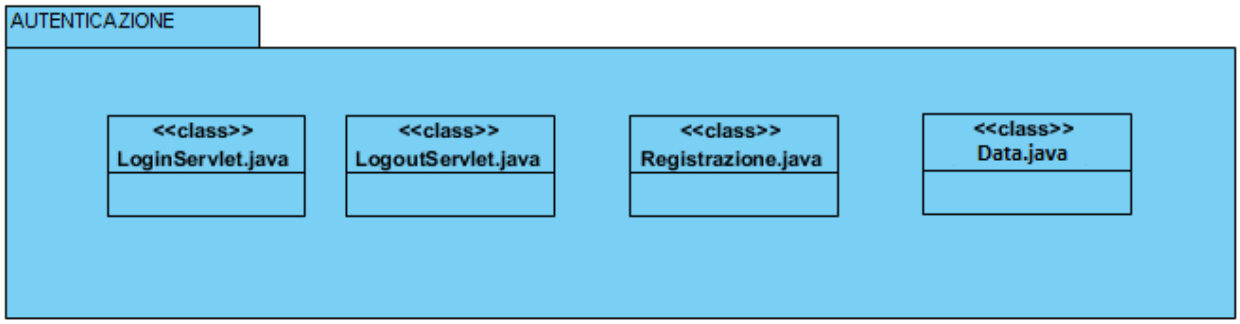
3.8 Package Competizione



Classe:	Descrizione:
CompetizioneServlet.java	Questa servlet recupera una lista di tutte le competizioni approvate tramite CompetizioneModel.java e li fornisce a GestioneCompetizioni.jsp per la visualizzazione.
RichiestaServlet.java	Questa servlet riceve in inputi i dati dalla forma Richiedi Competizione in competizioni.jsp e li fornisce a CompetizioneModel.java per l'elaborazione.
ScegliFormazioneServlet.java	Questa servlet recupera una lista di tutti i calciatori in base al ruolo selezionato tramite CalciatoreModel.java e li fornisce a competizioni.jsp per la visualizzazione.
AggiungiPartecipanteServlet.java	Questa servlet riceve in input i dati dalla form Aggiungi Partecipante in competizioni.jsp e li fornisce a CompetizioneModel.java per l'elaborazione.

RicercaCompetizioneServlet.java	Questa servlet invia dati a CompetizioneModel.java per la ricerca all'interno del database.
InserimentoVotiServlet.java	Questa servlet invia i voti ricevuti in input dalla form Inserimento Voti e li invia a CompetizioneModel.java per l'inserimento.
ApprovazioniServlet.java	Questa servlet indica tramite gli input della form Approvazioni in GestioneCompetizioni.jsp a CompetizioneModel.java quali competizioni sono da approvare.
VisualizzaCompetizioneServlet.java	Questa servlet recupera dati da CompetizioneModel.java per fornirli a Competizione.jsp per la visualizzazione
RimuoviCometizioneServlet.java	Questa servlet indica a CompetizioneModel.java quale competizione si deve eliminare.
InserimentoFormazioneServlet.java	Questa servlet invia la formazione ricevuta in input dalla form Inserimento Formazione in competizioni.jsp e li invia a CompetizioneModel.java per l'inserimento.
CalciatoriServlet.java	Questa servlet recupera una lista di tutti i calciatori che non hanno ancora una votazione tramite CalciatoreModel.java e li fornisce a competizioni.jsp per la visualizzazione.
ClassificaCompetizioneServlet.java	Questa servlet recupera la classifica relativa alla competizione selezionata tramite ClassificaModel.java e la fornisce a competizioni.jsp per la visualizzazione.
CompetizioniUtenteServlet.java	Questa servlet recupera una lista di tutte le competizioni relative all'utente autenticato tramite CompetizioneModel.java e la fornisce a competizioni.jsp per la visualizzazione.

3.9 Package Autenticazione

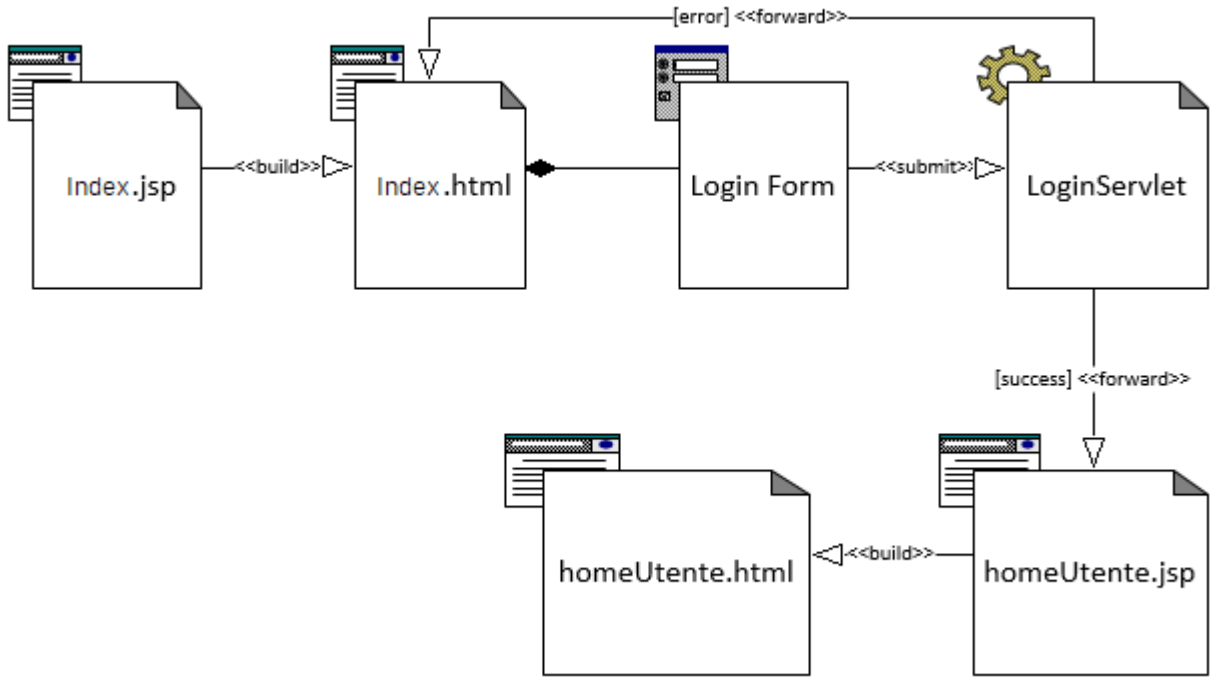


Classe:	Descrizione:
LoginServlet.java	Questa servlet si occupa di ricevere i dati dalla jsp di login elaborarli e decidere se consentire o no l'accesso all'area personale.
LogoutServlet.java	Questa servlet si occupa di invalidare la sessione per consentire il logout.
Data.java	Questa servlet si occupa di caricare informazioni relative all'utente autenticato.
RegistrazioneServlet.java	Questa servlet riceve i dati da registrazione.jsp e li elabora e utilizza i servizi di UtenteModel per effettuare la registrazione

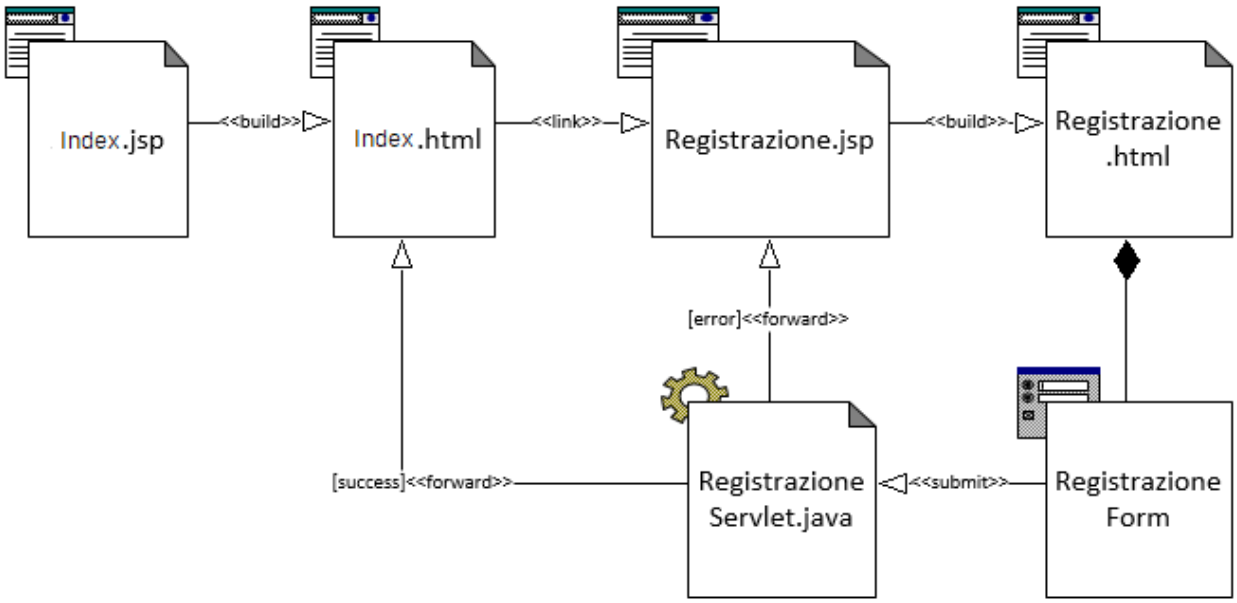
4. Diagrammi a Run Time

A completamento dei diagrammi a design time presenti nell'SDD si presentano qui i diagrammi a run time per rendere più chiara l'interazione tra le varie componenti.

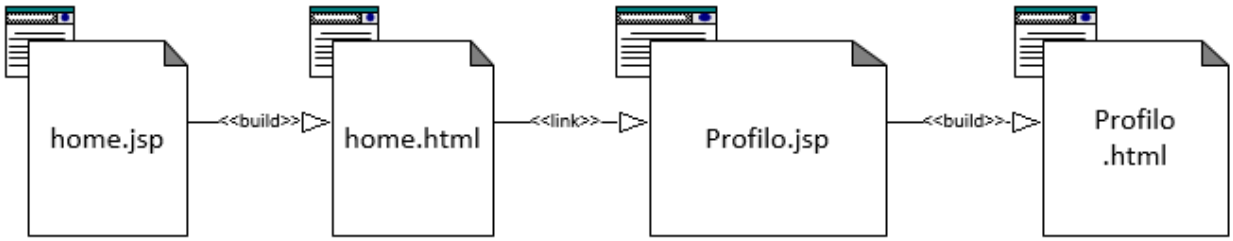
4.1. Login



4.2. Registrazione

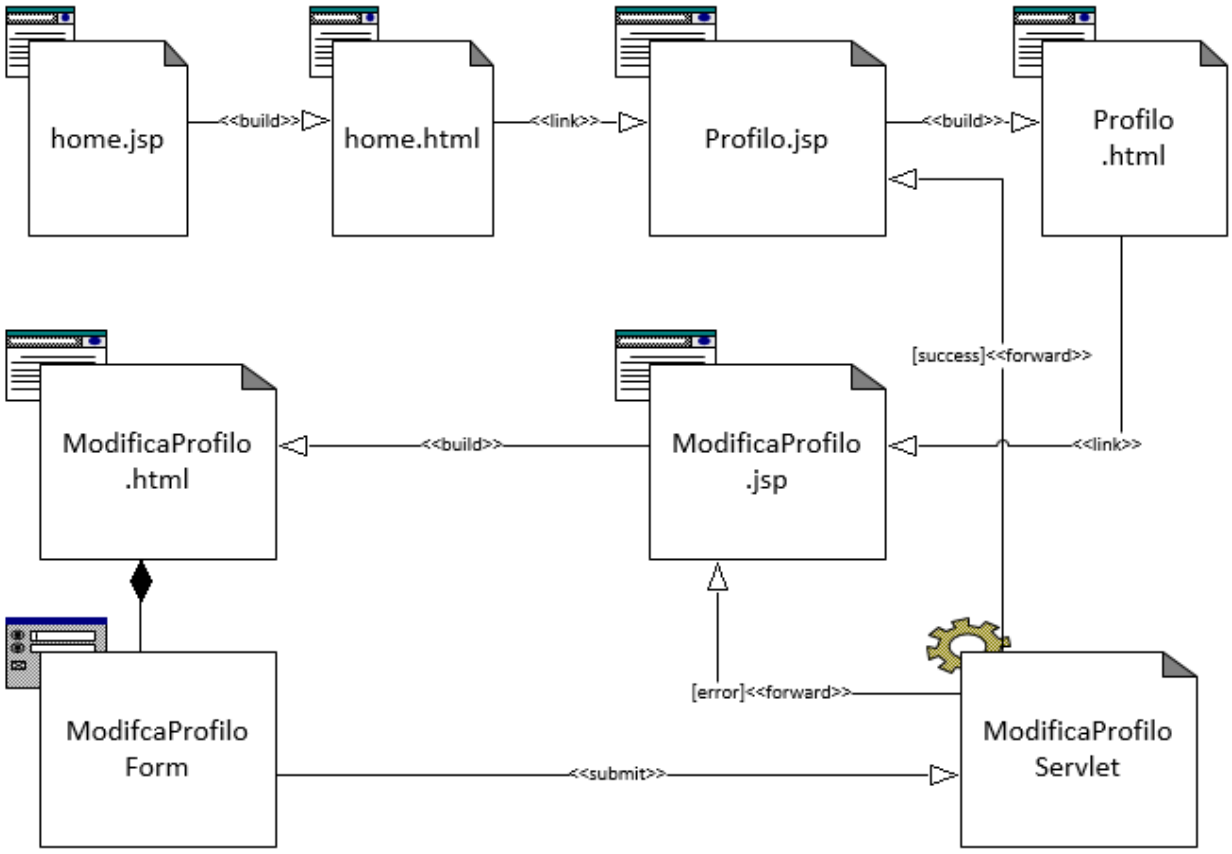


4.3. Visualizza Profilo



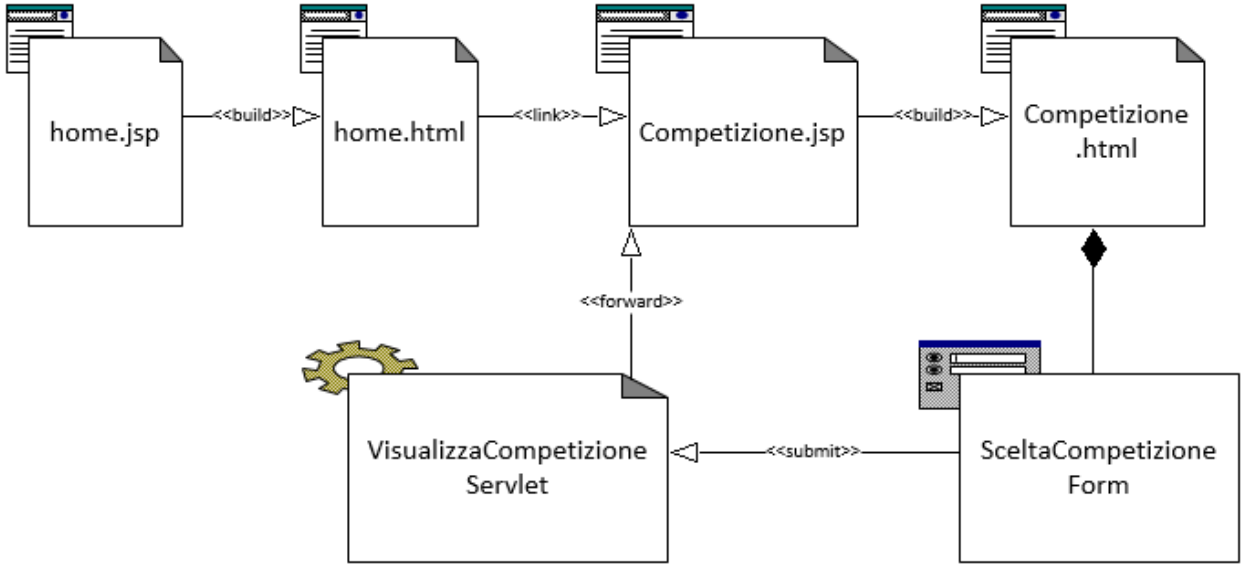
home.jsp rappresenta homeUtente che è uguale per Amministratore ed Utente ,a presenta due composizioni differenti che vengon generate in base al ruolo.

4.4. Modifica Profilo



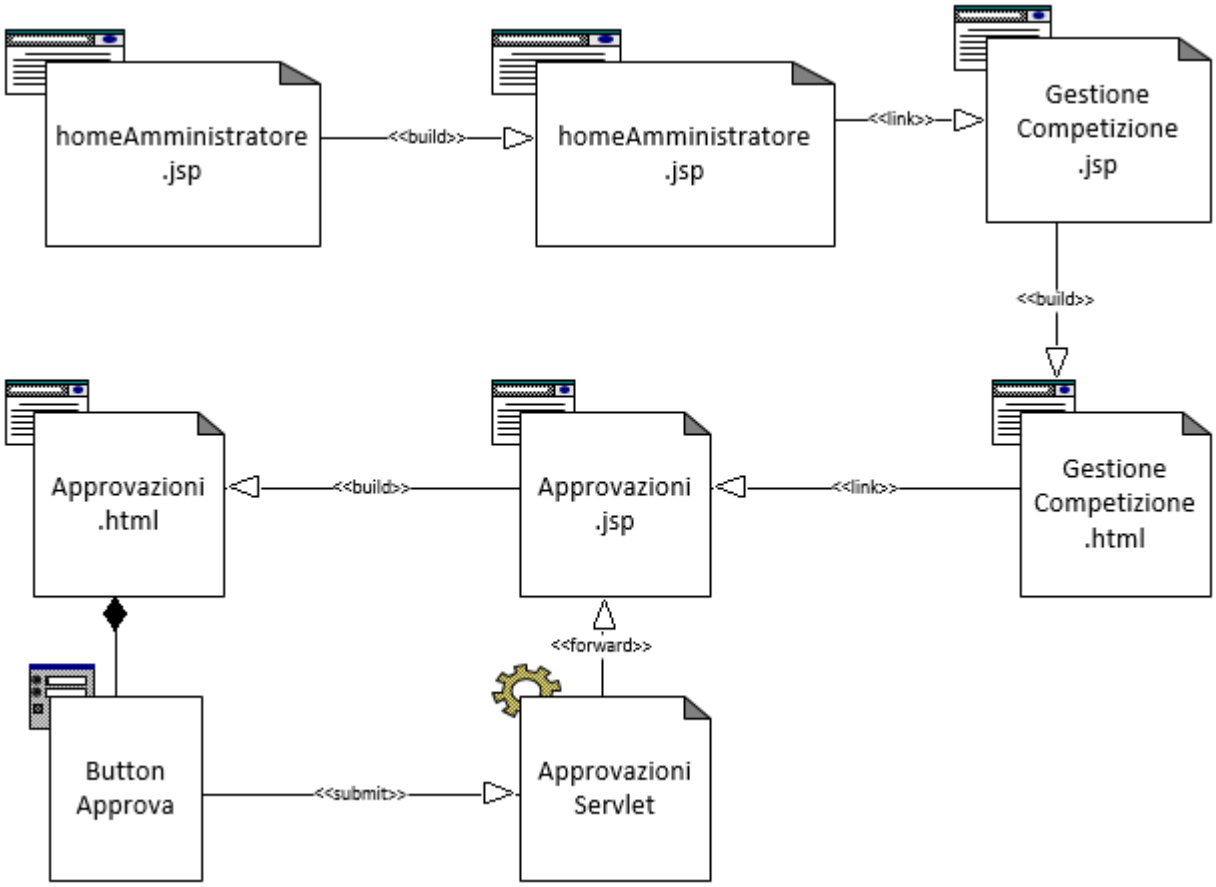
home.jsp rappresenta homeUtente che è uguale per Amministratore ed Utente ,a presenta due composizioni differenti che vengon generate in base al ruolo.

4.5 Visualizza Competizione

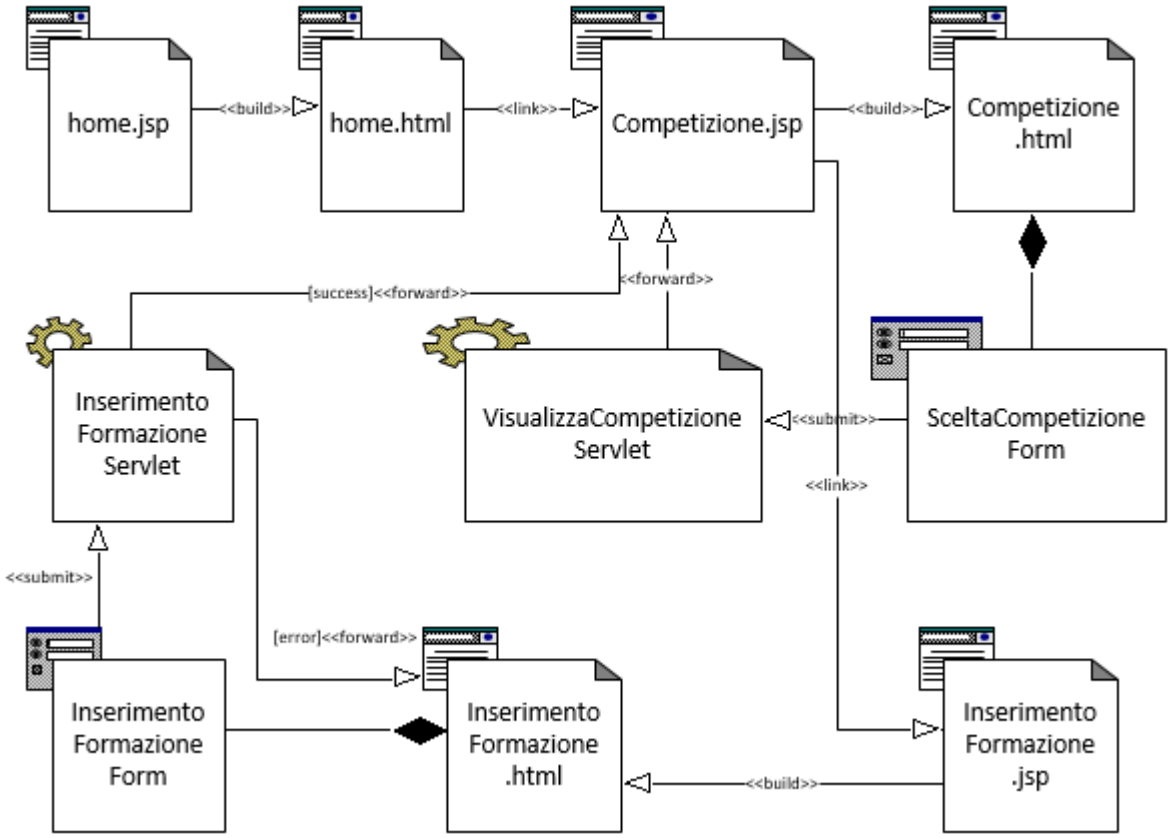


home.jsp rappresenta homeUtente che è uguale per Amministratore ed Utente ,a presenta due composizioni differenti che vengon generate in base al ruolo.

4.6. Approvazioni

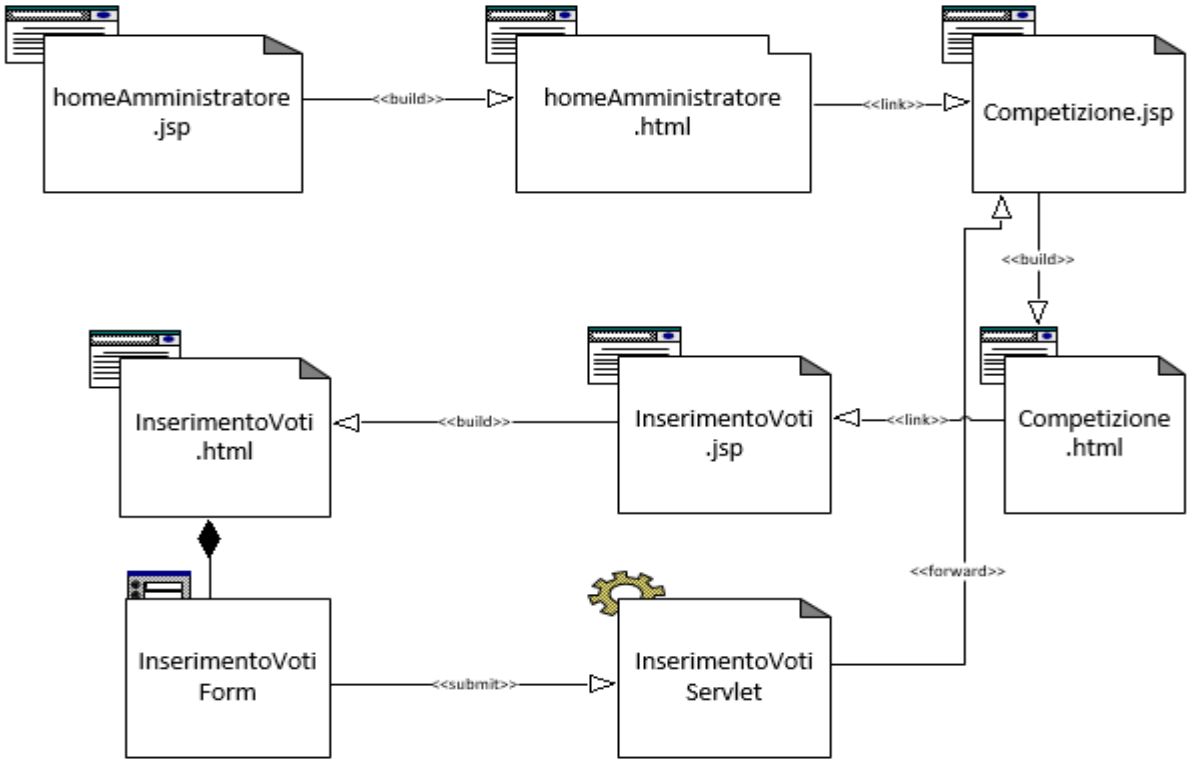


4.7. Inserimento Formazione

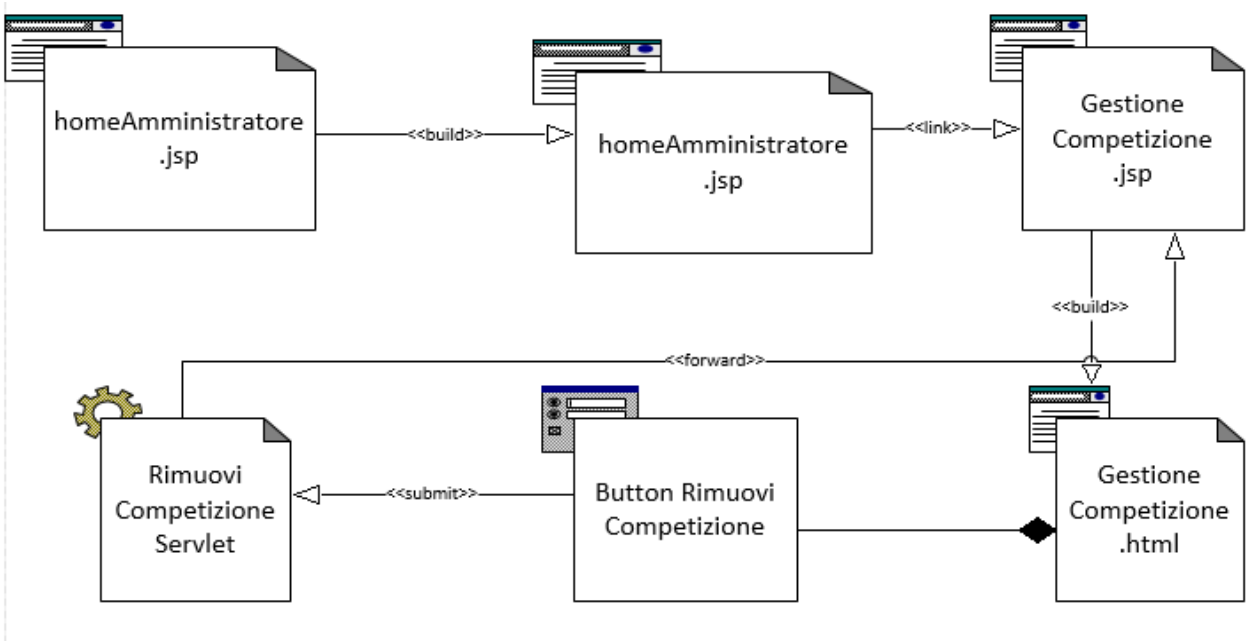


home.jsp rappresenta homeUtente che è uguale per Amministratore ed Utente ,a presenta due composizioni differenti che vengon generate in base al ruolo.

4.8. Inserimento Voti



4.9. Rimozione Competizione



homeAmministratore.jsp rappresenta homeUtente che è uguale per Amministratore ed Utente ,a presenta due composizioni differenti che vengon generate in base al ruolo.

5. Class Interfaces

5.1. UserModel

UtenteModel	
doSave	**Context:** UserModel: doSave(utente) **Pre:** utente!=null && utente.email !=null && utente.nickname !=null && utente.username != null && utente.password !=null && utenti->forAll(u |u.username != utente.username); **Post:** Utente ->include(utente)
doRetrieveRole	**Context:** UserModel: doRetrieveRole(username) **Pre:** username!=null && Utente-> exists(u|u.username==username); **Post:** utente->select(u.ruolo | u.username==username);
doUpdate	**Context:** UserModel: doUpdate(email, password, nickname, username) **Pre:** email != null && nickname != null && password != null && Utente->exists(u| u.username=username)); **Post:** Utente->exists(u| u.username=username && u.nickname==nickname && u.password==password && u.email==email));
doRetrieveAll	**Context:** UserModel: doRetrieveAll() **Post:** Utente→ select (u | u.ruolo==’u’);

5.2. CompetizioneModel

CompetizioneModel	
doSave	**Context:** CompetizioneModel: doSave(competizione) **Pre:** competizione!=null && competizione.nome !=null && competizione.numGiornate !=null && competizione.numPartecipanti != null && competizioni->forAll(c |c.nome != competizione.nome); **Post:** Competizione ->include(competizione)
doSearch	**Context:** CompetizioneModel: doSearch(nome) **Pre:** nome!=null && Competizione-> exists(c|c.nome==nome); **Post:** Competizione->select(c | c.nome==nome);
doDelete	**Context:** CompetizioneModel: doDelete(id) **Pre:** Competizione-> exists(c|c.id==id); **Post:** !Competizione-> exists(c|c.id==id);

Approva	**Context:** CompetizioneModel: approvaCompetizione(id)
	
	Pre: Competizione-> exists(c|c.idComp==id &&
	c.approvata==false);
	
	Post: Competizione-> exists(c|c.id==id &&
	c.approvata==true);

doRetrieveByUtente	**Context:** CompetizioneModel: doRetrieveByUtente(utente) **Pre:** utente!=null; ; **Post:** Competizioni-> select(c|c.utente==utente);

doRetrieveNotApproved	**Context:** CompetizioneModel: doRetrieveNotApproved() **Post:** Competizioni->select(c| c.approvata==0);
doRetrieveAll	**Context:** CompetizioneModel: listaCompetizioni() **Post:** Competizione→ select (c | c.approvata==1);

5.3. Calciatore Model

CalciatoreModel	
doSaveVoto	**Context:** CalciatoreModel: doSaveVoto(id,giornata,voto,gol,assist,ammonizione,espulsione) **Pre:** id!=null && giornata !=null && voto !=null && gol != null && assist != null && ammonizione != null && espulsione != null ; **Post:** Voto ->include(voto)
doRetrieveByRuolo	**Context:** CalciatoreModel: doRetrieveByRuolo(ruolo) **Pre:** ruolo!=null**;** **Post:** Calciatori->select(c | c.ruolo==ruolo);
doRetrieveAll	**Context:** CalciatoreModel: doRetrieveAll() **Post:** Calciatore-> select(c|c !exists(v|v.idCalc==c.id));

5.4. Classifica Model

ClassificaModel	
doRetrieveByComp	**Context:** ClassificaModel: doRetrieveByComp(nome) **Pre:** nome!=null**;** **Post:** Classifica->select(formazione f | f.utente exists(competizione c|c.nome==nome));
doRetrieveAll	**Context:** ClassificaModel: doRetrieveAll() **Post:** Classifica-> select(formazione f | f exists(f|f.utente==utente));

5.5. Formazione Model

FormazioneModel	
doSaveFormazione	**Context:** FormazioneModel: doSaveFormazione(utente, giornata) **Pre:** utente!=null && giornata !=null && formazione->forAll(f |f.utente != utente && f.giornata!=giornata); **Post:** Formazione ->include(formazione)
doRetrieveGiornata	**Context:** FormazioneModel: doRetrieveGiornata() **Post:** Giornata->select(g|max(g.id));

checkFormazione	**Context:** FormazioneModel: checkFormazione(utente, giornata) **Pre:** utente != null && giornata !=null;

doSaveComposizione	**Context:** FormazioneModel: doSaveComposizione(idForm, Calciatori[])
	
	Pre: formazione-> exists(f|f.idForm==idForm) &&
	Calciatori[]!=null &&
	idForm != null;
	
	Post: Composizione-> include(c|c.formazione==idForm &&
	forAllCalciatori[i](Calciatore==Calciatori[i]));

doRetrieveId	**Context:** FormazioneModel: doRetrieveId(utente, giornata)
	
	Pre: utente!=null &&
	giornata !=null;
	;
	
	Post: Formazione-> select(f|f.utente==utente &&
	f.giornata==giornata);

doRetrieveFormazione	**Context:** FormazioneModel:doRetrieveFormazione(utente,giornata) **Pre:** utente != null && giornata != null && f.idForm-> exists(f|f.utente==utente && f.giornata==giornata); **Post:** Composzione-> select(c|c.idForm==f.idForm);

5.6. Partecipazione Model

PartecipazioneModel	
doSave	**Context:** PartecipazioneModel: doSave(utente) **Pre:** utente!=null; **Post:** Partecipazione ->include(partecipazione);
aggiungi	**Context:** PartecipazioneModel: aggiungi(utente,competizione) **Pre:** utente!=null && competizione!=null; **Post:** Partecipazione ->include(partecipazione);

5.7. Servlet Autenticazione

LoginServlet	**Pre**: username!=null && password!=null && utente->exists(ut| ut.username==username);
LogoutServlet	
RegistrazioneServlet	**Pre**: username!=null && nickname !=null && email !=null && indirizzo!=null && password!= null && !utente->exists(ut| ut.username==username); **Post**: utente->exists(ut| ut.username==username);
Data	**Post:** competizione!=null && giornata != null && formazione !=null;

5.8. Servlet Profilo

UserControl.java	**Pre**: nickname != null &&
	email != null &&
	password != null &&
	username!=null;
	
	Post: Utente->exists(u| u.username=username &&
	u.nickname==nickname &&
	u.password==password &&
	u.email==email));

5.5. Servlet Gestione Utente

UtenteServlet	**Pre:** utente.ruolo == 'a' **Post:** Utente→ select (u | u.ruolo=='u');
RicercaUtenteServlet	**Pre:** username!=null && Utente-> exists(u | u.username==username); **Post:** utente->select(u | u.username==username);
ModificaUtenteServlet	**Pre:** (nickname != null || email != null || password != null) && Utente->exists(u| u.username=username)); **Post:** Utente->exists(u| u.username=username && u.nickname==nickname && u.password==password && u.email==email));

5.6. Servlet Competizione

CompetizioneServlet	**Pre:** utente.ruolo == ‘a’ **Post:** Competizione→ select (c | c.approvata==true);
RicercaCompetizioneServlet	**Pre:** utente.ruolo == ‘a’ && nome!=null && Competizione-> exists(c|c.nome==nome); **Post:** Competizione->select(c | c.nome==nome);
RimuoviCompetizioneServlet	**Pre:** utente.ruolo == ‘a’ && Competizione-> exists(c|c.id==id); **Post:** !Competizione-> exists(c|c.id==id);

ApprovaServlet	**Pre:** utente.ruolo == ‘a’ &&
	Competizione-> exists(c|c.nome==nome &&
	c.approvata==false);
	
	Post: Competizione-> exists(c|c.nome==nome &&
	c.approvata==true);

InserimentoVotoServlet	**Pre:** utente.ruolo == ‘a’ && voto.idCalciatore!=null && voto.idGiornata!=null && voto.votazione!=null ; **Post:** Voto-> exists(v|v.idCalciatore==voto.idCalciatore && v.idGiornata==voto.idGiornata);
ScegliFormazioneServlet	**Pre:** utente.ruolo == ‘u’ && ruolo!=null ; **Post:** Calciatore->select(c | c.ruolo==ruolo);

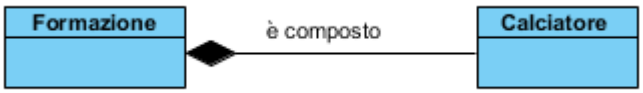
InserimentoFormazioneServlet	**Pre:** utente.ruolo == ‘u’ &&
	formazione.idGiornata!=null &&
	formazione.username!=null &&
	formazioni->forAll(f |f.idGiornata != formazione.idGiornata &&
	f.username != formazione.username);
	
	Post: Formazione-> exists(f|f.idGiornata==formazione.idGiornata &&
	f.username==formazione.username);

AggiungiPartecipanteServlet	**Pre:** username!=null &&
	idCompetizione!=null &&
	partecipazioni->forAll(p | p.username != username &&
	p.idCompetizione != idCompetizione);
	
	Post: Partecipazioni-> exists(p | p.idCompetizione== id.Competizione
	&&
	p.username==username);

RichiestaServlet	**Pre:** utente.ruolo == 'u' && competizione!=null && competizione.nome !=null && competizione.numGiornate !=null && competizione.numPartecipanti != null && competizione.approvata == true && competizioni->forAll(c |c.nome != competizione.nome); **Post:** Competizione ->include(competizione)
VisualizzaCompetizioneServlet	**Pre:** Competizione-> exists(c|c.nome==nome); **Post:** Competizione->select(c | c.nome==nome);
CalciatoreServlet	**Post:** Calciatore→ select (all);
ClassificaCompetizioneServlet	**Pre:** nome !=null; **Post: Post:** Classifica->select(sum(formazione.punteggio) | formazione.utente==formazione.utente);
CompetizioniUtente	**Pre:** utente!=null; **Post:** Competizioni->select(partecipazione.competizione | partecipazione.utente==utente);

6. Mapping Model

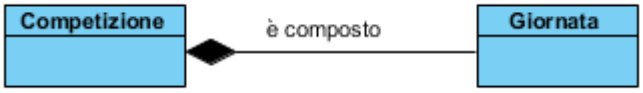
6.1. Formazione



Per gestire questo mapping abbiam inserito una nuova tabella Composizione avente come attributi le chiavi primarie delle entità Formazione e Calciatore, cioè:

- Int Formazione.idForm
- Int Calciatore.idCalc

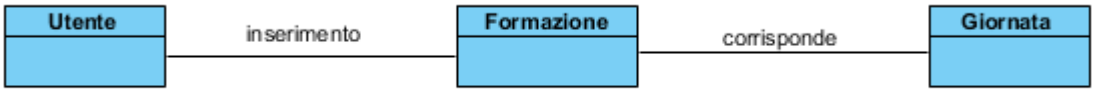
6.2. Giornate



Per gestire questo mapping abbiamo inserito un altro attributo all'interno di Giornata:

- Int Competizione.idCompetizione

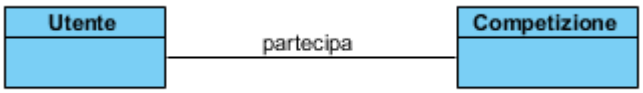
6.3. Inserimento Formazione



Per gestire questo mapping abbiam inserito due attributi alla tabella di Formazione:

- String UtenteRegistrato.Username
- Int Giornata.idGiornata

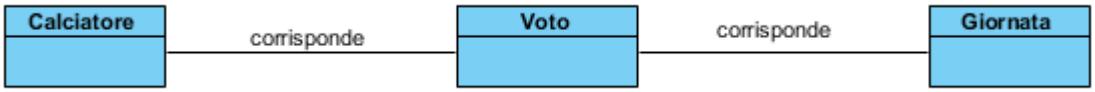
6.4. Partecipazione



Per gestire questo mapping abbiamo inserito una nuova tabella Partecipazione per tenere traccia dei componenti di ciascuna competizione e avente come attributi:

- String UtenteRegistrato.Username
- Int Competizione.idComp

6.5. Voto



Per gestire questo mapping abbiamo inserito all'interno della tabella Voto due attributi:

- Int Calciatore.idCalc
- Int Giornata.idGiornata

7. Glossario

RAD: Documento di Analisi dei Requisiti.

DBMS: Sistema di gestione di basi di dati.

SDD: Documento di System Design

ODD: Documento di Object Design

Database: Insieme organizzato di dati persistenti.

Query: Termine utilizzato per indicare l'interrogazione da parte di un utente di un database.

Utente Registrato: Il termine identifica utente che ha effettuato la registrazione sul sistema

Web-Based: Il termine identifica un sistema basato sul web, quindi accessibile simultaneamente da più postazioni