



Universitatea TRANSILVANIA din Brașov
Facultatea de Matematică și Informatică
Programul de studiu Informatică aplicată



Lucrare de licență

Autor: **Trifanov Paula**
Coordonator științific: **Prof. univ. dr. Bocu Dorin**

Brașov
Iulie, 2016

Aplicație suport pentru
constituirea de echipe echilibrate
și customizabile

**FIȘA LUCRĂRII DE ABSOLVIRE/ LUCRĂRII DE LICENȚĂ/ PROIECTULUI DE
DIPLOMĂ/ DISERTAȚIE**

Universitatea <i>Transilvania</i> din Brașov	Lucrare de absolvire
Facultatea Matematică și Informatică Brașov	
Departamentul Matematică și Informatică	Viza facultății
Programul de studii: Informatică aplicată	Anul universitar 2015-2016
Candidat: Trifanov Paula	Promoția 2016
Cadrul didactic: îndrumător Prof. univ. dr. Bocu Dorin	

LUCRARE DE ABSOLVIRE/ LUCRARE DE LICENȚĂ/ PROIECT DE DIPLOMĂ/ DISERTAȚIE
Titlul lucrării: Aplicație suport pentru constituirea de echipe echilibrate și personalizabile
<p>Problemele principale tratate:</p> <p>Organizarea persoanelor din cadrul unui grup</p> <p>Planificarea activităților în cadrul unui grup</p> <p>Consemnarea schimbărilor apărute în cadrul organizației</p> <p>Consultarea istoricului evenimentelor</p> <p>Consemnarea evoluției membrilor grupului</p>
<p>Locul și durata practicii:</p> <p>Siemens, 20 iunie 2015-iulie 2016</p>
<p>Bibliografie:</p> <ul style="list-style-type: none"> • http://en.wikipedia.org/wiki/JavaServer_Faces • http://www.roseindia.net/jsf/

- http://www.w3schools.com/bootstrap/bootstrap_get_started.asp
- *Inițiere în programarea orientată pe obiecte din perspectivă C++ și Java.* - Dorin Bocu
- 2., *Thinking in Java*” -Bruce Eckel. Retrieved from „Thinking in Java” -Bruce Eckel

Aspecte particulare:

(desene, aplicații practice, metode specifice etc.)

Primit tema la data de:

Data predării lucrării:

Director departament,
Conf.univ.dr. MARIUS PĂUN,
semnătura

Cadru didactic îndrumător,
(nume,prenume,semnătură)

Candidat,
(nume,prenume, semnătura

**LUCRARE DE ABSOLVIRE/ LUCRARE DE LICENȚĂ/ PROIECT DE DIPLOMĂ/
DISERTAȚIE – VIZE**

Data vizei	Capitole/ problemele analizate	Semnătura cad didactic îndrumător

APRECIEREA ȘI AVIZUL CADRULUI DIDACTIC ÎNDRUMĂTOR		
Data:	ADMIS pentru susținere/ RESPINS	CADRU DIDACTIC ÎNDRUMĂTOR (nume,prenume,semnătură)

AVIZUL DIRECTORULUI DE DEPARTAMENT		
Data:	ADMIS pentru susținere/ RESPINS	Director departament Conf.univ. dr. Marius Păun, semnătură

--	--	--

<p align="center">SUSȚINEREA LUCRĂRII DE ABSOLVIRE/ LUCRĂRII DE LICENȚĂ/ PROIECTULUI DE DIPLOMĂ/ DISERTAȚIEI</p>	
Sesiunea	
Rezultatul Susținerii	PROMOVAT cu media:
	RESPINS cu refacerea lucrării
	RESPINS fără refacerea lucrării
PREȘEDINTE COMISIE (nume, prenume, semnătura)	

Cuprins

1	Introducere	1
2	Descriere problemei	2
3	Tehnologii folosite	3
3.1	Java.....	3
3.2	Maven.....	5
3.2.1	Descriere	5
3.2.2	POM.....	5
3.3	Java Server Faces	6
3.3.1	Prezentare generală	6
3.3.2	Caracteristici	7
3.3.3	Arhitectura JSF	8
3.3.4	Utilitatea framework-ului.....	10
3.3.5	Utilizarea în cadrul unei aplicații	11
3.4	Primefaces	12
3.4.1	Introducere	12
3.4.2	Setup.....	13
3.4.3	Dependințe	13
3.4.4	Componente Primefaces	14
3.5	Bootstrap	17
3.5.1	Prezentare generală	17
3.5.2	Structură și funcționalitate	17
3.5.3	Descărcare și instalare.....	18
3.6	PostgreSQL	18
3.6.1	Caracteristici	18

3.6.2	pgAdmin.....	19
3.7	JPA(Java Persistence API)	20
3.7.1	Componente JPA.....	20
4	Descrierea soluției.....	26
4.1	Stabilirea obiectivelor.....	26
4.2	Cerințele utilizatorilor față de aplicație	26
4.3	Diagrama claselor.....	29
4.4	Considerații relativ la implementare	39
4.4.1	Suprascrierea stilului CSS implicit din Bootstrap	40
4.4.2	Persistența datelor	41
4.4.3	Algoritmul de generare de echipe ‘echilibrate’	44
4.4.4	Actualizarea modificărilor	45
4.4.5	Actualizarea rating-urilor	46
4.5	Ghid de utilizare	47
4.6	Index de figuri	63
5	Concluzii	66
6	Bibliografie	Error! Bookmark not defined.

1 Introducere

Problema organizării unui grup de persoane, indiferent de domeniul de activitate sau de scopul propus, poate fi destul de complexă. Poate fi desemnată o persoană căreia să îi revină această sarcină, însă îndeplinirea obiectivului poate fi destul de complicată, având în vedere faptul că persoana responsabilă ar trebui să aibă acces la o multitudine de date despre fiecare persoană agregată în grup.

Pentru a se putea organiza într-un mod eficient, un grup are nevoie de informații ce privesc membrii grupului în mod general, dar și informații particulare pentru fiecare membru în mod individual. De asemenea, membrii grupului ar trebui să își poată exprima opinia în legătură cu modul în care se face organizarea, acest lucru presupunând găsirea unei modalități eficiente de comunicare între leader-ul grupului și restul membrilor.

O primă problemă care apare o reprezintă împărțirea grupului în mai multe subgrupuri, pentru a desfășura diferite activități. Trebuie stabilite niște reguli care să stea la baza formării subgrupurilor, reguli acceptate și validate de către toți membrii grupului. De asemenea, evidența acțiunilor grupului este greu de păstrat și întreținut, dacă nu există o modalitate flexibilă și dinamică, precum un tool sau un soft corespunzător.

Proiectul se dorește a fi unul care eficientizează modul în care se face organizarea în cadrul unui grup de persoane, utilizatorii săi având posibilitatea de a consulta evoluția grupului.

Lucrarea cuprinde 5 capitole, fiecare având un rol semnificativ în înțelegerea logicii și a modurilor de folosire a tehnologiilor pentru realizarea proiectului.

În cel de-al doilea capitol este realizată o descriere a problemei. Acesta va cuprinde o prezentare a principalelor probleme întâmpinate în organizarea unor activități cu caracter sportiv la nivel de grup structurat pe diferite criterii, dar și o descriere a potențialilor utilizatori.

Al treilea capitol este destinat prezentării tehnologiilor folosite pentru realizarea aplicației, urmând ca în următorul capitol să se facă o prezentare a modului în care au fost folosite tehnologiile pentru realizarea produsului final, precum și o prezentare a principalelor moduri în care un utilizator poate interacționa cu aplicația.

2 Descriere problemei

Problema concretă o reprezintă realizarea unor activități cu caracter sportiv la nivel de grup structurat pe diferite criterii. Se dorește realizarea a doua echipe care să fie “echilibrate”, echilibrul fiind obținut pe baza mai multor criterii. De asemenea, se dorește stabilirea programului pentru activitățile în desfășurare sau activitățile următoare.

Una dintre principalele dificultăți întâlnite în cadrul organizării unui grup de persoane o reprezintă preluarea și folosirea datelor fiecărui individ din grup. Folosirea datelor personale ale fiecărui individ ar trebui făcută cu informarea și confirmarea acestuia, numai de către persoane autorizate.

O altă problemă o reprezintă delegarea responsabilităților în cadrul grupului. Pentru acceptarea unor noi membrii în cadrul grupului, aceștia trebuie să îndeplinească anumite condiții, impuse de leader-ul grupului și confirmate de ceilalți membrii. În cazul unei aplicații web, acest lucru presupune existența mai multor nivele de acces: utilizator de tip administrator sau utilizator simplu. Un utilizator simplu nu are posibilitatea de a modifica structura curentă a grupului, nici de a introduce noi membrii în cadrul acestuia, sau de a stabili un eveniment următor. Acest lucru trebuie să fie făcut de către un membru care deține aceste drepturi.

De asemenea, se dorește păstrarea unei evidențe a acțiunilor desfășurate în cadrul grupului, precum și existența unui feedback pentru membrii grupului care au participat la un eveniment. După fiecare acțiune, se dorește consemnarea detaliilor acesteia, pentru a se ține cont ulterior de consemnările făcute pentru o mai bună organizare a următoarelor evenimente.

Un alt punct de interes îl reprezintă consultarea schimbărilor apărute de-a lungul timpului, pentru deciderea următoarelor acțiuni. Scopul înregistrării detaliilor îl reprezintă formarea unei ierarhii în cadrul indivizilor. Cu cât un membru participă la mai multe acțiuni și primește feedback pozitiv, cu atât poziția sa în cadrul ierarhiei crește.

Potențialii utilizatori ar putea fi membrii unui grup, în cazul acesta al unei companii, interesați de activități sportive, precum meciurile de fotbal. Aceștia vor forma un grup ce va fi organizat în două subgrupuri, echipe, ce vor participa la activitățile sportive planificate. Grupul se dorește a fi unul dinamic, putând fi modificată structura acestuia în orice moment, prin introducerea de noi membrii. Subgrupurile sunt de asemenea dinamice, configurația acestora fiind influențată de mai mulți factori, precum poziția fiecărui individ în cadrul ierarhiei.

3 Tehnologii folosite

3.1 Java

Java este un limbaj de programare dezvoltat de Sun Microsystems, inițiat de James Gosling și lansat în 1995. Acesta rulează pe mai multe platforme, precum Windows, Mac OS și diferite versiuni de Linux. Cea mai recentă versiune de Java Standard Edition este Java SE 8. Odată cu avansarea rapidă a crescut și popularitatea limbajului, fiind realizate mai mult configurații, potrivite pentru diferite platforme: ex: J2EE pentru aplicațiile Enterprise sau J2ME pentru aplicațiile de mobil.

Limbajul împrumută o mare parte din sintaxă de la C și C++, dar are un model al obiectelor mai simplu și prezintă mai puține facilități de nivel jos. Un program Java compilat, corect scris, poate fi rulat fără modificări pe orice platformă care e instalată o mașină virtuală Java (Java Virtual Machine, prescurtat JVM). Acest nivel de portabilitate (inexistent pentru limbaje mai vechi cum ar fi C) este posibil deoarece sursele Java sunt compilate într-un format standard numit cod de octeți (byte-code) care este intermediar între codul mașină (dependent de tipul calculatorului) și codul sursă.

Mașina virtuală Java este mediul în care se execută programele Java. În prezent, există mai mulți furnizori de JVM, printre care Oracle, IBM, Bea, FSF. În 2006, Sun a anunțat că face disponibilă varianta sa de JVM ca open-source.

Există 4 platforme Java furnizate de Oracle:

- Java Card - pentru smartcard-uri (carduri cu cip)
- Java Platform, Micro Edition (Java ME) — pentru hardware cu resurse limitate, gen PDA sau telefoane mobile
- Java Platform, Standard Edition (Java SE) — pentru sisteme gen workstation
- Java Platform, Enterprise Edition (Java EE) — pentru sisteme de calcul mari, eventual distribuite

Deoarece folosește obiecte, Java este un limbaj de programare obiect orientat. Programarea Orientată pe Obiecte este o paradigmă de programare care utilizează obiecte și interacțiuni între acestea pentru a modela arhitectura unui program. Obiectele din POO modelează obiecte din lumea reală. Obiectele din POO sunt instanțe ale unui tip de date numit clasă.

În lumea reală, există niște tipare, care grupează niște atribute ale obiectelor cu acțiunile lor, pentru a forma un tot ce definește obiectul respectiv. Pe acest concept, numit încapsulare, se sprijină programarea orientată obiect.

Se consideră drept principale caracteristici ale obiectelor încapsularea, moștenirea și polimorfismul.

Încapsularea este proprietatea obiectelor de a-și ascunde o parte din date și metode. Din exteriorul obiectului sunt accesibile numai datele și metodele publice. Putem deci să ne imaginăm obiectul ca fiind format din două straturi.

Obiectul se comporă ca și când ar avea două "învelișuri": unul "transparent", care permite accesul la datele și metodele publice ale obiectului, și un al doilea înveliș "opac", care cuprinde datele și metodele invizibile (inaccesibile) din exterior. Starea obiectului depinde atât de datele publice, cât și de cele încapsulate. Metodele publice ale obiectului au acces la datele și metodele încapsulate (ascunse) ale acestuia. În consecință, starea obiectului poate fi modificată atât prin modificarea directă, din exterior, a valorilor variabilelor publice, fie prin utilizarea unor metode publice care modifică valorile variabilelor încapsulate. În mod similar, valorile variabilelor încapsulate pot fi obținute numai utilizând metode publice ale obiectului respectiv.

Încapsularea obiectelor prezintă avantaje importante în programare, deoarece mărește siguranța și fiabilitatea programelor, prin eliminarea posibilității modificării accidentale a valorilor acestora ca urmare a accesului neautorizat din exterior. Din această cauză, programatorii evită în general să prevadă într-un obiect date publice, preferând ca accesul la date să se facă numai prin metode.

Partea vizibilă (publică) a obiectului constituie interfața acestuia cu "lumea exterioară". Este posibil ca două obiecte diferite să aibă interfețe identice, adică să prezinte în exterior aceleași date și metode. Datorită faptului că partea încapsulată diferă, astfel de obiecte pot avea comportament diferit. Moștenirea este proprietatea unei clase de a conține toate atributele (variabilele) și metodele superclasei sale. În consecință, trecerea de la clasă la subclasă se face prin adăugarea de atribute și/sau de metode.

În general, în programarea orientată pe obiecte, moștenirea poate fi simplă sau multiplă. În cazul moștenirii simple fiecare clasă are cel mult o superclasă. În limbajul Java se admite numai moștenirea simplă. Lipsa moștenirii multiple este compensată în limbajul Java prin introducerea conceptului de interfață. (1) (2) (3)

3.2 Maven

3.2.1 Descriere

Maven este un sistem de build și management al proiectelor, scris în Java. Face parte din proiectele găzduite de Apache Software Foundation. Funcționalitățile sale principale sunt descrierea procesului de build a software-ului și descrierea dependențelor acestuia. Un fișier XML descrie proiectul care urmează să fie build-uit, dependențele acestuia sau ale modulelor și componente de care depinde, ordinea în care se execută build-ul, directoarele și plug-in-urile necesare. Maven descarcă dinamic bibliotecile Java și plug-in-uri necesare, din unul sau mai multe repository-uri. (4, 2016)

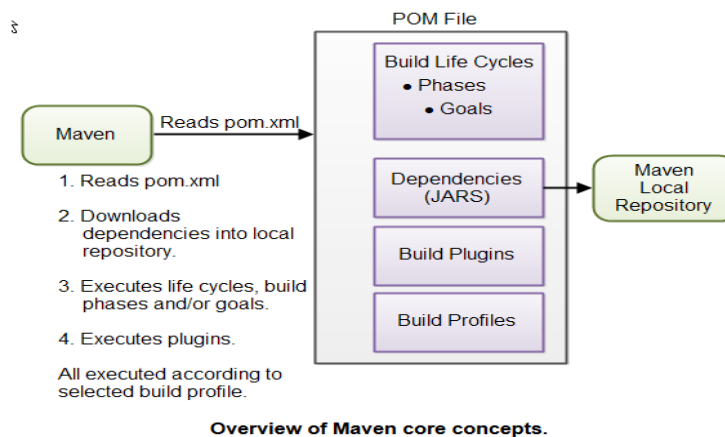
3.2.2 POM

POM reprezintă abrevierea de la Project Object Model. Este un fișier xml și reprezintă componenta principală în lucrul cu Maven. Pentru fiecare proiect Maven, există un singur fișier pom.xml. Înainte de a crea un fișier POM, trebuie să alegem un nume pentru proiect și un groupId.

```
<modelVersion>4.0.0</modelVersion>
<groupId>TFL</groupId>
<artifactId>TFL</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>
```

Figură 1. Crearea unui proiect Maven

Fișierul pom.xml conține referințe către toate resursele proiectului: Jar-URI externe, dependențe, directoarele în care se găsește codul de testare sau codul sursă. Acesta ar trebui să fie salvat în folderul principal al proiectului.



Figură 2. Conceptele Maven

În diagrama din Figura2 este evidențiat modul în care Maven utilizează fișierul pom.xml, precum și principalele componente ale fișierului. (5, 2016)

De exemplu, pentru a folosi framework-ul Primefaces, vom aduce dependențele sale cu ajutorul Maven-ului. În fișierul pom.xml din cadrul proiectului, adăugăm dependențele, precum în Figura 3.

```
8  <repositories>
9    <repository>
10      <id>prime-repo</id>
11      <name>Prime Repo</name>
12      <url>http://repository.primefaces.org</url>
13    </repository>
14  </repositories>
15
16  <dependencies>
17    <dependency>
18      <groupId>org.primefaces</groupId>
19      <artifactId>primefaces</artifactId>
20      <version>5.3</version>
21    </dependency>
22
23    <dependency>
24      <groupId>org.primefaces.themes</groupId>
25      <artifactId>all-themes</artifactId>
26      <version>1.0.10</version>
27    </dependency>
28  </dependencies>
```

Figură 3. Dependențe funcționale incluse cu Maven

La acțiunea de build, se vor aduce dependențele proiectului. De asemenea, se poate face build în diferite moduri: de exemplu pe un computer local, pentru dezvoltare și testare sau un build pentru implementare într-un sistem de producție. Pentru acest lucru se folosesc diferite profiluri de build, specificând la execuția Maven-ului profilul folosit. (6, 2016)

3.3 Java Server Faces

3.3.1 Prezentare generală

Java Server Faces este un framework folosit pentru dezvoltarea de aplicații web. A fost creat de Java Community Process (JPC) format din experți în dezvoltarea de aplicații web din diferite grupuri, ca: Jakarta Struts, Oracle, Sun, IBM, ATG etc. (7, 2016)

JSF face parte din șabloanele Web bazate pe MVC (Model-View-Controller) design pattern. Aplicațiile create folosind framework-ul JSF sunt ușor de dezvoltat și de întreținut comparativ cu alte aplicații create folosind JSP și Servlet. (8, 2016)

Ce oferă JSF:

- permite crearea UI folosind componente standard, reutilizabile
- permite accesarea și manipularea componentelor UI folosind taguri JSP
- salvează starea componentelor UI când clientul face o solicitare pentru o nouă pagină și o restaurează când solicitarea este returnată (asigură persistența stării la nivel Web)
- oferă un model de interacțiune bazat pe evenimente

- furnizează mecanisme pentru dezvoltarea de componente proprii
- separă prezentarea componentelor de funcționalitate, astfel încât acestea să poată fi utilizate în pagini HTML, WML, etc
- simplifică modul de creare a IDE-urilor pentru dezvoltarea de aplicații Web (9, 2016)

Componente JSF:

- Un set de API pentru reprezentarea componentelor interfeței utilizatorului (UI) și administrarea stării lor, tratarea evenimentelor și validarea intrărilor, convertirea valorilor, definirea navigării în pagini, și suport pentru internationalizare și accesibilitate
- Librării de taguri JSP folosite la crearea componentelor UI (10, 2016)

3.3.2 Caracteristici

JSF este un framework orientat pe partea de server, nu pe partea de client. Acest lucru înseamnă că în arhitectura JSF, majoritatea evenimentelor legate de UI management sunt tratate pe partea de server.

Un exemplu de framework UI orientat pe partea de client este Swing. (11, 2016)

O caracteristica importantă a framework-ului JSF este faptul că se separă tipurile de activități efectuate la crearea unei aplicații web:

- Design - editarea paginilor JSP, HTML
- Dezvoltare - implementarea logicii aplicației
- Crearea componentelor UI - componente ce vor fi utilizate de designeri pentru realizarea UI
- Arhitectura - asigurarea fluxului aplicației, a scalabilității, configurare, etc.

Caracteristici generale pe care funcționează framework-ul:

- Definirea regulilor de navigare între pagini – JSF are propriile componente care facilitează navigarea printre pagini și oferă posibilitatea controlului navigării.
- Validator – este componenta care se ocupă cu validarea datelor de intrare
- Convertors - JSF convertește valori de tip date și numere în valori ce pot fi afișate cu ușurință (strings)
- Messages – folosite pentru afișarea de mesaje de success/failure utilizatorului
- Beans – se ocupă de logica aplicației sunt obiecte de pe partea de server asociate unor UI components folosite în pagini web
- Event handling – JSF simplifică tratarea evenimentelor
- Internaționalizare

- Custom GUI controls – JSF oferă un set de API și taguri asociate pentru a crea forme HTML cu interfețe complexe

3.3.3 Arhitectura JSF

JSF este un User Interface framework orientat pe partea de server. În cazul în care un client face o cerere, cererea ajunge prin intermediul rețelei la server, unde framework-ul JSF construiește o reprezentare UI și o afișează clientului în mark-up language-ul corespunzător clientului. Userul interacționează cu pagina web și submite o cerere de procesare serverului. Framework-ul JSF interpretează parametrii cererii, îi decodează și îi convertește la evenimente apoi le propagă către logica de tratare a evenimentelor. (12, 2016)

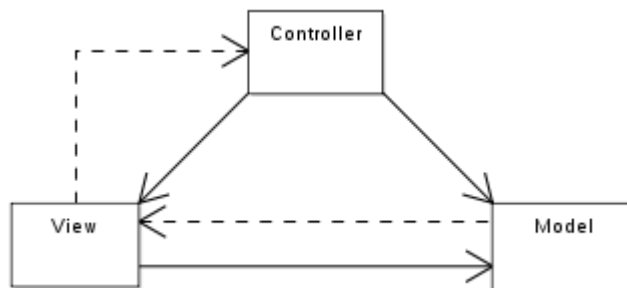
JSF este unul din framework-urile care se bazează pe structura Model-View-Controller(MVC). Ca orice alt framework bazat pe MVC, arhitectura JSF are propriul Front controller, numit FacesServlet. Rolul acestui controller este de gatekeeper. (13, 2016)

MVC este un model arhitectural folosit în ingineria software. El separă interfața aplicațiilor web în trei părți: model, view și controller, rezultând astfel o aplicație unde este mai ușor de modificat aspectul vizual sau nivelele inferioare ale regulilor de business fără a afecta alte nivele. Datele introduse de user, modelarea lumii externe și feedback-ul vizual către user sunt tratate separat de fiecare componentă a modelului. Controllerul interpretează intrările de la mouse și tastatură și mapează aceste acțiuni la comenzi care sunt trimise către model sau/și view pentru a efectua schimbările corespunzătoare. Modelul prelucrează unul sau mai multe elemente, răspunde la interogări asupra stării lui și răspunde la instrucțiuni de schimbare a stării. View-ul controlează o zonă din suprafața de afișat și este responsabil cu prezentarea datelor către user printr-o combinație de grafică și text.

Model-ul este responsabil cu reprezentarea datelor și a acțiunilor ce operează asupra datelor de la nivelul Web al aplicației. Este folosit pentru a controla informațiile și a notifica observatorii când acestea se modifică. Pe lângă controlarea informațiilor și notificarea observatorilor când starea acestora se modifică, modelul mai este folosit ca o abstractizare a proceselor sau sistemelor din lumea reală. El surprinde nu numai starea unui proces sau a unui sistem, ci și cum acesta funcționează.

View-ul este responsabil cu prezentarea rezultatelor către utilizator. View-ul are o relație de corespondență de 1 la 1 cu suprafața de afișare și știe cum să afișeze conținutul către utilizator. În plus, când modelul se modifică, view-ul automat redesenează partea afectată pentru a reflecta modificările apărute. Componenta View este creată uzual folosind: pagini JSP, taguri proprii, taguri definite de utilizator, JSTL, șabloane Velocity, Transformări XSLT.

Controler-ul reprezintă componenta prin care userul interacționează cu aplicația. Controller-ul primește informații de la user și instruește model-ul și view-ul să execute acțiuni bazate pe input-ul primit. În principiu, controller-ul este responsabil cu maparea acțiunilor efectuate de user la răspunsul care trebuie oferit de aplicație. De exemplu, dacă userul selectează un meniu item, controlerul trebuie să determine cum ar trebui aplicația să răspundă acestei acțiuni. (14, 2016)



Figură 4. Arhitectura MVC (Model-View-Controller)

Modelul, view-ul și controller-ul sunt strâns legate între ele și într-un continuu contact. Diagrama de mai sus ilustrează modul de comunicare dintre cele trei componente. Modelul comuniă cu view-ul, fără să știe informații despre acesta, transmitându-i notificări asupra schimbărilor întâmplare, iar view-ul comunică la rândul lui cu modelul, dar acesta știe tipul modelului observat, ceea ce îi oferă posibilitatea să apeleze metodele modelului. View-ul comunică de asemenea și cu controlerul, dar acesta nu-îi poate apela decât metodele din clasa de bază. Controlerul comunică cu amândouă: model-ul și view-ul și știe tipul ambelor componente, deoarece acesta trebuie să știe să răspundă corect la orice input primit de la user. (15, 2016)

Arhitectura componentelor UI cuprinde cinci tipuri de modele:

- Modelul claselor UI
- Modelul de prezentare
- Modelul de conversie
- Modelul de tratare a evenimentelor
- Modelul de validare

Modelul claselor UI :

- Descrie funcționalitatea componentelor UI prin extinderea clasei `UIComponent` (ex. `UICommand`, `UIForm`, `UIPanel`, `UIOutput`, `UIInput`, `UIMessage`, `UIColumn`, etc.)

- Implementează interfețe comportamentale precum: ActionSource, ValueHolder, StateHolder, NamingContainer
- Nu definesc reprezentarea grafică a componentelor

Modelul de prezentare:

- Definește reprezentarea grafică a componentelor UI prin folosirea de obiecte de tip Renderer – convertește componente din și în un limbaj markup specific
- Claselor de tip UIComponent le pot fi asociate mai multe tipuri de reprezentări

Modelul de conversie:

- Componentele UI pot avea asociate date, memorate la nivelul serverului într-un bean
- Datele componentelor sunt organizate astfel:
 - model: informația propriu-zisă, reprezentată printr-un tip de date Java (int, long, String);
 - prezentare: reprezentarea la nivel de client astfel încât să permită citirea/editarea informației
- JSF va face automat conversia între model și prezentare, și invers.

Modelul Event-Listener

- Interacțiunea utilizatorului cu componente UI generează evenimente
- La apariția unui eveniment, sunt notificate metodele claselor listener înregistrate pentru componenta sursa a evenimentului

Modelul pentru validare:

- Tipuri de validatori:
 - standard: referiți prin intermediul tagurilor de validare
 - JSF, include în corpul unui tag unei componente UI;
 - proprii: validarea din cadrul unei componente bean

3.3.4 Utilitatea framework-ului

Când experții JSF au început să lucreze la specificațiile JSF, aceștia voiau să creeze un framework care să îndeplinească anumite condiții. În primul rând să fie user-friendly, și să ofere posibilitatea de a crea aplicații web prin drag and drop de componente UI. O altă condiție este aceea de a se face distincție clară între UI component model și afișarea componentelor folosind orice tip de client și orice tip de protocol. În plus, JSF ar trebui să poată lucra cu JSP, dar și cu alte tehnologii. (16, 2016)

3.3.5 Utilizarea în cadrul unei aplicații

Pentru a folosi JSF într-o aplicație, în primul rând trebuie să îi aducem dependențele. În cazul aplicației curente, aducem dependențele cu ajutorul Maven-ului.

```
<!-- Faces Library -->
<dependency>
  <groupId>com.sun.faces</groupId>
  <artifactId>jsf-impl</artifactId>
  <version>2.2.4</version>
</dependency>

<dependency>
  <groupId>com.sun.faces</groupId>
  <artifactId>jsf-api</artifactId>
  <version>2.2.4</version>
</dependency>

<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>

<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>servlet-api</artifactId>
  <version>2.5</version>
</dependency>

<dependency>
  <groupId>javax.servlet.jsp</groupId>
  <artifactId>jsp-api</artifactId>
  <version>2.1</version>
</dependency>
```

Figură 5. Dependențele JSF

După încărcarea dependențelor se vor crea containerele (“Managed bean”), clasele în care se va găsi informația cu care lucrează utilizatorii și care poate fi accesată de către paginile JSF. Pentru a indica faptul că o clasă este un container se va folosi adnotarea ‘@ManagedBean’’. (17, 2016)

```
package views;

import javax.faces.application.FacesMessage;

@ManagedBean(name="singUpView")
public class SingUpView {

    private String name;
    private String pass;
    private String cpass;

    public void setCpass(String value)
    {
        this.cpass=value;
    }

    public String getCpass()
    {
        return this.cpass;
    }

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name =name;
    }

    public String getPass() {
        return this.pass;
    }
}
```

Figură 6. Crearea unui ManagedBean

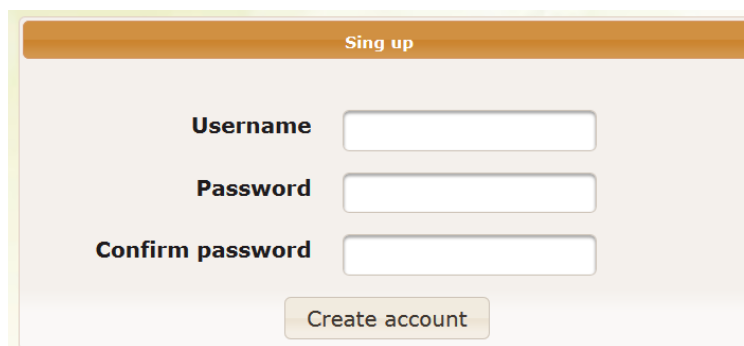
În final, putem folosi JSF în paginile HTML, ca în exemplul din Figura 7.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml"
5       xmlns:h="http://java.sun.com/jsf/html"
6       xmlns:f="http://java.sun.com/jsf/core"
7       xmlns:ui="http://java.sun.com/jsf/facelets"
8       xmlns:p="http://primefaces.org/ui">
9
10 <h:head>
11
12
13
14
15
16
17
18 <h:body class="bodystyle">
19
20 <div class="row biggerFont">
21 <div class="col-xs-5 col-md-5" style="text-align:right">
22 <p:outputLabel value="Username"></p:outputLabel>
23 </div>
24 <div class="col-xs-7 col-md-7" style="text-align:left">
25 <p:inputText id="name" value="#{singUpView.name}" required="true" >
26 <f:validateLength minimum="2"></f:validateLength>
27 </p:inputText>
28 </div>
29 </div>
30 </h:body>
```

Figură 7. Exemplu de folosire a JSF în HTML

Pagina web se conectează la container prin Expression Language (EL), prin expresia “#{singUpView.name}” se obține valoarea name din containerul ‘singUpView’. Dacă pentru clasă nu se specifică un nume alături de adnotarea @ManagedBean, containerul poate fi accesat prin numele clasei cu prima literă transformată din majusculă în literă mică. (18, 2016)

În browser, se va obține:



Figură 8. Exemplu formular JSF

3.4 Primefaces

3.4.1 Introducere

Primefaces este o bibliotecă ce furnizează componente UI care pot fi incluse în paginile JSF. Acesta a fost dezvoltat pentru Java Server Faces 2.0, și se bazează în mare măsură pe jQuery UI.

PrimeFaces adaugă suport suplimentar pentru tehnologii moderne, pe partea de sus a jQuery UI, adăugarea unor widget-uri suplimentare, permițând dezvoltatorilor să construiască interfețe bogate, interactiv cu utilizatorul. (19, 2016)

3.4.2 Setup

Primefaces are un singur jar numit 'primefaces-{version}.jar'. Acesta poate fi downloadat de pe <http://www.primefaces.org/downloads> sau poate fi downloadat cu ajutorul Maven-ului, adăugând dependența sa în fișierul pom.xml.

3.4.3 Dependențe

Pentru a putea fi folosit în dezvoltarea aplicațiilor, framework-ul necesită Java 5+ și JSF 2.x. Pentru mai multe funcționalități, se pot include și dependențe opționale. Nu avem nevoie de configurații suplimentare, însă putem face acest lucru în funcție de cerințe.

Primefaces cuprinde mai mult de 30 de teme predefinite. Pentru a putea folosi o temă pentru paginile web, putem adăuga dependența în fișierul pom.xml.

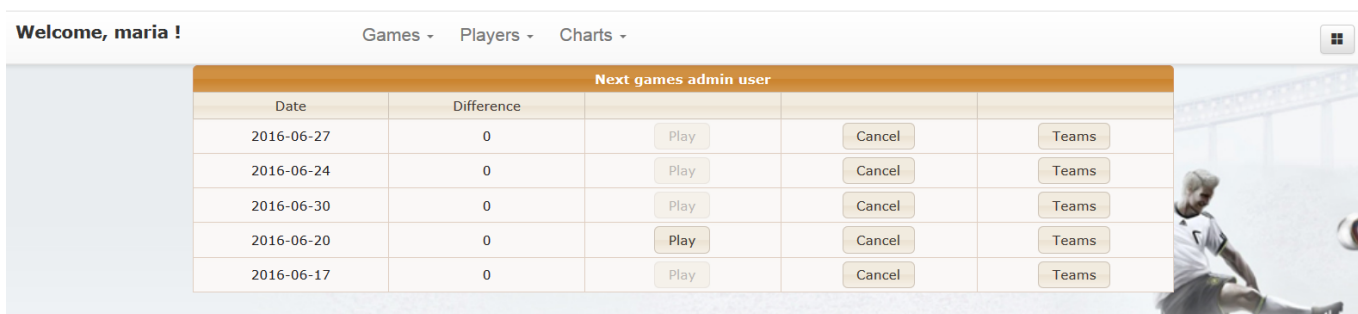
```
<dependency>
  <groupId>org.primefaces.themes</groupId>
  <artifactId>all-themes</artifactId>
  <version>1.0.10</version>
</dependency>
```

Figură 9. Adăugarea dependenței pentru temele Primefaces

Pentru a seta o temă pentru paginile web, o adăugăm în fișierul web.xml. (20, 2016)

```
<context-param>
  <param-name>primefaces.THEME</param-name>
  <param-value>humanity</param-value>
</context-param>
```

Tema setată pentru pagina web va arăta precum în Figura10.



Figură 10. Tema 'Humanity' Primefaces

3.4.4 Componente Primefaces

Componentele Primefaces pot fi folosite alături de mai multe atribute. Câteva dintre atributele ce pot fi folosite sunt: (21, 2016)

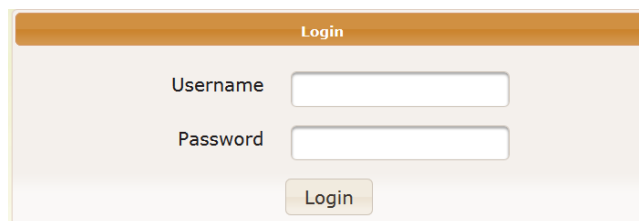
- Id(string), reprezentând un identificator unic pentru componentă
- Rendered(Boolean), în cazul în care este false componenta nu este vizibilă în pagină
- Binding(Object) reprezintă componenta din container asociată butonului
- ActionListener(MethodObj) reprezintă acțiunea ce va fi executată atunci când se dă click pe buton
- Style(String) stilul ce poate fi declarat inline
- StyleClass(String) reprezintă clasa stilului pentru element
- Multiple(Boolean) pentru componente de selecție în cazul în care se dorește selectarea multiplă

3.4.4.1 Panel

Pentru autentificare, se completează câmpurile din formularul ce va conține un panel, componenta Primefaces folosită pentru a grupa elementele, care are sintaxa:

```
<p:panel>  
    //child components here...  
</p:panel>
```

Rezultatul obținut va fi cel din Figura 11.

The image shows a Primefaces Panel component. It has a title bar at the top with the text "Login". Inside the panel, there is a form with two input fields: "Username" and "Password". Below these fields is a "Login" button. The panel has a light gray background and a thin border.

Figură 11. Primefaces Panel

3.4.4.2 CommandButton

Dupa autentificare, din meniu, utilizatorul poate alege acțiunile pe care dorește să le execute, prin click pe un item al meniului, reprezentat de un commandButton. Tag-ul Primefaces este `<p:commandButton></p:commandButton>` și poate avea diferite atribute și stiluri css. (22, 2016)

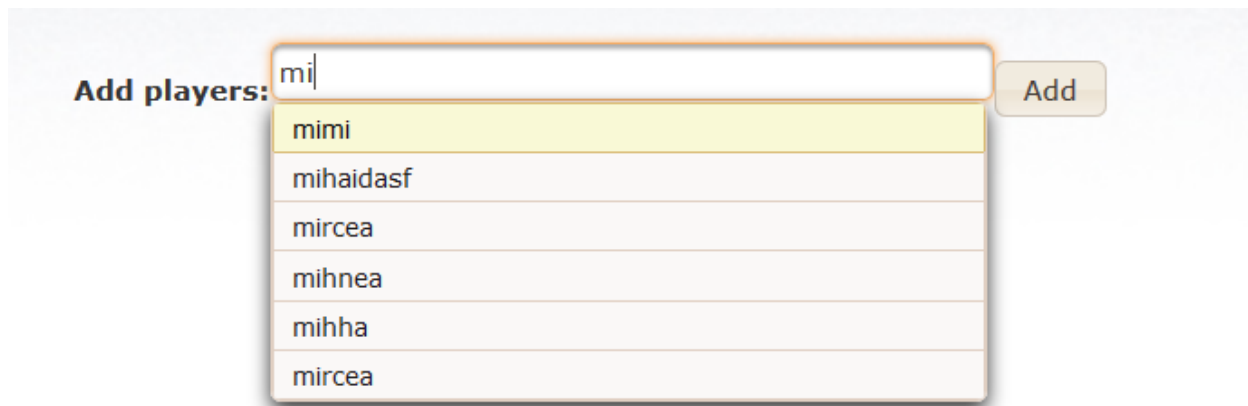
3.4.4.3 AutoComplete

Componenta folosită pentru a afișa o listă de sugestii în timpul scrierii unui input de către utilizator.

Sintaxa este:

```
<p:autoComplete id="themes" multiple="true" value="#{ autoCompleteView.selectedPlayers}"
completeMethod="#{ autoCompleteView.completeTheme}" var="theme"
itemLabel="#{ theme.username}" itemValue="#{ theme}" converter="playerConverter"
forceSelection="true">
    <p:column style="width:80%">
        <h:outputText value="#{ theme.username}"/>
    </p:column>
</p:autoComplete>
```

Rezultatul afișat pe interfața va fi:



Figură 12. Primefaces Autocomplete

3.4.4.4 Chart

Chart-ul este o componentă grafică care folosește librăria jqplot pentru afișarea datelor într-un format interactiv. Există mai multe tipuri de chart-uri: Pie Chart, Line Chart, Bar Chart, Area.

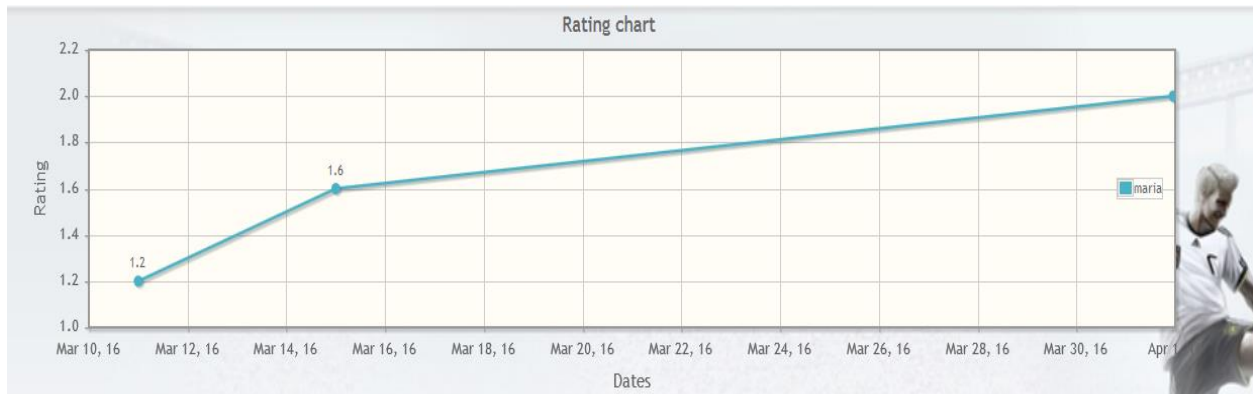
Codul html pentru un LineChart ar putea fi:

```
<div class="col-sm-12 col-md-12">
    <p:chart type="line" model="#{chartView.LineModel}" style="height:300px;"/>
</div>
```

Unde 'lineModel' este creat în controller, clasa adnotată cu @ManagedBean, ChartView, și care conține atributul:

```
private LineChartModel lineModel;
```

Rezultatul obținut este:



Figură 13. Primefaces Chart

3.4.4.5 FileUpload

FileUpload adaugă un plus de inovație și flexibilitate față de componenta `<input>` de tip "file" `<input type="file">` din HTML5. Pentru partea de server pe care se face upload se poate folosi servlet 3.0 sau alt tip de server pe care se poate face upload. Acesta detectează automat cel mai potrivit tip de server pe care să se faca upload, dar se pot adăuga și alte configurări:

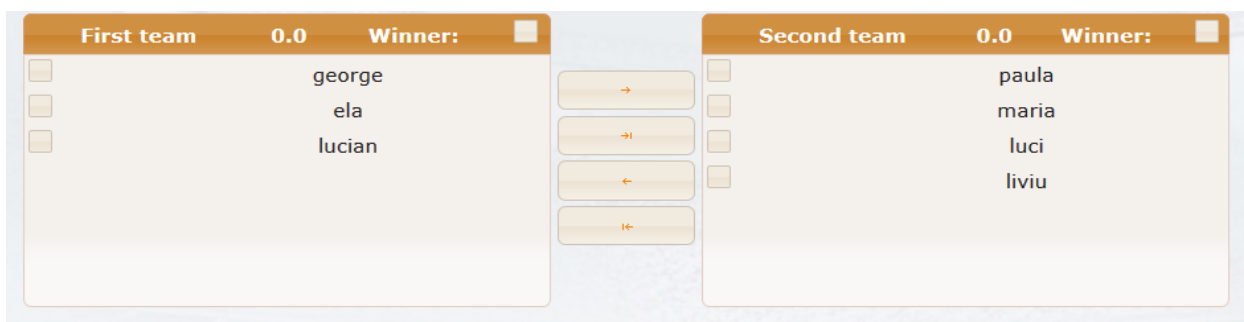
```
<context-param>
    <param-name>primefaces.UPLOADER</param-name>
    <param-value>auto|native|commons</param-value>
</context-param>
```

3.4.4.6 PickList

PickList este folosită pentru a putea transfera date între două colecții diferite.

Sintaxa componentei este:

```
<p:pickList id="pick" value="#{teamsView.players}" var="theme" effect="bounce"
itemValue="#{theme}" itemLabel="#{theme.username}" showCheckbox="true"
responsive="true" filterMatchMode="contains" converter="playerConverter">
</p:pickList>
```



Figură 14. Primefaces PickList

3.5 Bootstrap

3.5.1 Prezentare generală

Bootstrap este un framework de front-end, folosit pentru dezvoltarea aplicațiilor web și a website-urilor. Conține template-uri de HTML și CSS pentru componente din interfața, precum input-uri, butoane, formulare, componente pentru navigarea în pagină, dar și extensii de JavaScript. Spre deosebire de alte framework-uri, Bootstrap este orientat în special către partea de interfața și dezvoltarea front-end.

Numit inițial **Twitter Blueprint**, a fost dezvoltat de către Mark Otto și Jacob Thornton ca și framework ce avea ca scop consecvența între diferite tool-uri interne, înainte de Bootstrap fiind folosite mai multe librării pentru a realiza partea de interfața. Folosirea unei multitudini de tipuri de librării avea ca rezultat aplicații greu de menținut și cu o flexibilitate redusă.

Bootstrap este compatibil cu cele mai noi versiuni de Google Chrome, Firefox, Opera, Internet Explorer și Safari, deși nu toate browsere-le sunt suportate de diferite platforme.

Începând cu versiunea 2.0, Bootstrap are de asemenea suport pentru un design web responsive. Acest lucru înseamnă că paginile își ajustează conținutul în funcție de tipul de device de pe care sunt accesate (desktop, tablete, telefoane mobile).

Framework-ul este disponibil open-source pe GitHub, dezvoltatorii fiind încurajați să contribuie la dezvoltarea acestuia.

3.5.2 Structură și funcționalitate

Bootstrap este împărțit pe module, și constă în principal dintr-o serie de stylesheet-uri care implementează o multitudine de componente din setul de instrumente. Dezvoltatorii pot adapta Bootstrap-ul prin selectarea componentelor pe care doresc să le utilizeze. În plus față de elementele de bază de HTML, framework-ul aduce elemente speciale, precum butoane cu aspect și funcționalitate specifică (ex: butoane grupate, butoane cu opțiunea de drop-down), liste de navigare sau tab-uri orizontale sau verticale. Componentele sunt implementate ca și clase CSS, care trebuie atribuite elementelor din HTML. (23, 2016)

Framework-ul mai conține și o serie de scripturi JavaScript, incluse ca și plug-in-uri de JQuery. Acestea aduc în plus alte elemente de interacțiune cu utilizatorul, precum ferestre de dialog, tooltip-uri sau un mod interactiv de vizualizare a imaginilor, numit carusel.

3.5.3 Descărcare și instalare

Pentru a putea folosi Bootstrap, îl putem descărca de pe repository-ul de GitHub, accesând link-ul: <http://getbootstrap.com/getting-started/>, sau îl putem include dependențele în tag-ul <head> din pagina html. (24, 2016) Dependențele ce trebuie incluse sunt următoarele:

```
<!-- Latest compiled and minified CSS -->
<link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
      integrity="sha384-1q8mTJOASx8j1Au+a5WDVnPi2lkFfwwEAa8hDDdjZlpLegxhjVME1fgjWPGmkzs7"
      crossorigin="anonymous">

<!-- Optional theme -->
<link rel="stylesheet"
      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap-theme.min.css"
      integrity="sha384-fLW2N01lMqjakBkx3l/M9EahuwPsfENvV63J5ezn3uZzapT0u7EYsXMjQV+0En5r"
      crossorigin="anonymous">

<!-- Latest compiled and minified JavaScript -->
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"
        integrity="sha384-0mSbJDEHialfmUBBQP6A4Qrprq5OVfW37PRR3j5ELqXsslyVqOtnepnHVP9aJ7xS"
        crossorigin="anonymous"></script>
```

3.6 PostgreSQL

PostgreSQL este un sistem de baze de date relaționale. Este disponibil gratuit sub o licență open-source de tip BSD. PostgreSQL nu este controlat de nici o companie, își bazează dezvoltarea pe o comunitate răspândită la nivel global, precum și câteva companii dezvoltatoare. (25, 2016)

3.6.1 Caracteristici

Sistemul are câteva limitări generale:

- Dimensiunea maximă a bazei de date: nelimitat
- Dimensiunea maximă a unei tabele: 32 TB
- Dimensiunea maximă a unei înregistrări: 1,6 TB
- Dimensiunea maximă a unui câmp: 1 GB
- Număr maxim de înregistri într-o tabelă: nelimitat
- Număr maxim de coloane într-o tabelă: 250 - 1600 în funcție de tipul coloanelor

PostgreSQL permite folosirea limbajelor procedurale (26, 2016) pentru a executa blocuri de cod direct în serverul de baze de date. Se pot folosi pentru a crea funcții definite de utilizator (subrutine, triggerre, agregate și funcții fereastră) sau pentru a crea blocuri ad hoc "DO". Instalarea standard a PostgreSQL permite utilizarea următoarelor limbaje:

- PL/pgSQL un limbaj asemănător cu PL/SQL existent în Oracle.
- PL/Tcl pune la dispoziție Tcl
- PL/Perl pune la dispoziție Perl
- PL/Python pune la dispoziție Python, versiunea 2 sau 3.

Alte limbaje disponibile în afara pachetului de bază includ: PL/Java, PL/php, PL/Ruby, etc.

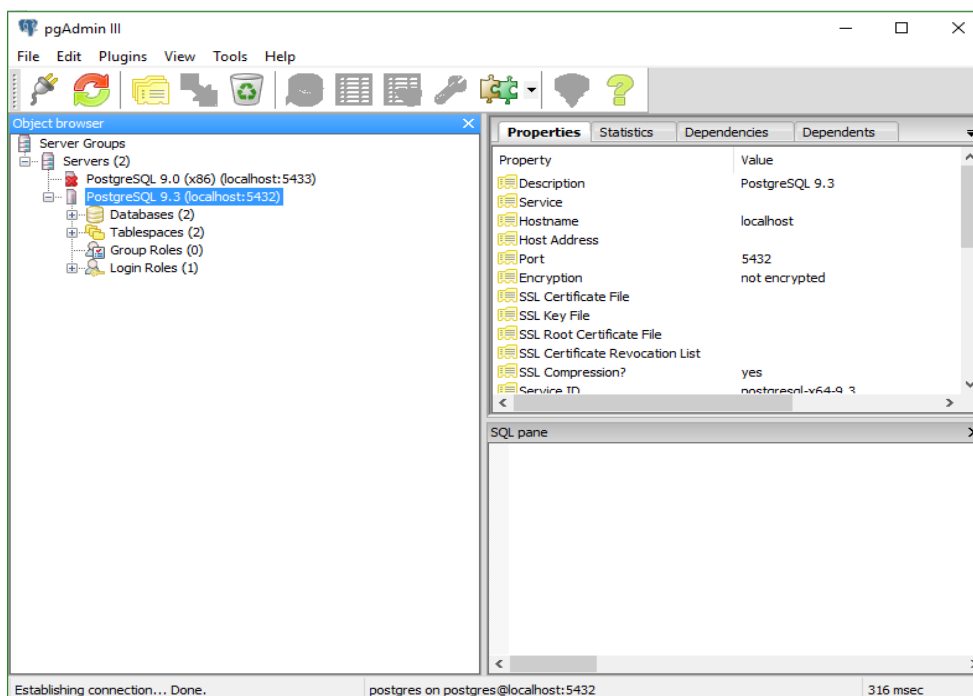
La fel ca și alte baze de date, PostgreSQL permite utilizarea indecșilor pentru accelerarea interogărilor. Suportă mai multe tipuri de indecși: B-tree, Hash, GiST sau GIN. Dacă tipul nu este specificat la crearea indexului, se utilizează B-tree.

Alte caracteristici suportate:

- Valorile indecșilor pot fi calculate printr-o expresie sau o funcție.
- Indecși parțiali permit să se indexeze doar o parte dintr-o tabelă. Pot fi creați prin specificarea unei clauze `WHERE` la sfârșitul unei comenzi `CREATE INDEX`.
- Planificatorul este capabil să folosească mai mulți indecși pentru a executa interogări complexe.

3.6.2 pgAdmin

PgAdmin este una dintre cele mai populare platforme open-source de administrare și dezvoltare a bazelor de date PostgreSQL. PgAdmin pune la dispoziție un design capabil să rezolve o multitudine de probleme ale utilizatorilor, de la query-uri simple, până la baze de date foarte complexe.



Figură 15. Interfața pgAdmin

Interfața grafică suportă toate caracteristicile principale ale limbajului și face ca administrarea bazei de date să devină un proces simplu și intuitiv. Conexiunea la baza de date se poate face cu ajutorul protocoalelor precum TCP/IP sau IPC(inter-process communication socket) și poate fi criptată folosind SSL pentru asigurarea securității datelor.

3.7 JPA(Java Persistence API)

JPA este un framework lightweight ce folosește POJO (Plain Old Java Objects) pentru a persista obiecte Java ce reprezintă date relaționale. JPA 1.0 a început ca parte a specificațiilor EJB 3.0 pentru a standardiza un model pentru ORM (object-relational mapping). JPA 2.0 (JSR-317) îmbunătățește specificațiile originale. (27)

POJO face parte, de asemenea, din specificațiile EJB 3.0. Orice obiect normal este un obiect POJO

Beneficiile utilizării JPA:

- Nu trebuie să cream obiecte complexe de acces la date (DAO)
- API-ul este folosit pentru a gestiona tranzacții
- Codul de interacțiune cu baza de date este standard, indiferent de vendorul bazei de date relaționale
- Putem evita SQL și în schimb putem folosi un query language, orientat pe obiect
- Putem folosi JPA pentru persistența aplicațiilor desktop

JDBC a fost primul mecanism pe care dezvoltatorii Java l-au folosit pentru a persista date. JPA este un framework de mapare ce păstrează abilitatea de a manipula baza de date direct.

Conceptele fundamentale ale JPA sunt:

- Entitate, este utilizată pentru a reprezenta un tabel relațional într-un obiect Java
- Unitate de persistență, definește mulțimea tuturor claselor ce au legătură cu aplicația și care sunt mapate unei singure baze de date
- Context persistent, este o mulțime de instanță de entități în care este o unică instanță a entității pentru orice identificator de entitate persistentă
- Entity manager, face munca de creare, citire și scriere a entităților

3.7.1 Componente JPA

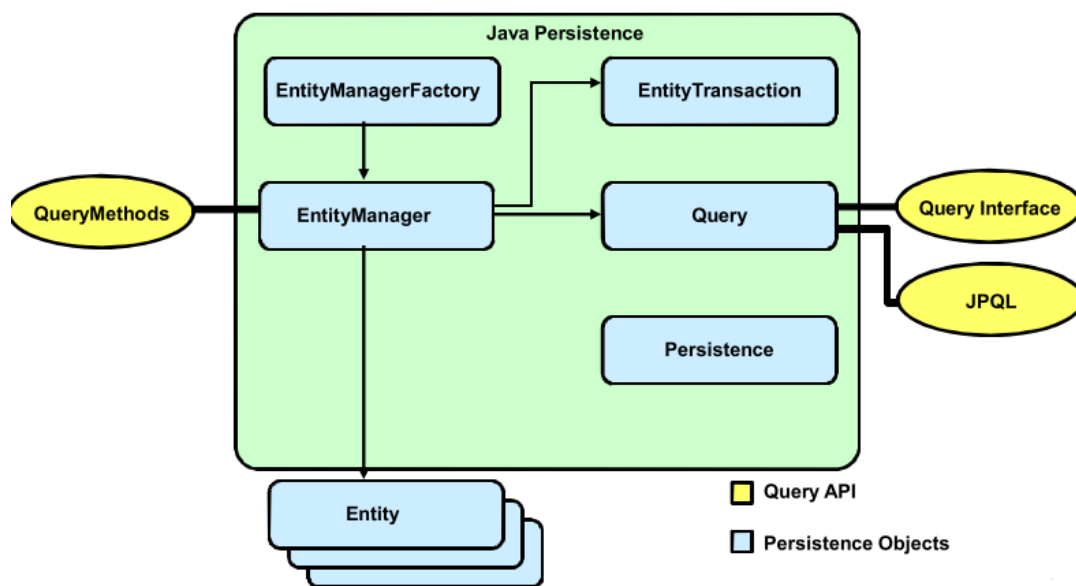
Persistența este mecanismul utilizat de aplicații pentru a păstra datele, date ce altfel ar fi pierdute la închiderea aplicației sau a calculatorului, într-un context persistent, precum o bază de date. (7, 2016)

JPA stabilește relaționări statice ale modelului persistent, prin definirea componentelor entitate. API-ul definește clasa entitate ca un echivalent al unei tabele din baza de date, pe partea de business a aplicației. O instanță a entității este definită ca un obiect, echivalent al unei linii din tabela bazei de date.

JPA stabilește relaționarea dinamică a modelului persistent prin definirea unui obiect entity manager. Acesta este însărcinat cu sincronizarea datelor continute într-o instanță entitate cu datele conținute în linia echivalentă din baza de date. Spre exemplu, dacă pe timpul execuției aplicației un camp al instanței entitate este modificat, entity managerul modifica linia echivalenta din baza de date.

JPA furnizeaza programatorilor Java facilitatea de mapare obiect/relație, pentru a gestiona modelul relațional al bazelor de date implicate în aplicațiile Java. Persistența în Java constă din:

- JPA
- Limbajul de interogare (JPQL)
- Java Persistence Criteria API
- Metadatele de mapări obiect/relație



Figură 16.Arhitectura JPA

În Figura16 avem:

- Entity, obiectul persistent ce reprezintă o înregistrare din tabela bazei de date. Este un POJO cu anotații
- Interfața EntityManager, furnizează API-ul de interacțiune cu entitatea

- EntityManagerFactory este folosit pentru a crea o instanță de EntityManager.

Un obiect Java poate cuprinde date parțiale dintr-o tabelă sau date din mai multe tabele. Framework-urile ORM gestionează maparea între tabelele bazei de date relaționale și obiectele Java astfel încât programatorii să codeze cât mai puțin. EclipseLink și Hibernate sunt exemple de soft-uri ORM.

3.7.1.1 Entități

Așadar, o entitate reprezintă o structură de date prin intermediul căreia se asigură persistența. O entitate este asociată unui tabel al unei baze de date relaționale, iar fiecare instanță a entității corespunde unei linii din tabelă. Câmpul unei clase entitate corespunde unei coloane din tabelă bazei de date.

O entitate:

- Este un POJO creată utilizând cuvântul new
- Suportă moștenire și polimorfism
- Este serializată și poate fi utilizată ca un obiect detached

3.7.1.2 Relaționarea entităților

Primul pas în modelarea asocierii la nivelul datelor este definirea relaționării ca o mulțime de proprietăți. Aceste proprietăți pot fi apoi utilizate pentru a implementa relaționarea obiectelor prin clasele entitate.

Următoarele patru proprietăți sunt folosite pentru a descrie asocierea între obiecte:

- Cardinalitatea unei relații, care specifică numărul de relaționări între două entități relaționate, poate fi de următoarele tipuri: unu-unu, unu-mulți, mulți-unu, mulți-mulți.
- Direcția unei relații, care determină navigabilitatea și vizibilitatea, poate fi:

Bidirecțională, fiecare entitate poate vedea cealaltă entitate din relație. Putem astfel include cod în oricare dintre entități pentru a naviga către cealaltă entitate în vederea obținerii de informații și servicii de la această. În JPA natura bidirecțională a unei relații este specificată prin utilizarea unei anotații de cardinalitate în ambele clase entitate situate de cele două părți ale relației.

Unidirecțională, doar o singură entitate dintre cele două implicate în relație poate vedea cealaltă entitate. Natura unidirecțională a relației este specificată prin folosirea uneia dintre anotațiile de cardinalitate într-una dintre entități.

Proprietarul relației, specifică partea de owning a relaționării, care la rândul său conține maparea fizică. Cealaltă parte a relației se numește inverse side. Tipul de propagare a operației (cascading type), se

referă la propagarea efectului unei operații entitatilor asociate. Se exprimă prin termenii: All, Persist, Merge, Remove, Refresh, None.

Într-o relație bidirecțională fiecare entitate are un câmp sau o proprietate ce referă la cealaltă entitate din relație. Prin acest câmp codul entității poate accesa obiectul relaționat.

Relațiile bidirecționale trebuie să urmărească regulile:

- inverse side trebuie să refere către owning side utilizând elementul mappedBy al uneia dintre anotațiile: @OneToOne, @OneToMany sau @ManyToMany. Valoarea acestui element este câmpul sau proprietatea din entitatea ce joacă rol de owner al relației
- partea “multi” a unei relații multi-unu, într-o relaționare bidirecțională, nu trebuie să definească elementul mappedBy. Această parte este considerată întotdeauna owner
- într-o relație unu-unu bidirecțională owning side corespunde părții ce conține cheia străină corespunzătoare
- într-o relație multi-multi ambele părți pot fi owning side

Într-o relație unidirecțională doar o singură entitate are un câmp sau o proprietate ce referă la cealaltă entitate.

Direcționalitatea este necesară pentru a putea naviga între relații. Entitățile ce utilizează relațiile au deseori dependențe relativ la existența celeilalte entități din relație.

3.7.1.3 Gestiunea entităților

Entitățile sunt gestionate printr-un entity manager reprezentat de o instanță a interfeței `javax.persistence.EntityManager`. Fiecare entity manager este asociat unui context

persistent, adică instanțe ale unora dintre entități ce există mapate dintr-o bază de date. Contextul persistent definește domeniul sub care instanțele entităților sunt create, salvate sau șterse.

`EntityManager` definește metodele utilizate pentru a interacționa cu contextul persistent.

În cazul unui entity manager gestionat de container, o instanță a `EntityManager` este automat propagată de către container către toate componentele aplicației ce utilizează instanța în cadrul unei tranzații JTA. (28)

Tranzațiile JTA implică deseori apeluri peste mai multe componente. Pentru a executa o tranzație JTA aceste componente au nevoie de acces către un singur context persistent. Aceasta apare atunci când `EntityManager` este injectat în componentele aplicației, prin anotația `javax.persistence.PersistenceContext`. Acest context persistent este propagat automat împreună cu

tranzacția curentă JTA, iar referințele lui EntityManager sunt mapate aceluiași context persistent ca și tranzacția. Prin propagarea automată a contextului persistent componentele aplicației nu au nevoie să-și trimită referințele către instanțele EntityManager în cadrul unei tranzacții. Containerul Java EE gestionează ciclul de viață entity managerilor gestionați de container.

Obținerea unei instanțe se face prin construcția sintactică:

```
@PersistenceContext  
EntityManager em;
```

3.7.1.4 Folosirea JPA într-o aplicație Java SE

Pașii ce trebuie urmați pentru a include și utiliza JPA într-o aplicație sunt următorii: (5, 2016)

1. Creăm baza de date pe serverul de baze de date
2. Adăugăm bibliotecile EclipseLink și conectorul la BD
3. Definim entitățile în aplicație
4. Creăm persistence.xml și-l configurăm: definim o unitate de persistență și un tip de tranzacție, furnizăm nume claselor entitate, definim proprietățile conexiunii JDBC
5. Creăm instanțe ale EntityManagerFactory și EntityManager
6. Scriem cod pentru a efectua persistența entităților folosind instanța entity manager

3.7.1.5 Tranzacții

O tranzacție este un mecanism ce manipulează un grup de operații ca și cum ar fi o singură operație. Într-o tranzacție toate operațiile se execută sau niciuna. Operațiile implicate într-o tranzacție se pot baza pe mai multe baze de date. (29, 2016)

O tranzacție este formal definită ca o mulțime de proprietăți incluse în acronimul ACID:

- Atomicitate: o tranzacție este efectuată integral sau deloc. În caz de eșec operațiile și procedurile se consideră neefectuate și toate datele revin la starea anterioară tranzacției
- Consistență: o tranzacție transformă un sistem dintr-o stare consistentă într-o altă stare consistentă
- Izolare: fiecare tranzacție se efectuează independent de alte tranzacții ce se efectuează în același timp
- Durabilitate: tranzacțiile efectuate cu succes devin permanente chiar dacă sistemul eșuează

Operațiile pe entitate sunt tranzacționale, adică este necesar să fie parte a unei tranzacții. Avem două modele tranzacționale suportate de JPA:

- resource-local, reprezintă tranzacțiile native suportate de driver-ele JDBC în unitatea de persistență
- JTA, sunt parte a server-ului Java EE
- Interfața EntityTransaction suportă tranzacții resource-local, și este obținută din EntityManager prin apelul metodei `getTransaction()`.

Metode obișnuite ale unei tranzacții includ: `begin()`, pornesc un nou context tranzacțional, `commit()`, încheie contextul tranzacțional curent și scrie orice modificări nescrise încă în baza de date, `rollback()`, readuce tranzacția la starea inițială. (30) (31)

4 Descrierea soluției

4.1 Stabilirea obiectivelor

Aplicația are ca obiectiv principal formarea unor echipe de fotbal “echilibrate” și stabilirea programului pentru meciurile de fotbal. Acest lucru va duce la organizarea mai eficientă a grupului, fiecare membru fiind la curent cu toate evenimentele următoare și cu schimbările apărute, putând de asemenea interveni și participa la organizarea meciurilor de fotbal.

Un alt obiectiv îl reprezintă comunicarea eficientă în cadrul unui grup de indivizi, grup structurat pe diferite criterii. Se dorește ca aplicația să faciliteze transmiterea corectă și rapidă de informații în cadrul grupului.

De asemenea, aplicația se dorește a fi un instrument pentru păstrarea istoricului informațiilor, atunci când are loc o modificare a acestora. Utilizatorii vor putea consulta istoricul datelor, făcând modificări ce vor fi persistate într-o bază de date.

Dinamicitatea grupului este un alt obiectiv important. Pentru a păstra această caracteristică, membrii grupului trebuie să aibă posibilitatea de a modifica structura grupului, prin introducerea de noi membrii, sau prin eliminarea celor care nu mai doresc să facă parte din grup.

4.2 Cerințele utilizatorilor față de aplicație

În ceea ce privește utilizatorii aplicației, aceștia utilizează aplicația pentru a se putea organiza într-un mod eficient, urmând totodată anumite reguli unanim acceptate.

În primul rând, se stabilesc niște criterii conform cărora o echipă să se poată considera ‘echilibrată’ sau nu. Astfel, putem spune că două echipe sunt cu atât mai “echilibrate” cu cât diferența dintre suma rating-urilor jucătorilor din cele două echipe este mai mică. Se dorește ca acest lucru să fie realizat atât în funcție de rating-ul fiecărui jucător, cât și în funcție de preferințele acestuia, în mod automat, fără a fi nevoie ca împărțirea pe echipe să se facă manual, până la obținerea “echilibrului”.

Un altă cerință este reprezentată de păstrarea istoricului jocurilor, dar și evidența evoluției jucătorilor, aceștia primind o evaluare după fiecare meci câștigat sau pierdut. După fiecare meci jucat, se dorește păstrarea rezultatului. Rezultatul va fi un număr, ce va reprezenta diferența dintre numărul de goluri înscrise de fiecare echipă. În funcție de rezultat, atât pentru jucătorii echipei câștigătoare, cât și pentru cei din echipa învinsă, ratingul se va modifica. Astfel, după un meci jucat, fiecare jucător va avea un nou rating, reprezentând evoluția, sau involuția acestuia.

De asemenea, se dorește ca accesul la datele din cadrul aplicației să se facă ținând cont de drepturile de acces ale fiecărui utilizator. Astfel, la înscriere, un utilizator nu are dreptul de a stabili și organiza echipe, el poate doar să vadă următoarele meciuri și să se înscrie la cele la care dorește să participe. Poate vizualiza istoricul meciurilor sale, precum și graficul ce va reprezenta evoluția/involuția sa, realizat pe baza ratingului pe diferite perioade de timp. Un utilizator cu drepturi de administrator are acces la date despre toți jucătorii, poate elimina(logic) un jucător din baza de date, sau poate adăuga noi jucători. Tot el este cel care va stabili datele pentru meciurile următoare, va introduce scorul obținut de echipe și va putea să vizualizeze diagrame de rating pentru alți utilizatori, înafară de el însuși.

Pentru a se genera echipele în mod automat, trebuie ca la meci să fie înscrise minim 4 jucatori. Numărul maxim de jucători cu care se pot forma echipe este 16. În momentul în care s-a atins numărul de jucători necesari, un utilizator cu drepturi de administrator poate alege opțiunea de a genera echipele. În acest mod, se vor genera toate perechile de echipe posibile, care vor fi sortate crescător, după diferența dintre suma ratingurilor jucătorilor fiecărei echipe, astfel încât un utilizator de tip administrator poate alege oricare din perechile generate, chiar dacă aceasta nu este cea mai “echilibrată”.

Înregistrarea unui nou jucător se poate face atât de către un utilizator al aplicației care deține drepturi pentru a face acest lucru, cât și de user-ul care dorește crearea unui cont.

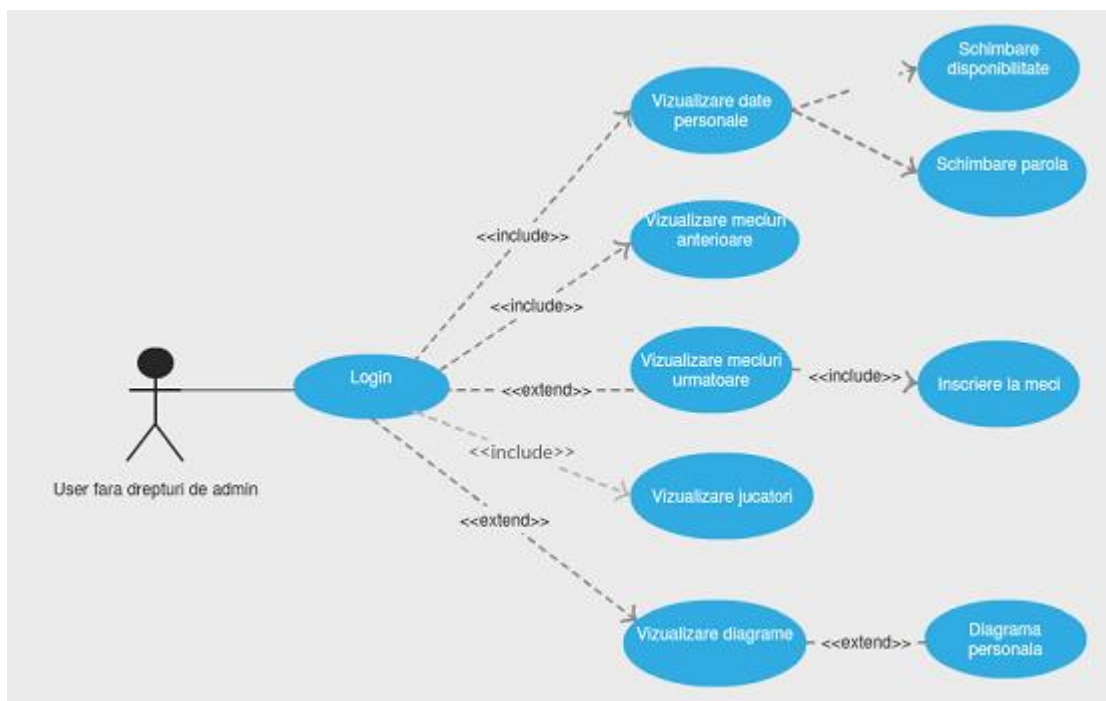
Un alt scop al aplicației este păstrarea istoricului datelor, pentru a putea fi consultate mai târziu, dar și adăugarea de date noi atât pentru userii existenți, cât și pentru activitățile în desfășurare. Datele pot fi vizualizate sub diferite forme: liste, diferite tipuri de grafice și pot fi salvate în diferite formate.

Fiecare utilizator are un drept de acces, care îi permite să realizeze diferite interacțiuni cu ajutorul interfeței web. Astfel, unui utilizator i se pot atribui drepturi de administrator de către alți utilizatori de acest tip, pentru a putea realiza interacțiuni suplimentare precum ștergerea logică a unui utilizator din baza, anularea unui joc sau realizarea echipelor.

La înregistrare, viitorul utilizator trebuie să își aleagă un username și o parolă, rating-ul sau fiind setat initial 5 în mod automat. Dacă înregistrarea s-a făcut cu succes, user-ul se va putea autentifica și va avea acces la datele din aplicație.

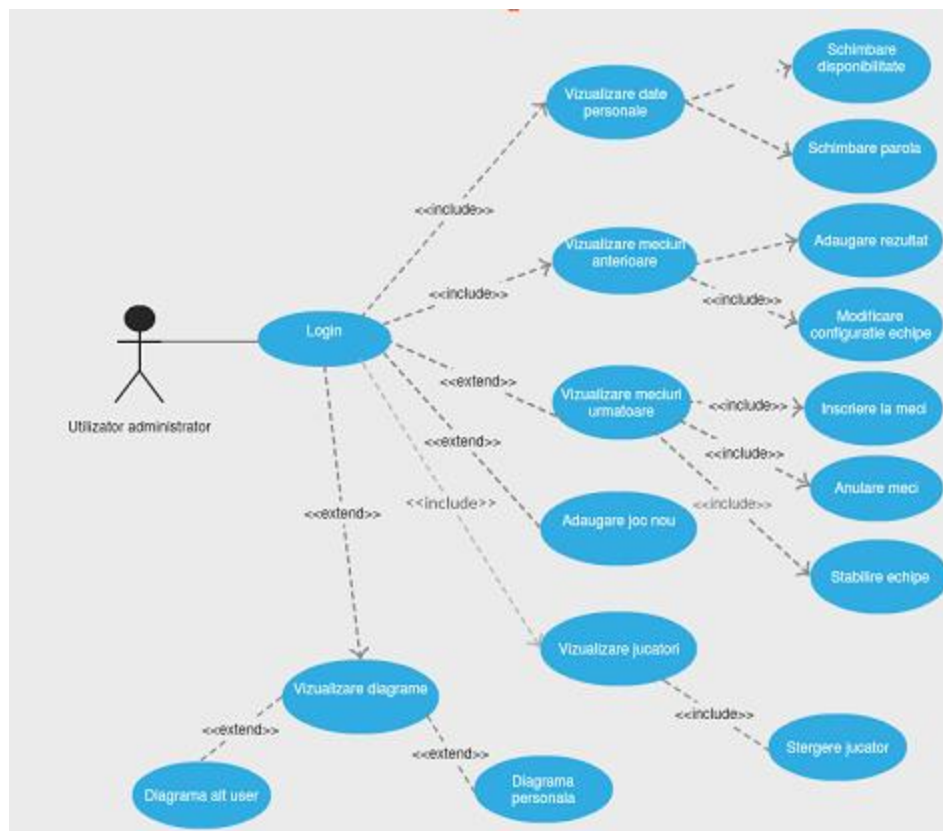
Opțiunile pe care acesta le are sunt structurate sub forma unui meniu. Astfel, un user fără drepturi de admin poate să vadă meciurile următoare și istoricul meciurilor, și se poate înscrie pentru a participa la un meci următor sau poate anula participarea sa la un joc. Își poate consulta datele personale și își poate actualiza poza de profil, disponibilitatea, sau parola. Are de asemenea acces la datele publice ale

celorlalți utilizatori, precum numele, rating-ul curent al acestora și poza de profil, însă doar pentru vizualizare. O altă opțiune pe care o poate alege este cea de a-și vizualiza diagrama de rating.



Figură 17. Diagrama use-case utilizator fără drepturi de administrator

Unui utilizator de tip administrator, i se pun la dispoziție câteva opțiuni în plus, precum adăugarea unui nou jucător și stabilirea datei/datelor pentru următoarele meciuri. De asemenea, acesta are acces la diagramele de rating pentru toți utilizatorii, poate vizualiza și poate elimina (la nivel logic), din baza de date un jucător. În plus, are acces la opțiunea de generare a echipelor, validând formatul echipelor și informând ceilalți participanți înscriși. Tot el este cel care poate anula un meci următor și poate introduce rezultatul unui anterior. Are posibilitatea de a schimba configurația echipelor stabilite până în momentul în care se loghează și de a le persista.



Figură 18. Diagrama use-case utilizator de tip administrator

4.3 Diagrama claselor

Codul sursă al server-ului este structurat în patru pachete: 'dataAccessLayer', 'helpers', 'model' și 'views', fiecare cuprinzând clase specifice unei anumite funcționalități.

4.3.1.1 Pachetul 'Model'

Pachetul 'model' cuprinde 4 clase, reprezentând principalele entități cu care lucrează aplicația: Player, Game, PlayerRating și Team.

Clasa 'Player' cuprinde atribute care reprezintă informațiile pentru fiecare jucător, precum și metode publice pentru interacțiunea cu entitatea.

Clasa Player cuprinde următoarele atribute:

- Int id, identificatorul unic al jucătorului
- String username, numele jucătorului
- String password, parola utilizatorului
- String picture, poza de profil în cazul în care utilizatorul a încărcat una, altfel se setează o poza de profil default

- Boolean available, atribut ce reține disponibilitatea jucătorului; un jucator available se poate înscrie la meciuri și poate fi inclus în formarea echipelor
- Double rating, conține rating-ul curent al jucătorului
- List<Games> games, va conține lista tuturor jocurilor unui user, care cuprinde atât jocurile următoare cât și jocurile care s-au desfășurat
- List<PlayerRating> playerRatings, păstrează istoricul ratingurilor pentru un utilizator

```

@NamedQuery(name="Player.findAll", query="SELECT p FROM Player p")
public class Player implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;

    private Boolean available;

    private String password;

    private String picture;

    private double rating;

    private Integer type;

    private String username;

    private Boolean archive;

    @ManyToMany(mappedBy="players", fetch=FetchType.EAGER)
    private List<Game> games;

    //bi-directional many-to-one association to PlayerRating
    @OneToMany(mappedBy="player", fetch=FetchType.EAGER)
    private List<PlayerRating> playerRatings;

    //bi-directional many-to-many association to Team
    @ManyToMany(mappedBy="players", fetch=FetchType.EAGER)

```

Figură 19. Clasa Player

Clasa Game este prezentată în Figura 18.

Atributele sale sunt:

- Integer id, identificatorul unic pentru joc
- Date date, reține data la care este programat jocul
- Integer difference, rezultatul jocului, reprezentând modulul dintre numărul de goluri înscrise de fiecare echipă
- Boolean archive, variabilă booleană ce va fi setată false în baza de date în cazul în care jocul este anulat

- Set<Player> players, o listă ce va conține jucătorii înscriși la joc

```

15 @Entity
16 @Table(name="game")
17 @NamedQuery(name="Game.findAll", query="SELECT g FROM Game g")
18 public class Game implements Serializable {
19     private static final long serialVersionUID = 1L;
20
21     @Id
22     @GeneratedValue(strategy=GenerationType.IDENTITY)
23     private Integer id;
24
25     @Temporal(TemporalType.DATE)
26     private Date date;
27
28     private Integer difference;
29
30     private Boolean archive;
31
32     //uni-directional many-to-many association to Player
33     @ManyToMany(fetch=FetchType.EAGER)
34     @JoinTable(name="game_player",
35         joinColumns={@JoinColumn(name="game_id")},
36         inverseJoinColumns={@JoinColumn(name="player_id")})
37     private Set<Player> players;
38 }

```

Figură 20. Clasa 'Game'

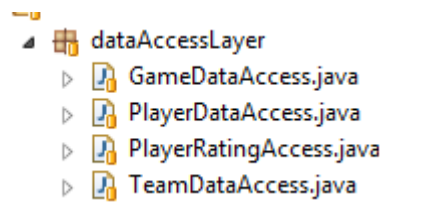
- Team firstTeam, prima echipă a meciului
- Team secondTeam, cea de-a doua echipă a meciului

Clasa 'Team' operează cu date referitoare la echipele fiecărui joc. Aceasta conține informații atât despre jucători, cât și despre modul în care sunt împărțiți în cele două echipe.

Clasa 'PlayerRating' este folosită pentru a păstra istoricul rating-urilor pentru fiecare jucător. Fiecare entitate de tip 'PlayerRating' conține următoarele informații: id-ul jucătorului, rating-ul său și data la care a fost setat acesta.

4.3.1.2 Pachetul 'DataAccessLayer'

Pachetul 'dataAccessLayer' cuprinde clase care ajută la salvarea modificărilor și persistarea entităților. Clasele conțin metode statice ce asigură nivelurile de acces la entități.



Figură 21. Pachetul 'DataAccessLayer'

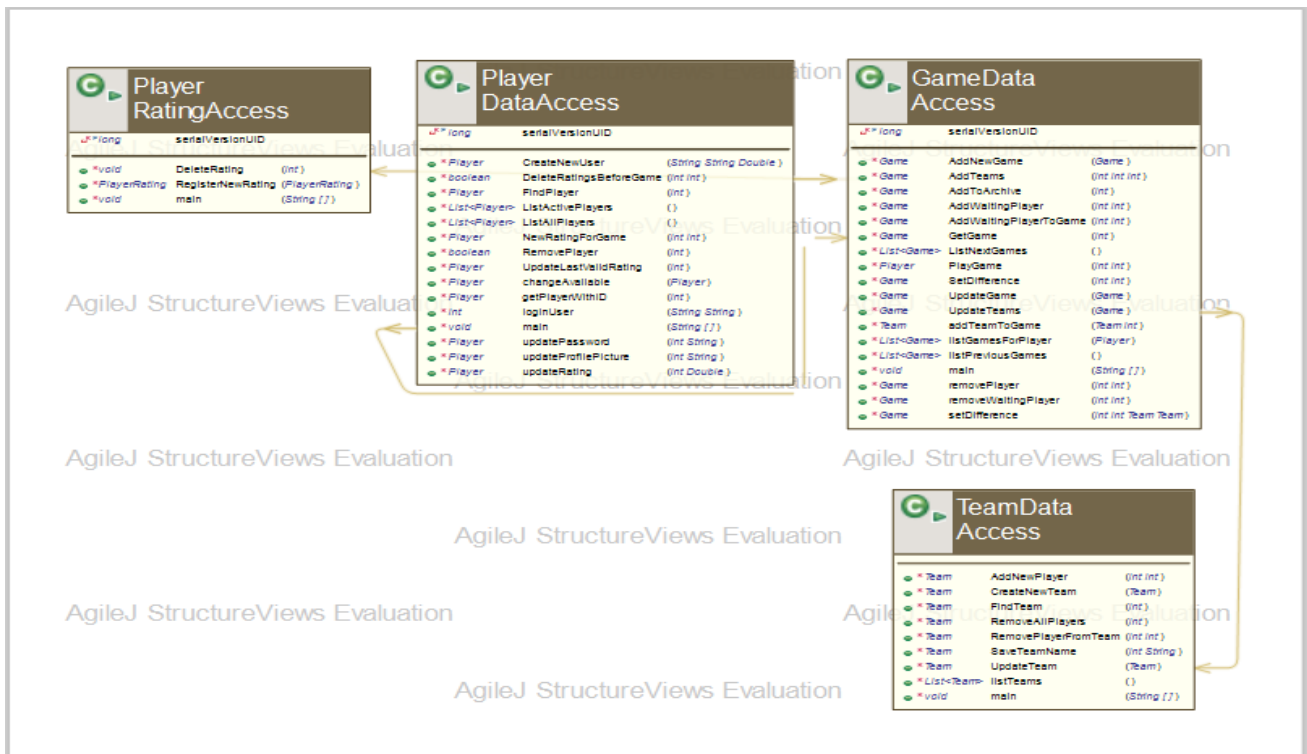
Clasa 'PlayerDataAccess' cuprinde următoarele metode:

```

14 @ManagedBean(name = "playerDataAccess")
15 @ApplicationScoped
16 public class PlayerDataAccess implements Serializable{
17     /**
18      *
19      */
20     private static final long serialVersionUID = 1L;
21
22     public static Player createUser(String username, String password) {
23
24     }
25
26     public static int loginUser(String username, String password) {
27
28     }
29
30     public static Player getPlayerWithID(int playerId) {
31
32     }
33
34     public static Player updatePassword(int playerId, String password) {
35
36     }
37
38     public static Player updateProfilePicture(int playerId, byte[] newPicture) {
39
40     }
41
42     public static Player changeAvailable(Player player) {
43
44     }
45
46     public static List<Player> ListAllPlayers() {
47
48     }
49
50     public static boolean removePlayer(int playerId) {
51
52     }
53 }

```

Figură 22. Clasa PlayerDataAccess



Figură 23. Diagrama claselor din pachetul 'dataAccessLayer'

Astfel, cu ajutorul acestei clase, se pot înregistra noi utilizatori în baza de date, se pot face diverse modificări asupra utilizatorilor existenți sau se pot lista utilizatorii curenți.

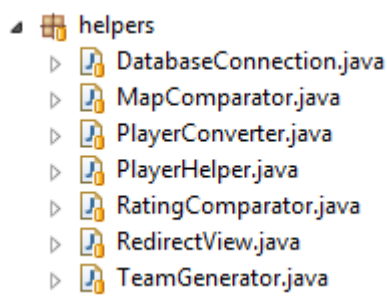
Clasa ‘GameDataAccess’ reprezintă un nivel de acces pentru clasa ‘Game’. Cu ajutorul acesteia se pot persista sau modifica entități de tip ‘Game’.

Funcționalitatea clasei ‘PlayerRatingAccess’ este aceea de a asigura actualizarea și salvarea ratingurilor pentru fiecare jucător participant la un meci de fotbal, după ce acesta a avut loc.

‘TeamDataAccess’ realizează salvarea configurației echipelor de fotbal, iar în cazul unor modificări, noile date sunt actualizate tot cu ajutorul acestei clase.

4.3.1.3 Pachetul ‘Helpers’

Pachetul ‘helpers’ cuprinde clase responsabile de realizarea legăturilor dintre clase sau generarea de informații ce vor fi utilizate ulterior pentru îndeplinirea altor funcții. Clasele au funcționalități bine delimitate, și reprezintă obiecte importante pentru funcționarea corectă a întregului program.



Figură 24. Pachetul ‘Helpers’

Clasa ‘DatabaseConnection’ este o clasă de tip Singleton, existând o singură instanță a acesteia pe parcursul execuției programului. Deoarece crearea unei conexiuni la baza de date este o operație destul de costisitoare din punct de vedere al resurselor și al performanței, crearea conexiunii se va face o singură dată, urmând să fie folosită de clase unde este necesară această conexiune. Obținerea unei conexiuni de către o clasă se face prin apelarea metodei `getConnection()`, ce returnează un obiect de tip `EntityManager`, necesar pentru realizarea managementului entităților. În general, clasele care folosesc un obiect de acest tip sunt clasele din pachetul ‘dataAccessLayer’.

Clasele `RatingComparator` și `MapComparator` implementează interfața `Comparable` și suprascriu metoda `Compare`. Sunt folosite în realizarea sortărilor unui grup de jucători după ratingul acestora, respectiv în compararea echipelor în scopul obținerii celei mai ‘echilibrate’ echipe.

```

1 package helpers;
2
3 import java.util.Comparator;
4
5
6
7 public class RatingComparator implements Comparator<Player> {
8
9     @Override
10    public int compare(Player p1, Player p2) {
11        if(p1.getRating()<p2.getRating())
12        {
13            return -1;
14        }
15        if(p1.getRating()>p2.getRating())
16        {
17            return 1;
18        }
19        return 0;
20    }
21
22 }

```

Figură 25. Clasa RatingComparator

PlayerConverter este clasa ce reprezintă un convertor custom pentru obiectul de tip Player. Pentru a crea o clasă de tip convertor al unui obiect, clasa trebuie să implementeze interfața Converter și să suprascrie metodele:

@Override

```
public Object getAsObject(FacesContext context, UIComponent component,String value) {    //...}
```

@Override

```
public String getAsString(FacesContext context, UIComponent component,Object value) {//...}
```

Pentru a atribui convertorului un ID, putem folosi adnotarea „@FacesConverter”. În cazul clasei PlayerConverter, am utilizat adnotarea pentru a-i atribui id-ul „playerConverter”. În cadrul aplicației, este utilizată atunci când se dorește mutarea unui jucător dintr-o echipă în alta, folosind interfața web. Echipele sunt afișate dinamic, sub formă a două tabele, utilizatorul având posibilitatea să facă schimbări de elemente din tabel, realizând astfel un schimb de jucători între echipe. Acest lucru este implementat cu ajutorul elementului PickList din framework-ul Primefaces, căruia i se plasează ca și atribut convertorul custom ‘PlayerConverter’.

```

1 package helpers;
2
3 import javax.faces.component.UIComponent;
4
5
6
7
8
9
10 @FacesConverter("playerConverter")
11 public class PlayerConverter implements Converter {
12
13
14     public Object getAsObject(FacesContext arg0, UIComponent arg1, String value) {
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43     public String getAsString(FacesContext arg0, UIComponent arg1, Object value) {
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
}

```

Figură 26. Clasa PlayerConverter

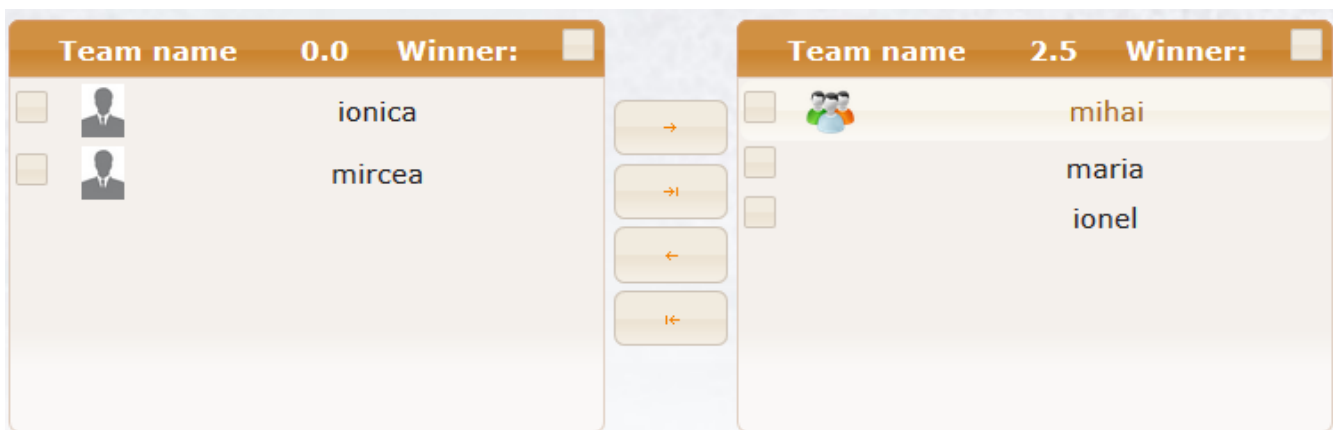
```

<p:pickList id="pick" value="#{teamsView.players}" var="player" effect="bounce" itemValue="#{player}"
itemLabel="#{player.username}" showCheckbox="true" responsive="true"
filterMatchMode="contains" converter="playerConverter" style="width:100%">

```

Figură 27. Sintaxa elementului Picklist cu atributul ‘converter’

În pagina web accesată de client, cele două echipe sunt reprezentate precum în Figura25.

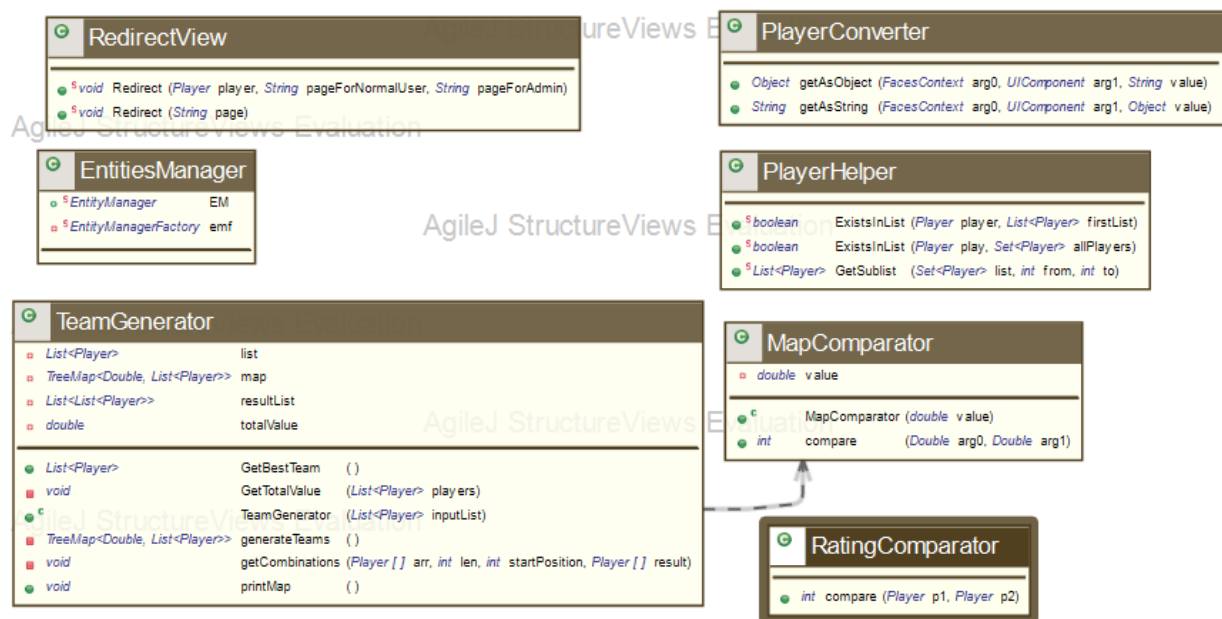


Figură 28. Reprezentarea echipelor cu ajutorul elementului PickList

Clasa ‘RedirectView’ este folosită în cadrul claselor din pachetul Views, pentru redirectarea către diferite pagini, în funcție de tipul de utilizator. Aceasta conține două metode statice:

- void Redirect (Player player, String pageForNormalUser, String pageForAdmin), ce realizează redirectarea către pagini custom pentru fiecare tip de utilizator
- void Redirect (String page), redirecționează toate categoriile de utilizatori către aceeași pagină web

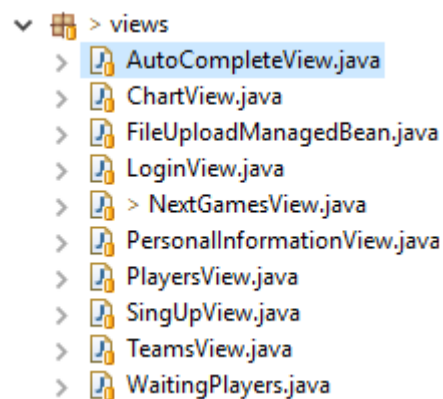
Un exemplu de utilizare a clasei este atunci când se face autentificarea unui utilizator. După validarea datelor, utilizatorul este redirecționat către o pagină, ce va cuprinde conținut accesibil și util pentru acesta.



Figură 29. Diagrama claselor pachetului 'helpers'

4.3.1.4 Pachetul 'Views'

În cadrul acestui pachet sunt cuprise clasele de tip 'ManagedBean', cele care sunt responsabile de logica aplicației. Ele reprezintă practic obiecte pe partea de server, asociate unor componente UI folosite în paginile web. Clasele regăsite în pachet sunt cele din Figura26.



Figură 30. Pachetul 'views'

Fiecare clasă este asociată cu componente UI din paginile web, iar la interacțiunea utilizatorului cu aplicația, acestea sunt responsabile de preluarea și prelucrearea, dacă este cazul, a datelor cu care lucrează utilizatorul.

De exemplu, pentru a adăuga noi jucători în lista de jucători deja existenți ai unui joc, se folosește elementul Primefaces AutoComplete, căruia i se asociază o listă de jucători din managed bean-ul 'AutoCompleteView'. Asocierea se face prin setarea atributului value din autoComplete:

```
<p:autoComplete          multiple="true"          value="#{autoCompleteView.selectedPlayers}"
converter="playerConverter" forceSelection="true">
    <p:column style="width:80%">
        <h:outputText value="#{theme.username}"/>
    </p:column>
</p:autoComplete>
```

unde ,selectedPlayers' este o listă din clasa 'AutoCompleteView'.

```
@ManagedBean
public class AutoCompleteView {

    public List<Player> selectedPlayers;

    public List<Player> getSelectedPlayers() {
        return selectedPlayers;
    }

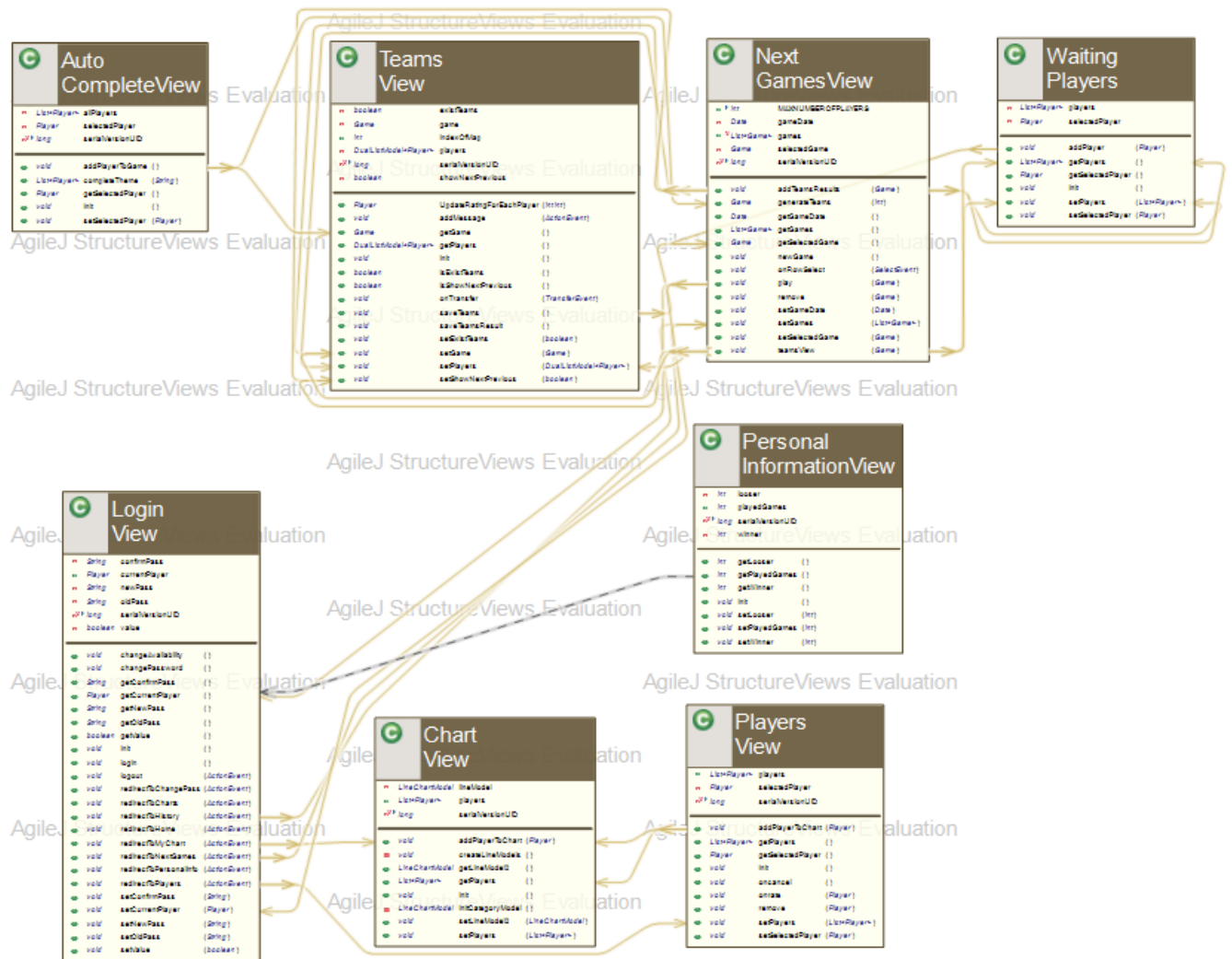
    public void setSelectedPlayers(List<Player> selectedPlayers) {
        this.selectedPlayers = selectedPlayers;
    }
}
```

Figură 31. ManagedBean-ul AutoCompleteView

În cadrul paginii web, utilizatorul va interacționa cu fereastra:

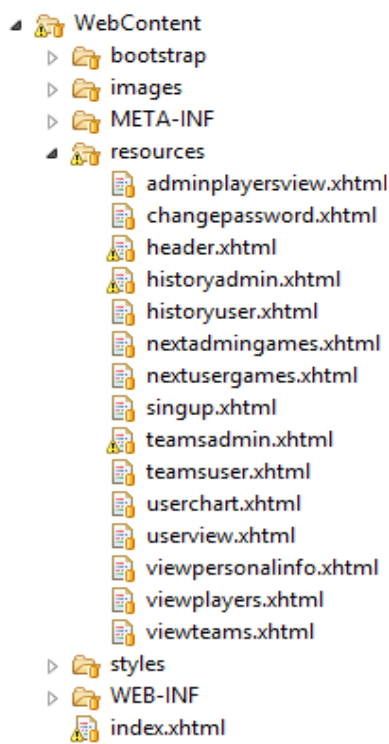


Figură 32. Fereastra pentru adăugarea unui nou jucător



Figură 33. Diagrama claselor pachetului 'views'

Partea de client este reprezentată de paginile web, interfața web cu care interacționează clientul. Paginile web sunt cuprinse în folderul 'WebContent->resources'. Conținutul din 'WebContent' este ilustrat în Figura 29. Paginile au extensia xhtml, fiecare având un scop bine stabilit în interacțiunea cu utilizatorul. Folderul mai conține în plus alte foldere ce cuprind librării sau resurse necesare pentru buna funcționare a aplicației.



Figură 34. Conținutul folderului 'WebContent'

În folderul 'bootstrap' se află librăria Bootstrap, ce va fi inclusă ulterior în paginile web pentru a putea fi folosită. Librăria conține jar-uri, pagini .css dar și pagini .js.

Folderul 'images' conține pozele de profil ale jucătorilor, încărcate pe server. De fiecare dată când se face update pozei de profil, aceasta este salvată în acest folder, existând o singură fotografie pentru fiecare user.

În folderul 'styles' sunt cuprinse câteva pagini .css, care sunt folosite pentru a suprascrie stilurile implicite fie din Bootstrap sau cele din Primefaces.

4.4 Considerații relativ la implementare

“Aplicația suport pentru constituirea de echipe echilibrate și personalizabile” este o aplicație web. Pe parte de server este folosit Tomcat, codul fiind scris în limbajul de programare Java, folosind mediul de dezvoltare Eclipse. Partea de Client, care reprezintă interacțiunea cu utilizatorul, este dezvoltată folosind limbajul HTML, împreună cu framework-urile Bootstrap și Primefaces.

4.4.1 Suprascrierea stilului CSS implicit din Bootstrap

Pentru a se face suprascrierea unui stil CSS din Bootstrap sau pentru a adăuga noi elemente de stil se creează o nouă pagină cu extensia .css. Apoi se adaugă referința către pagina creată în elementul `<head></head>` al paginii:

```
<h:head>  
    <link rel="stylesheet" type="text/css" href="../styles/custCss.css"/>  
</h:head>
```

Pentru a aplica paginii stilul custom dorit, este de ajuns să adăugăm descrierea acestuia în pagina .css creată anterior. De exemplu, dacă dorim să schimbăm stilul elementului `div` ce are clasa Bootstrap ‘`navbar navbar-default`’ din secvența de cod următoare:

```
<div class="navbar navbar-default" style="margin-bottom:0px">  
<div class="container-fluid">  
<div class="row"></div>  
</div>  
</div>
```

adăugam în pagina .css descrierea stilului:

```
. navbar navbar-default  
{  
    padding-top: 10px;  
    float: none;  
    text-align: center;  
    font-size: 18px;  
}
```

Putem defini propriile clase de stil ce pot fi aplicate elementelor. În “Aplicația suport pentru constituirea de echipe echilibrate și personalizabile” am definit clasa ‘`withBackground`’, care setează un fundal unui element html dacă este aplicată acestuia.

```
.withBackground  
{  
background: url('../images/d.jpg') no-repeat center center fixed;  
-webkit-background-size: cover;  
-moz-background-size: cover;
```



```

-o-background-size: cover;
background-size: cover;
}

```

Pentru a adăuga o clasă de stil unui element, se adaugă numele clasei custom la numele clasei din Bootstrap:

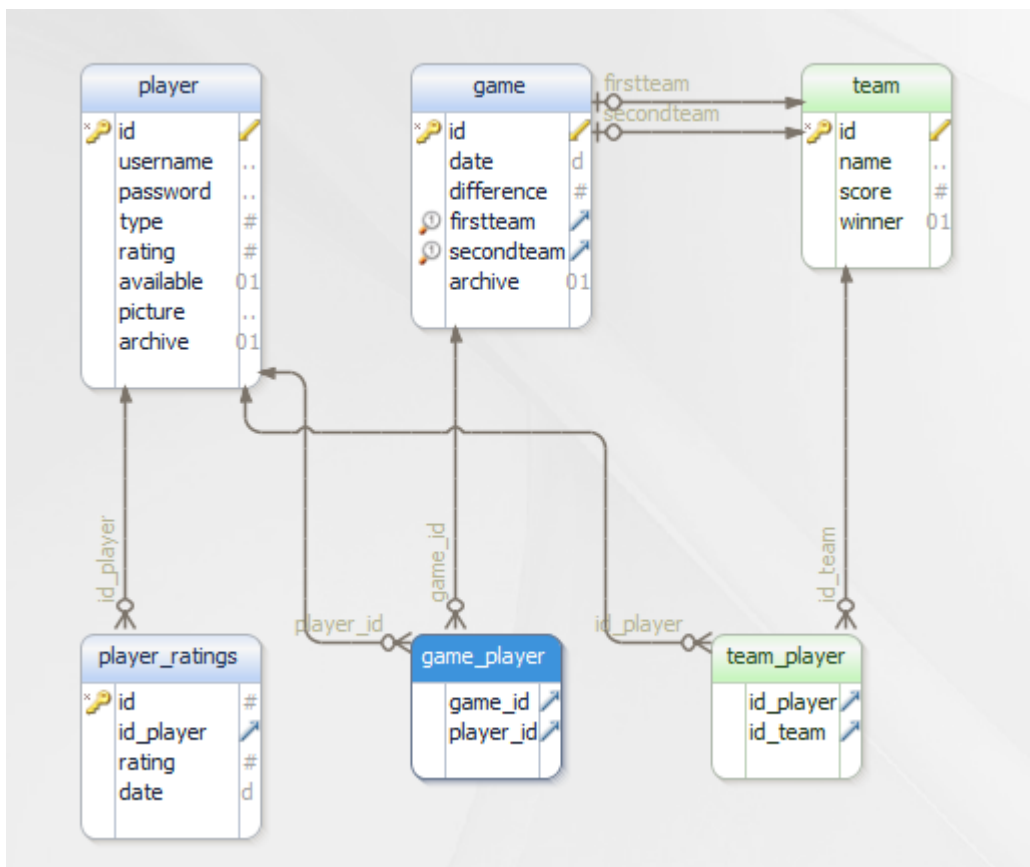
```

<div class="container-fluid withBackground" ></div>

```

4.4.2 Persistența datelor

Datele fiecărui utilizator sunt salvate într-o bază de date, realizată cu ajutorul sistemului de baze de date relațional PostgreSQL și administrată cu ajutorul tool-ului pgAdmin. Aceasta este formată din 6 tabele, ce reprezintă entitățile principale cu care operează aplicația.



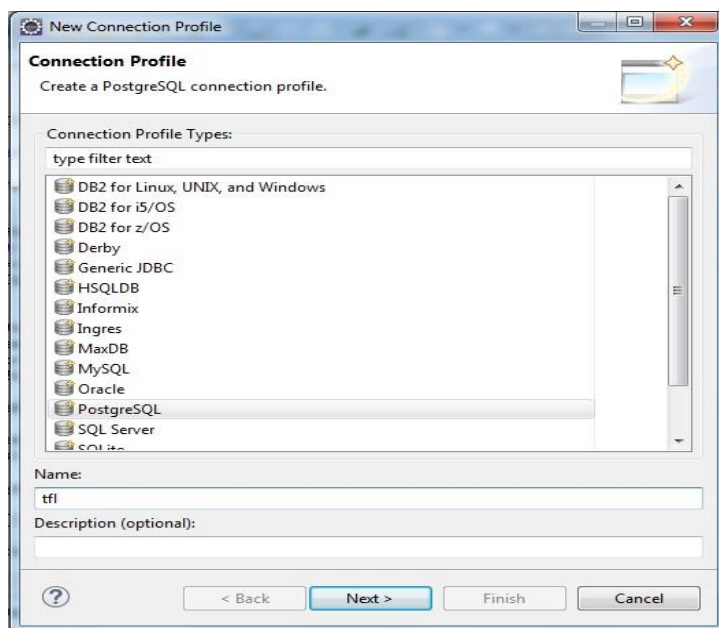
Figură 35. Baza de date a aplicației

Tabelele sunt mapate cu ajutorul framework-ului JPA, devenind clase cu care se va realiza logica aplicației. Clasele ce reprezintă entitățile din baza de date sunt integrate în pachetul ‘model’.

4.4.2.1 Maparea entităților cu ajutorul JPA

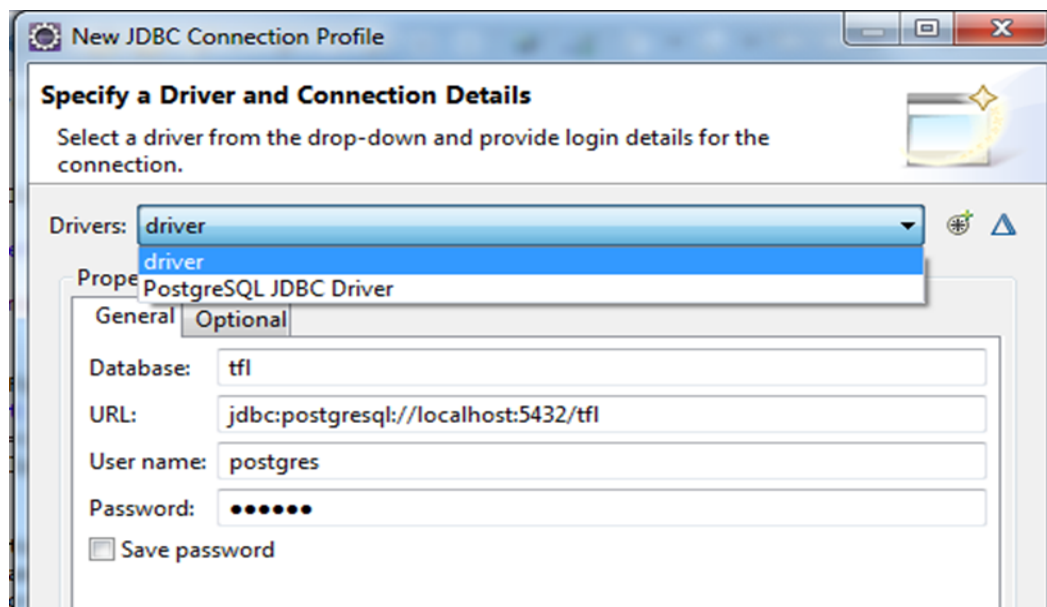
Pentru a se obține clase pornind de la o bază de date existentă cu ajutorul JPA, în primul rând trebuie să creem o conexiune la baza de date. Folosind Eclipse, se urmează pașii:

1. Din tab-ul Eclipse 'Data Source Explorer' se alege opțiunea Database Connections-> New...



Figură 36. Crearea conexiunii la baza de date PostgreSQL

2. Se alege driver-ul PostgreSQL pentru a se realiza conexiunea



Figură 37. Alegerea driver-ului pentru conexiune

3. Se verifică existența JPA în cadrul proiectului. Acest lucru se poate face accesând: Click dreapta pe proiect->Proprietates->JPA

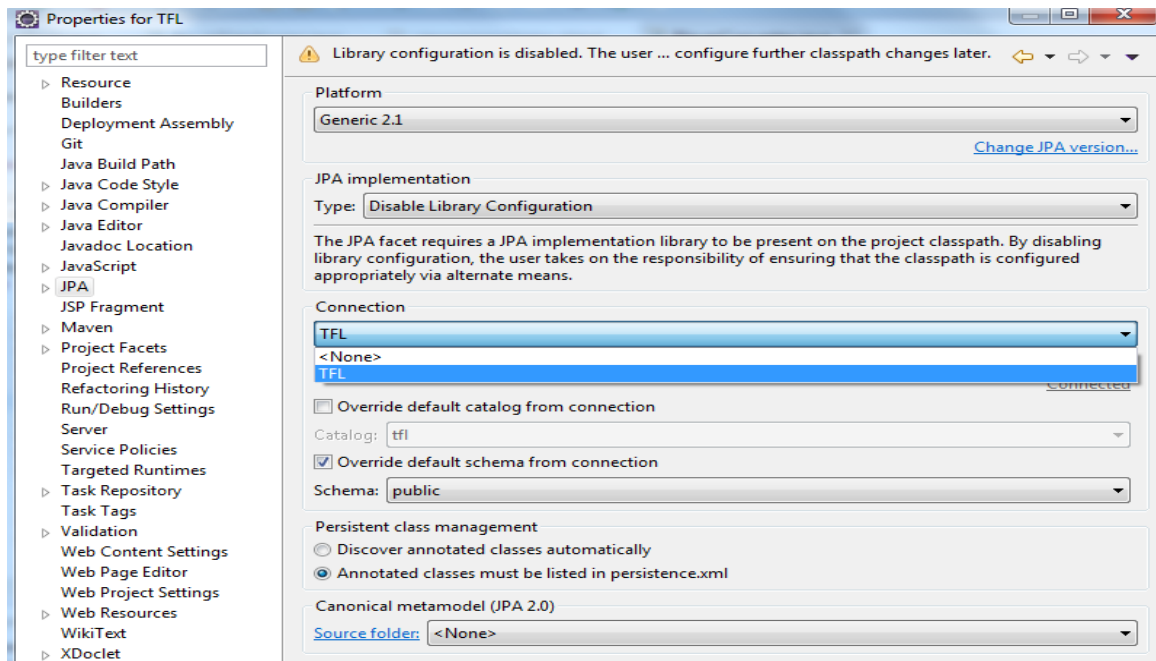
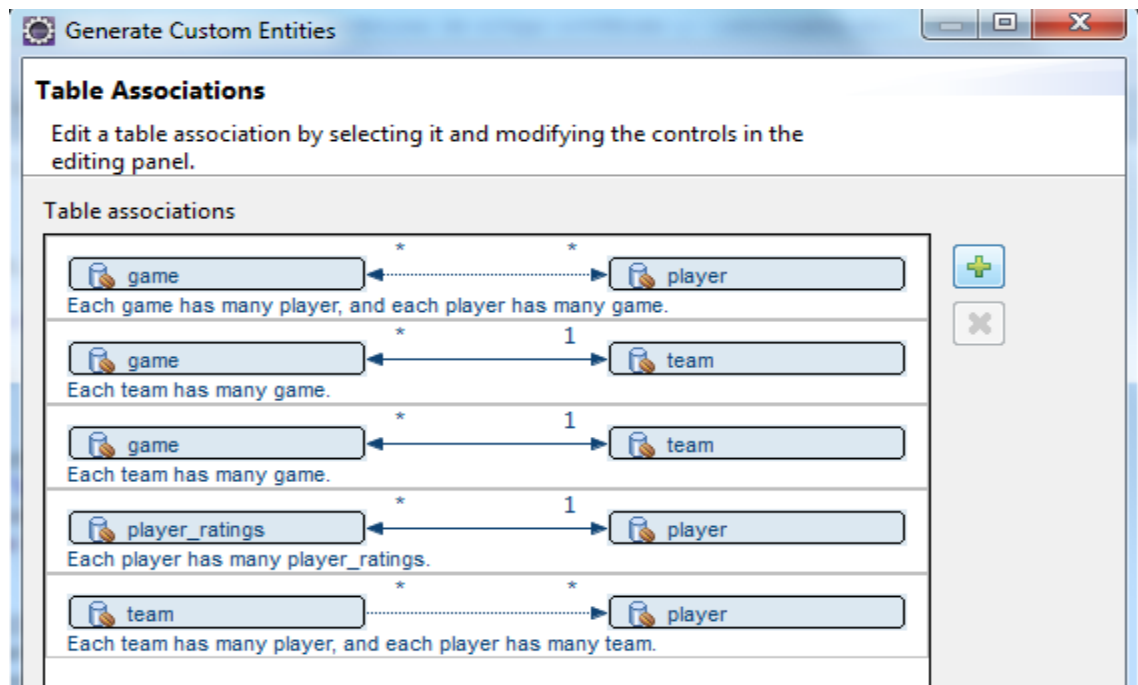


Figura 38. Verificarea existenței JPA în cadrul proiectului

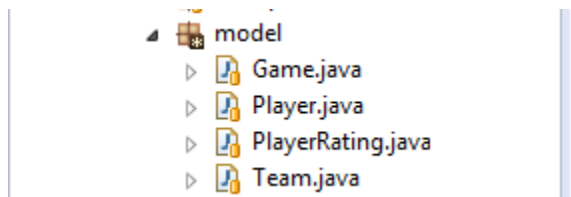
4. Se generează entitățile prin accesarea: Click dreapta pe proiect-> New-> JPA Entities from tables, apoi din fereastra apărută se aleg entitățile pentru care se dorește generarea claselor.



Figură 39. Maparea entitatilor din baza de date

În această etapă se pot adăuga noi asocieri între tabele sau se pot customiza cele existente.

În final, pachetul ‘model’ este format din 4 clase: Game, Player, PlayerRating și Team, rămânând două tabele de legătura pentru care nu a fost necesară crearea unei entități.



Figură 40. Clasele rezultate în urma mapării JPA a entităților

4.4.3 Algoritmul de generare de echipe ‘echilibrate’

Algoritmul de generare al echipelor este cuprins în clasa ‘TeamGenerator’. Acesta reprezintă o formă mai complexă a algoritmului de generare a tuturor combinațiilor din n elemente luate câte k . Formula generală a combinațiilor este:

$$C_n^p = \frac{A_n^p}{P_p} = \frac{n!}{p!(n-p)!}, \text{ cu } 0 \leq p \leq n.$$

Figură 41. Formula generala combinari

Algoritmul implementat se folosește de această formulă, generând în primă instanță toate combinațiile din n elemente luate câte $n/2$, pentru ca echipele să aibă și același număr de jucători.

Principalele atribute ale clasei ‘TeamGenerator’ sunt:

- List<Player> list, reprezintă lista de jucători ce se dorește a fi împărțită în două echipe
- Double totalValue, însumează rating-ul tuturor jucătorilor împărțit la 2 și va fi necesar în algoritm pentru compararea echipelor
- TreeMap<Double, List<Player>> resultMap, va conține echipele sortate în ordine crescătoare, în funcție de gradul de ‘echilibru’ obținut

O echipă se consideră cu atât mai echilibrată cu cât modulul diferenței dintre totalValue și suma rating-urilor jucătorilor din echipă este mai mic. Considerăm:

$$diff = totalValue - \sum_{1}^n ratingjucator$$

calculată pentru fiecare echipă. În TreeMap se va reține fiecare grup obținut împreună cu valoarea sa *diff*. Deoarece TreeMap primește pe constructor comparatorul de echipe ‘MapComparator’, în final obținem toate grupurile posibile formate din lista inițială de jucători, sortate crescător.

De exemplu, în Figura38, avem lista de jucători și rating-ul fiecăruia, listă ce se dorește a fi împărțită în două echipe.

```
mihai 1.8  
mircea 3.0599999999999996  
ion 2.52  
marius 5.5
```

Figură 42 Lista jucatori pentru împărțirea în echipe

După aplicarea algoritmului, se obține rezultatul:

```
Valoare ideala a unei echipe: 6.4399999999999995  
Echipele sortate crescator in functie de 'echilibrul' obtinut sunt:  
Echipa: mircea ion Diff : 0.8599999999999994 Suma rating-urilor : 5.58  
Echipa: mihai marius Diff : 0.8600000000000003 Suma rating-urilor : 7.3  
Echipa: ion marius Diff : 1.58 Suma rating-urilor : 8.02  
Echipa: mircea marius Diff : 2.1199999999999999 Suma rating-urilor : 4.32
```

Figură 43. Rezultatul împărțirii în echipe e listei

Se observă că suma rating-urilor pentru grupul format din jucătorii ‘Mircea’ și ‘Ion’, 5.58, este cea mai apropiată valoare de valoarea ideală a unei echipe în cazul acestui grup, având diff=0.859. Evident, al doilea grup va fi format din jucătorii rămași, ‘Mihai’ și ‘Marius’, cu diff=0.86.

4.4.4 Actualizarea modificărilor

După fiecare meci jucat, apar modificări ce trebuie înregistrate, acest lucru necesitând și o actualizare a datelor ce depind de fiecare schimbare apărută. Una dintre schimbările ce poate apărea este rearanjarea echipelor în cadrul terenului. Chiar daca echilibrul dintre echipe este influențat într-un mod negativ, aplicația se dorește a fi una flexibilă, ca urmare pot apărea schimbări de ultim moment, deși echipele au fost salvate în baza de date.

În cazul acesta, un utilizator ce deține drepturi poate accesa prin intermediul aplicației datele despre meci, și poate actualiza configurația celor două echipe, recalculându-se rating-ul total al echipei. De asemenea, utilizatorii au dreptul să își aleagă un nume personalizat pentru fiecare echipă, numele putând fi schimbat de mai multe ori în cadrul unui joc, însă numai de către utilizatori autorizați.

În plus, un user admin poate adăuga noi jucători, în cazul în care nu este depășit numărul maxim de jucători înscriși. În această situație, se păstrează echipele existente, jucătorii fiind schimbați după aplicarea algoritmului de generare de echipe.

Orice utilizator are opțiunea de a se înscrie pentru a participa la un meci de fotbal, sau poate opta să nu mai participe, prin intermediul meniului pus la dispoziție de pagina web. În cazul înscrierii unui jucător algoritmul urmează următorii pași:

1. Se verifică dacă utilizatorul este disponibil pentru înscriere
2. Dacă este disponibil, se verifică dacă numărul maxim de jucători ce se pot înscrie nu a fost depășit
3. Dacă se mai pot înscrie jucători, user-ul se adaugă în lista de jucători ai jocului curent și se reconfigurează echipele
4. În cazul în care s-a depășit numărul maxim de locuri, utilizatorul este adăugat în lista de așteptare a meciului, pentru cazurile în care alți jucători renunță.

Next games		
Date	Difference	
2016-06-28	0	Play
2016-06-30	0	Call off
2016-07-01	0	Play
2016-07-02	0	Call off
2016-06-30	0	Play

Figură 44.Înscrierea unui jucător în cadrul unui meci

În situația în care un jucător nu mai poate participa, are disponibilă opțiunea ‘Call off’. El este eliminat din lista de jucători ai meciului, și este adăugat primul jucator din lista de așteptare, în cazul în care există. Apoi se face o reconfigurarea a echipelor, pentru a restabili echilibrul.

4.4.5 Actualizarea rating-urilor

După fiecare meci jucat, utilizatorii au opțiunea de a introduce rezultatul, sub formă de număr, ce reprezintă numărul de goluri pentru fiecare echipă. În funcție de diferența dintre numărul de goluri, rating-urile jucătorilor sunt actualizate, putându-se astfel urmări evoluția sau involuția utilizatorilor.

Pentru a se realiza actualizarea rating-urilor algoritmul urmează următorii pași:

1. Se calculează modulul diferenței dintre numărul de goluri dintre prima echipă și a doua. Fie diff diferența dintre numărul de goluri dintre cele doua echipe

$$diff = |numărGoluriEchipa1 - numărGoluriEchipa2|$$

2. Se desemnează echipa câștigătoare (echipa cu cele mai multe goluri)

3. Pentru fiecare jucător se recalculează ratingul. Dacă acesta face parte din echipa câștigătoare, rating-ul său va crește după formula:

$$\text{Noul rating} = \text{ratingVechi} + 0.01 * \text{diff}$$

Altfel, noul rating va fi:

$$\text{Noul rating} = \text{ratingVechi} - 0.01 * \text{diff}$$

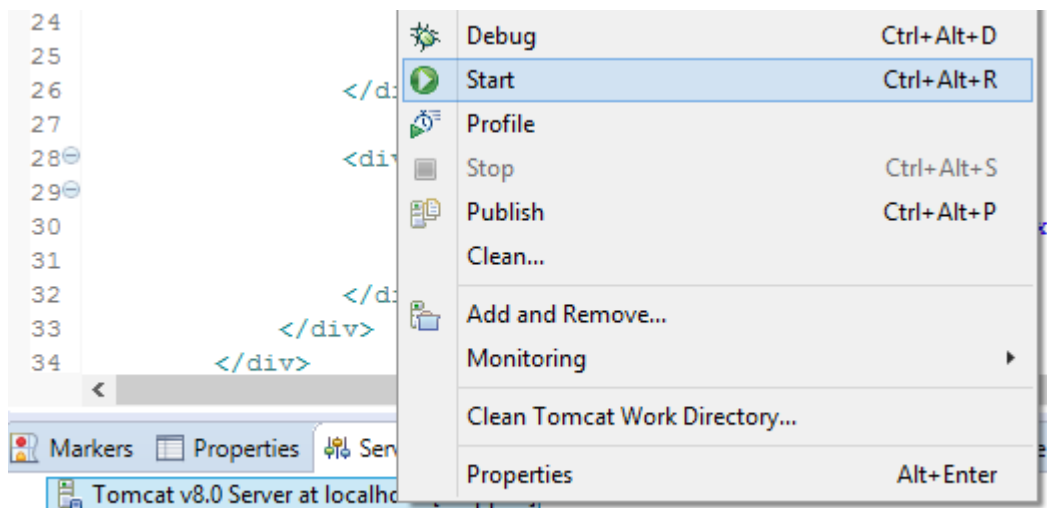
O problemă apare în cazul în care se dorește actualizarea numărului de goluri ale unui meci, posibil înregistrat greșit anterior. Deoarece rating-ul curent al fiecărui utilizator a fost actualizat în urma consemnării eronate a rezultatului precedent, o nouă înregistrare a rezultatelor ar putea duce la un update necorespunzător al fiecărui jucător. În cazul acesta se vor șterge toate datele începând cu data meciului pentru care se face actualizarea și se vor recalcula din nou pentru fiecare meci, păstrându-se astfel continuitatea și acuratețea înregistrărilor.

De exemplu, dacă rating-ul unui jucător care se înscrie la un meci este 5.0, iar după terminarea jocului echipa sa a înscris 4 goluri, câștigând cu diferența de 3, noul rating va fi 5.01.

4.5 Ghid de utilizare

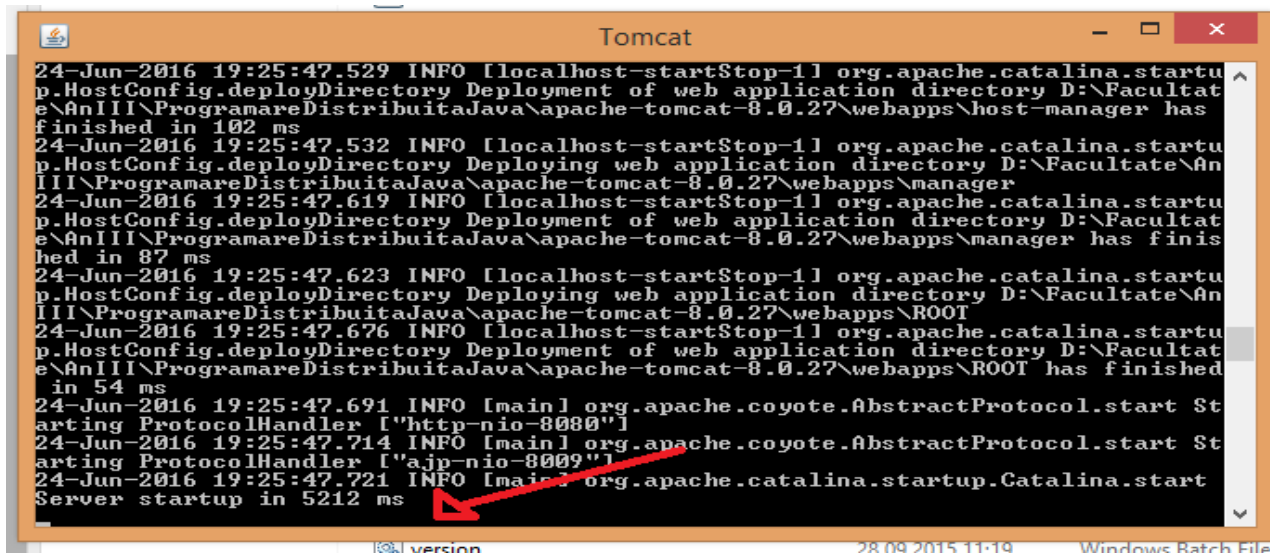
Pentru a putea utiliza aplicația, trebuie să pornim serverul, care va prelua datele de la client.

Serverul Tomcat se poate integra în mediul de dezvoltare Eclipse și se poate porni cu ajutorul acestuia, prin Click dreapta pe server->Start.



Figură 45. Pornirea serverului Tomcat din mediul de dezvoltare Eclipse

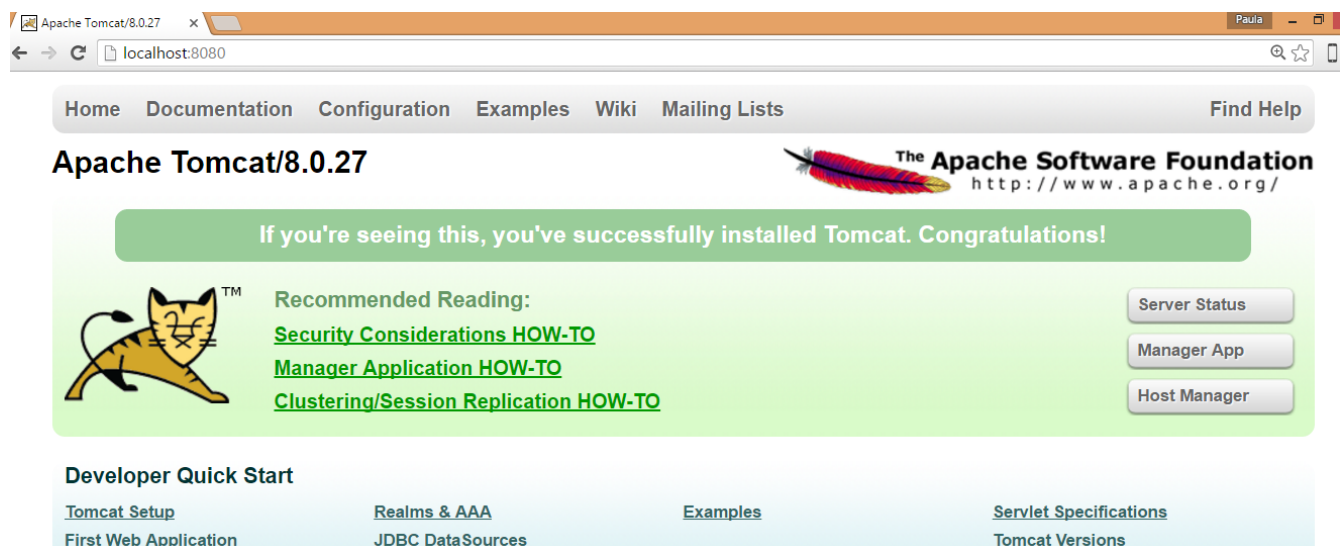
Dacă nu este integrat în mediul Eclipse, se poate porni prin fișierul startup.bat, accesat fie din linia de comandă, fie din folderul unde este localizat, prin dublu click.



```
24-Jun-2016 19:25:47.529 INFO [localhost-startStop-1] org.apache.catalina.startup
p.HostConfig.deployDirectory Deployment of web application directory D:\Facultate\AnIII\ProgramareDistribuitaJava\apache-tomcat-8.0.27\webapps\host-manager has
finished in 102 ms
24-Jun-2016 19:25:47.532 INFO [localhost-startStop-1] org.apache.catalina.startup
p.HostConfig.deployDirectory Deploying web application directory D:\Facultate\AnIII\ProgramareDistribuitaJava\apache-tomcat-8.0.27\webapps\manager
24-Jun-2016 19:25:47.619 INFO [localhost-startStop-1] org.apache.catalina.startup
p.HostConfig.deployDirectory Deployment of web application directory D:\Facultate\AnIII\ProgramareDistribuitaJava\apache-tomcat-8.0.27\webapps\manager has finis
hed in 87 ms
24-Jun-2016 19:25:47.623 INFO [localhost-startStop-1] org.apache.catalina.startup
p.HostConfig.deployDirectory Deploying web application directory D:\Facultate\AnIII\ProgramareDistribuitaJava\apache-tomcat-8.0.27\webapps\ROOT
24-Jun-2016 19:25:47.676 INFO [localhost-startStop-1] org.apache.catalina.startup
p.HostConfig.deployDirectory Deployment of web application directory D:\Facultate\AnIII\ProgramareDistribuitaJava\apache-tomcat-8.0.27\webapps\ROOT has finished
in 54 ms
24-Jun-2016 19:25:47.691 INFO [main] org.apache.coyote.AbstractProtocol.start St
arting ProtocolHandler ["http-nio-8080"]
24-Jun-2016 19:25:47.714 INFO [main] org.apache.coyote.AbstractProtocol.start St
arting ProtocolHandler ["ajp-nio-8009"]
24-Jun-2016 19:25:47.721 INFO [main] org.apache.catalina.startup.Catalina.start
Server startup in 5212 ms
```

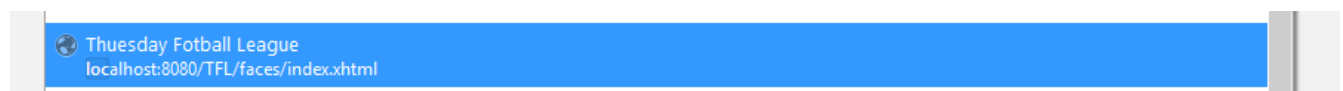
Figură 46. Pornirea serverului Tomcat din linia de comandă

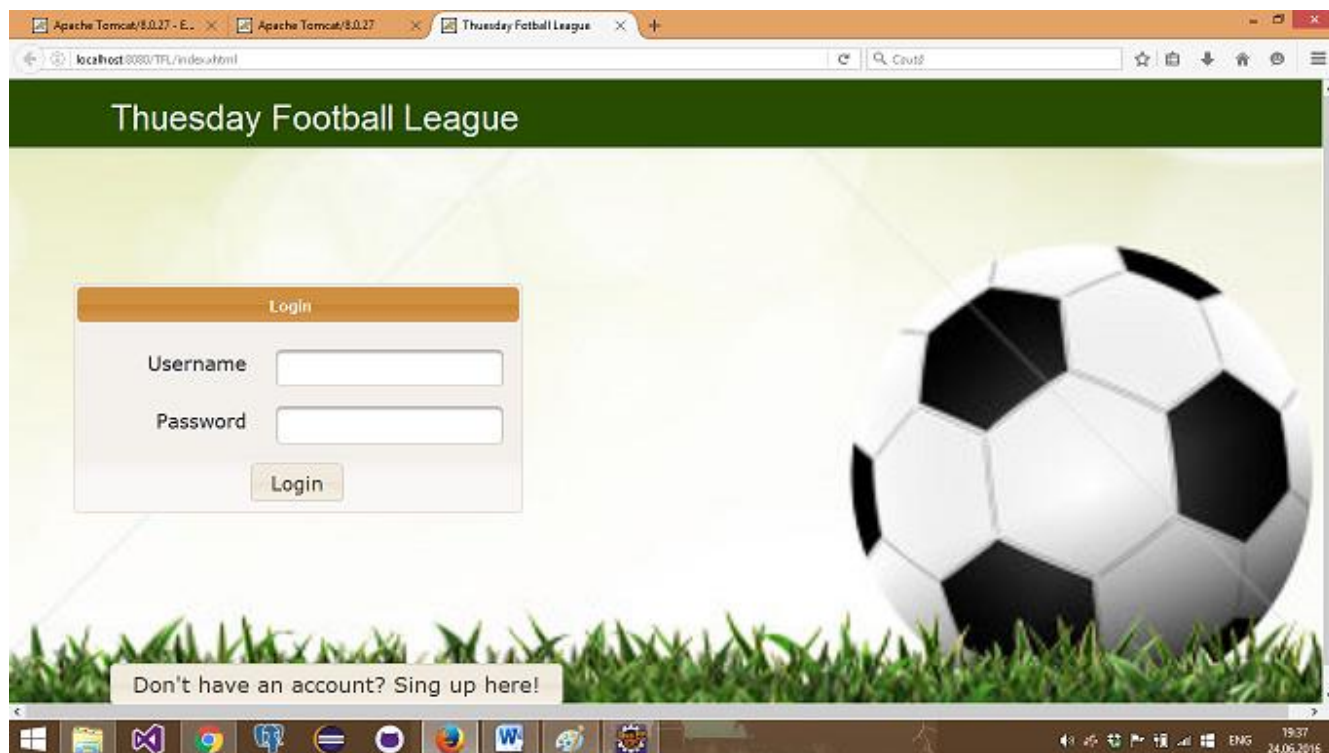
O altă modalitate de a ne asigura că serverul este pornit este aceea de a accesa într-un browser web pagina: localhost:8080, 8080 fiind portul pe care Tomcat rulează implicit.



Figură 47. Verificarea pornirii corecte a serverului Tomcat

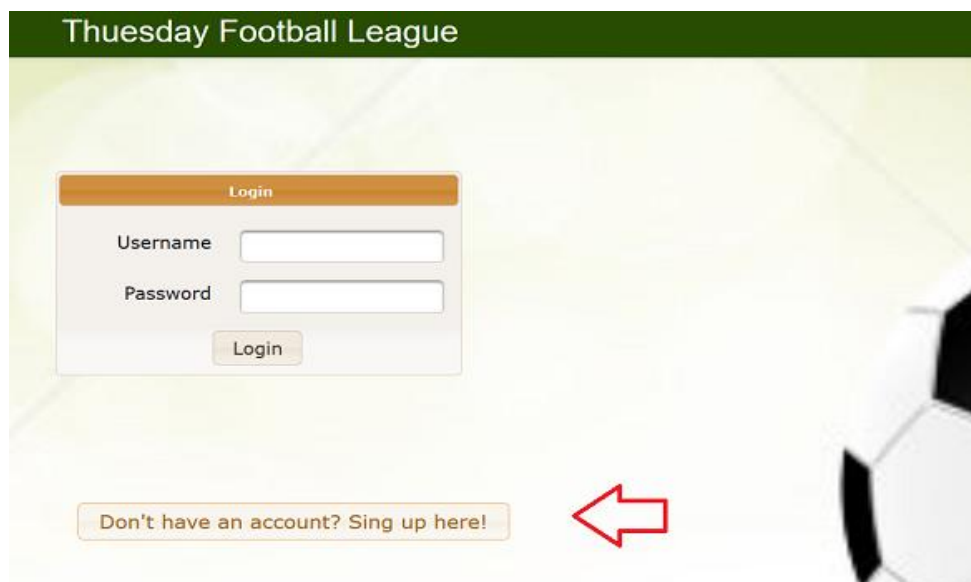
Accesul în aplicație se face tot prin intermediul browser-ului.





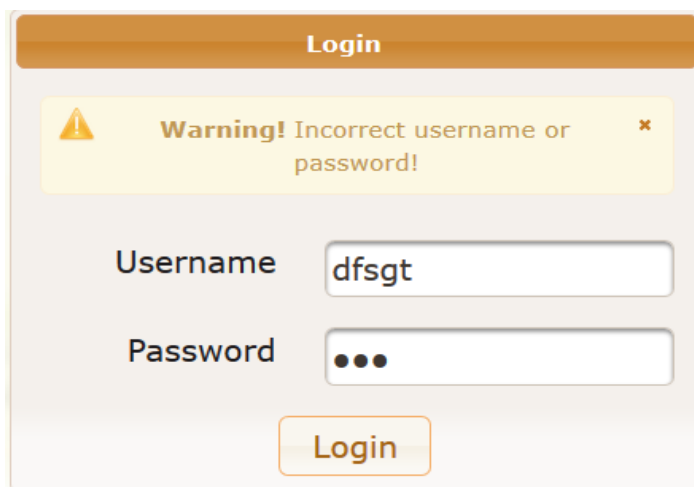
Figură 48. Accesul la aplicatie prin intermediul unui browser web

Următorul pas este completarea datelor necesare pentru autentificare: numele de utilizator și parola, în cazul în care există un cont creat, altfel se poate crea un cont nou, prin accesarea butonului din partea de jos a paginii.



Figură 49. Înregistrarea unui nou jucător

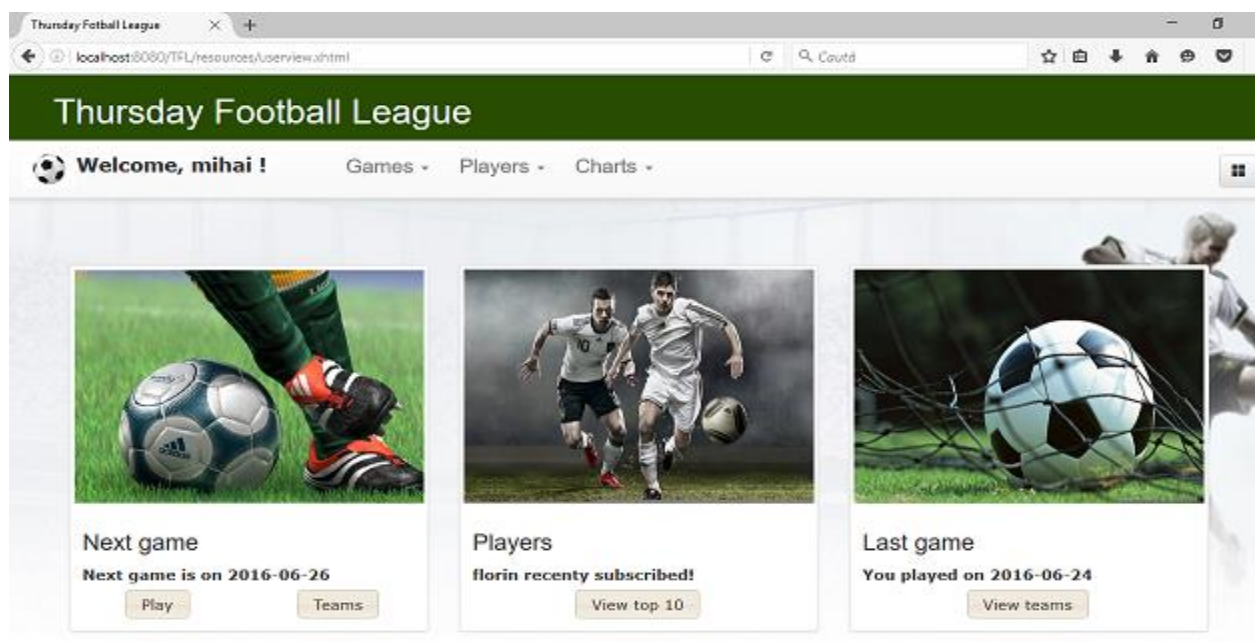
După completarea datelor, se verifică dacă acestea sunt corecte, în caz contrar se afișează un mesaj de atenționare:



The image shows a login form with a title bar labeled "Login". Below the title bar is a yellow warning box with a warning icon and the text "Warning! Incorrect username or password!". Below the warning box are two input fields: "Username" with the value "dfsgt" and "Password" with three dots indicating a masked password. At the bottom of the form is a "Login" button.

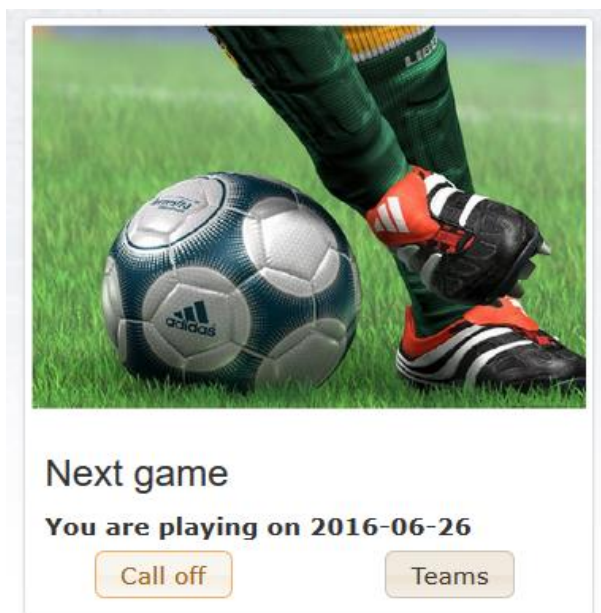
Figură 50. Exemplu autentificare cu date greșite

În cazul în care datele sunt corecte, utilizatorul este redirectat către o altă pagină, unde va putea interacționa cu aplicația prin intermediul unui meniu.



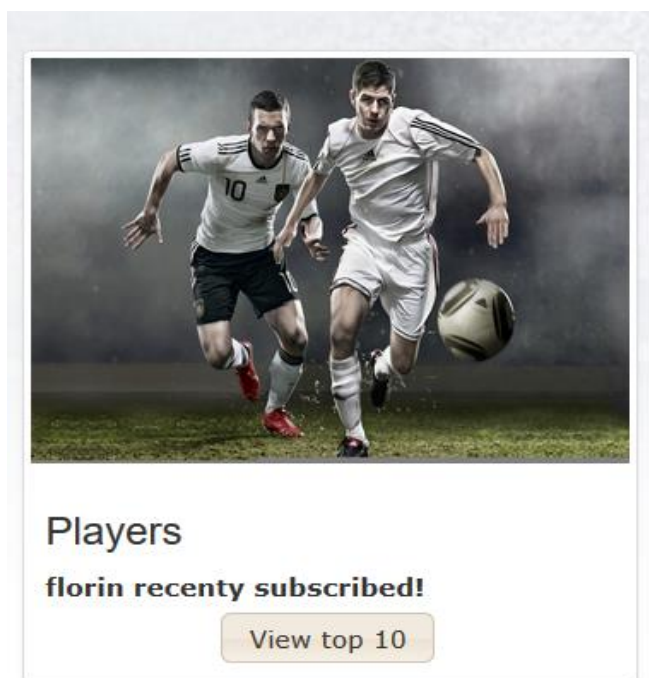
Figură 51. Pagina către care este redirectat un utilizator după autentificare

În pagina principală acesta are informații despre următorul meci, la care se poate înscrie sau poate vedea echipele formate până în momentul curent. În cazul în care acesta este înscris la un meci următor, sunt afișată informații despre acel meci, iar user-ul are opțiunea de anulare a meciului.



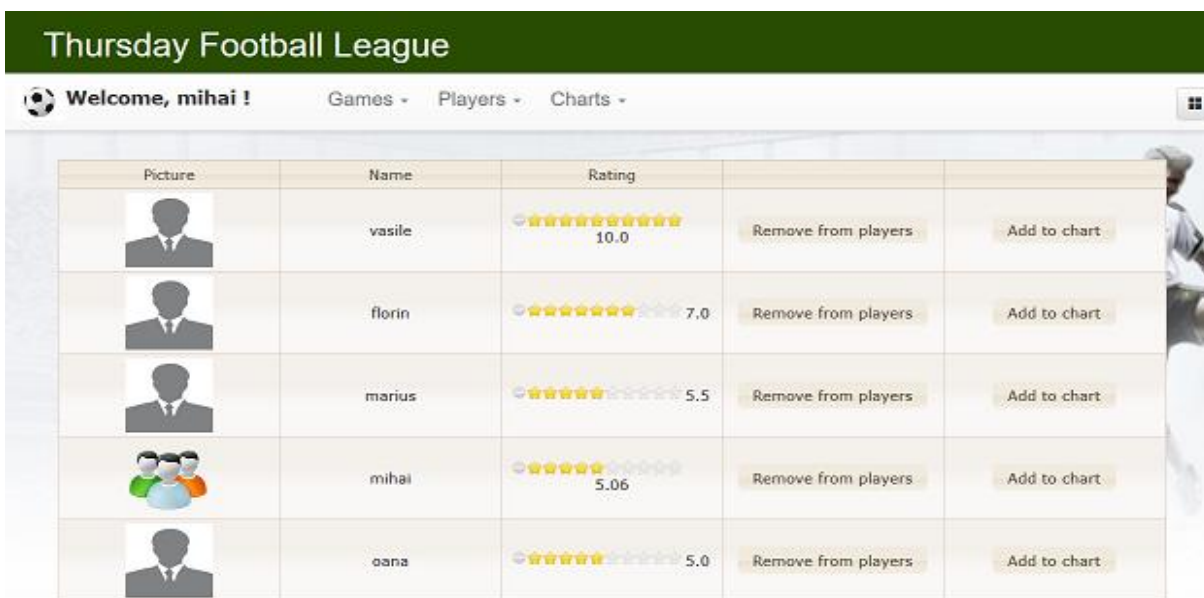
Figură 52. Prezentarea informațiilor despre meciul următor











Pagina principală conține de asemenea o secțiune în care se găsesc informații despre jucători. Este afișat ultimul user înregistrat în ultima săptămână, dacă există. Se poate accesa pagina ce va conține top 10 jucători, ordonați după rating.



Figură 53. Prezentarea informațiilor despre ultimii jucători înscriși

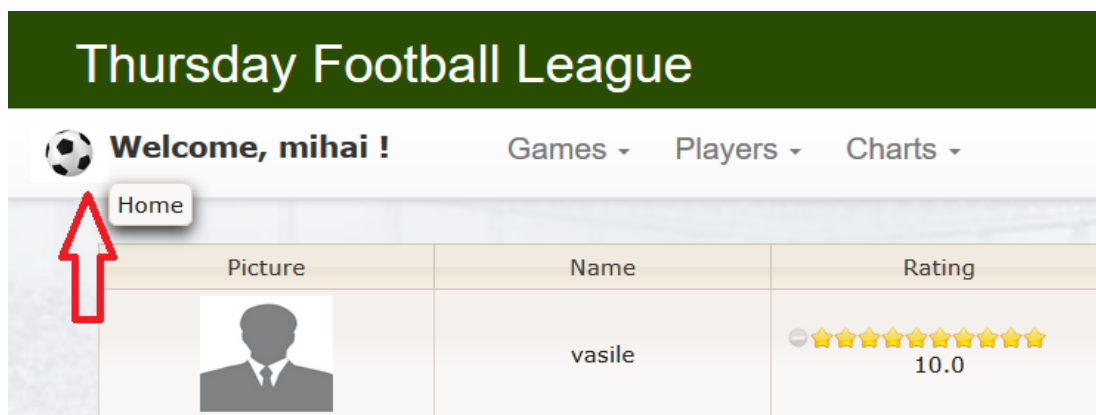
La click pe butonul 'View top 10', utilizatorul este redirectat către vizualizarea top-ului.



Picture	Name	Rating		
	vasile	 10.0	Remove from players	Add to chart
	florin	 7.0	Remove from players	Add to chart
	marius	 5.5	Remove from players	Add to chart
	mihai	 5.06	Remove from players	Add to chart
	oana	 5.0	Remove from players	Add to chart

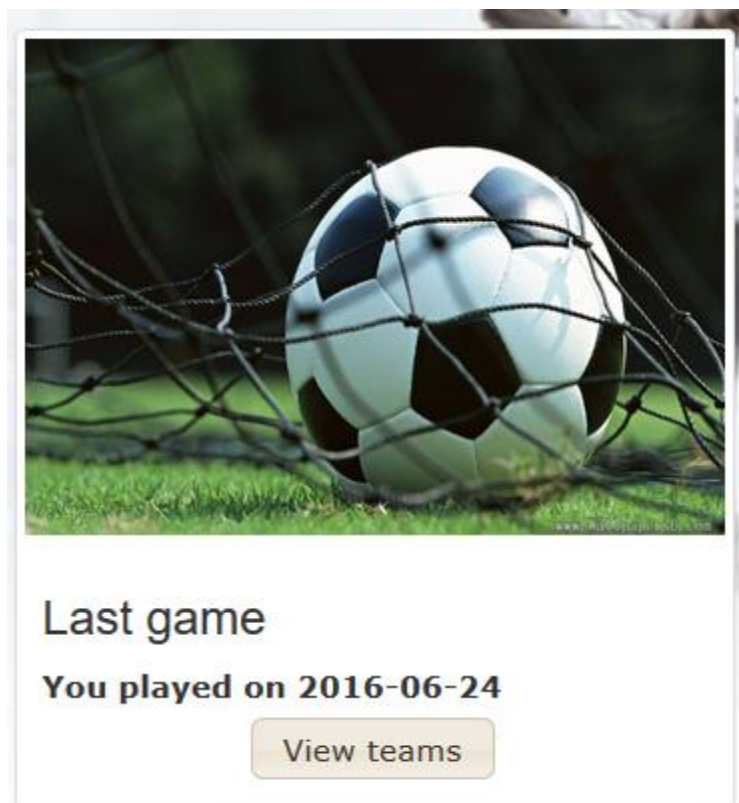
Figură 54. Top 10 jucători

Revenirea la pagina de start se poate face prin click pe imaginea din stânga, care este prevăzută cu un tool-tip util pentru navigarea în pagină.



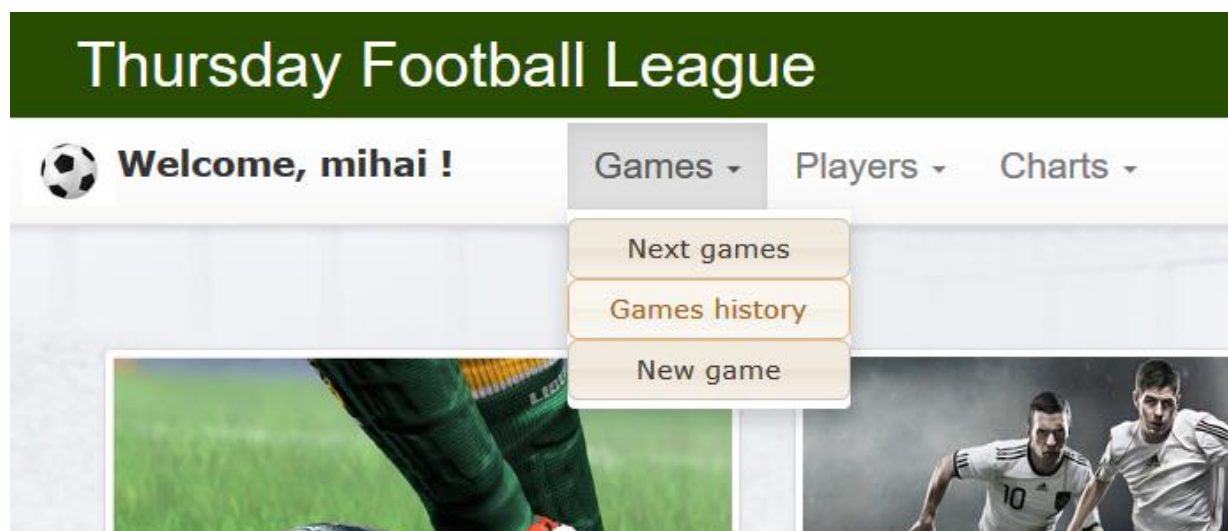
Figură 55. Revenirea la pagina Home a aplicației

Ultima secțiune prezentă în pagina principală este cea dedicată meciurilor ce au avut loc. Aici avem informații despre ultimul meci la care a participat user-ul logat. La click pe butonul 'View teams' utilizatorul este redirectat către pagina de vizualizare a echipelor, unde poate introduce și rezultatul meciului.



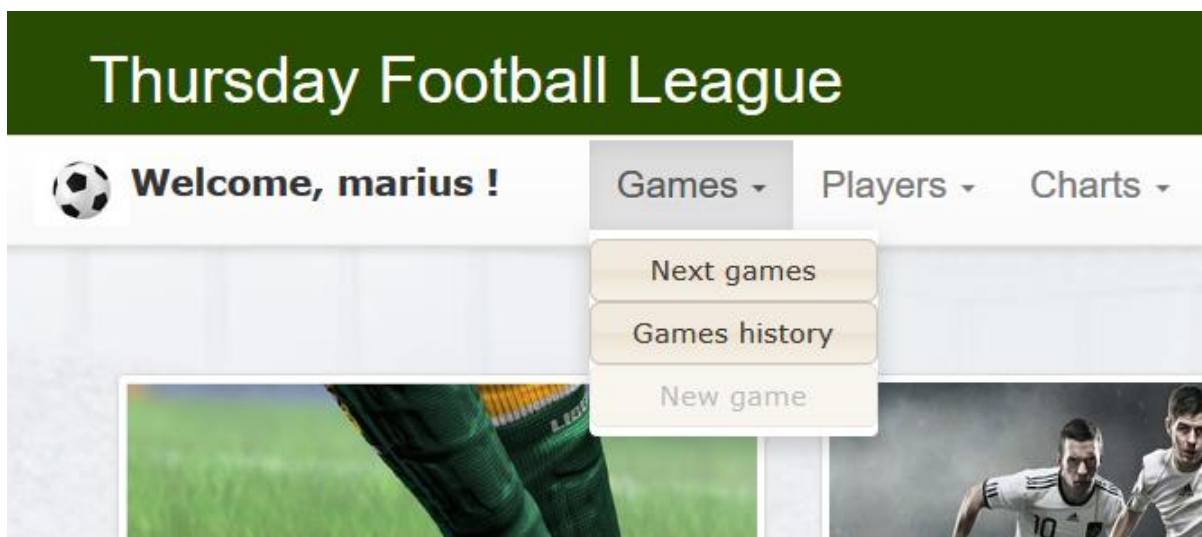
Figură 56. Prezentarea informațiilor despre meciul anterior

Utilizatorii pot naviga în cadrul aplicației prin intermediul meniului din partea de sus a paginii, care cuprinde secțiunile: Games, Players, Charts. Fiecare secțiune are mai multe subsecțiuni, unele dintre acestea nefiind active în cazul în care utilizatorul nu are drepturi de administrator. Meniul pentru un utilizator de tip administrator, are toate butoanele active.



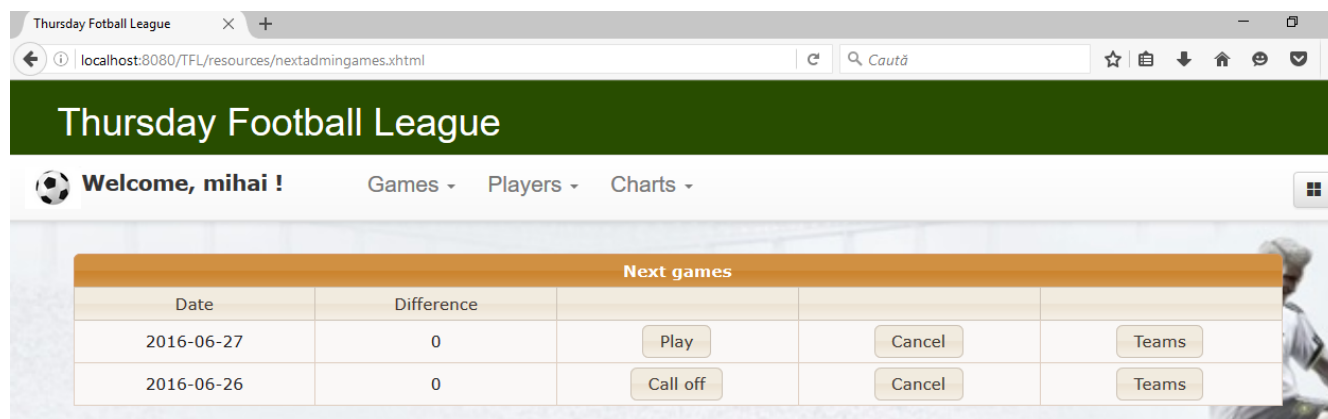
Figură 57. Prezentarea meniului pentru un utilizator de tip admin

Un utilizator fără drepturi de administrator nu are activă opțiunea pentru adăugarea unui joc nou.



Figură 58. Prezentarea meniului pentru un utilizator fără drepturi de admin

Jucătorul are opțiunile de vizualizare a jocurilor următoare, vizualizare a istoricului jocurilor, sau stabilirea datei pentru un meci următor. La click pe butonul Next games, se va afișa o listă cu următoarele jocuri. Pentru fiecare joc, există 3 opțiuni în cazul unui user admin: se poate înscrie sau poate anula înscrierea făcută, poate anula jocul, sau poate consulta echipele.



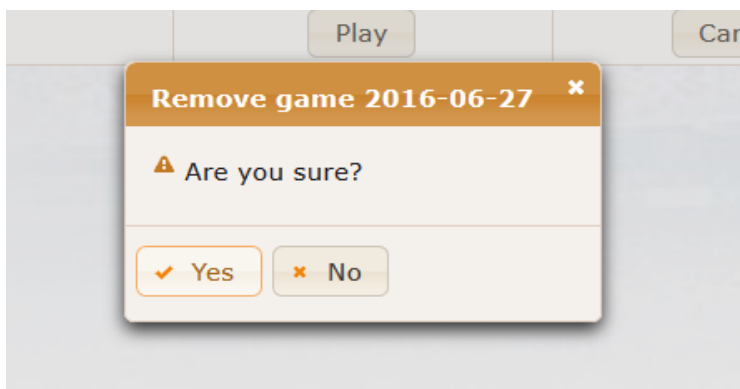
Figură 59. Opțiunile pentru meciurile anterioare

După fiecare acțiune efectuată, utilizatorul este informat în legătura cu reușita acesteia. De exemplu, dacă utilizatorul alege să nu mai joace în jocul din data de 26-06-2016, va primi un mesaj de confirmare și va fi de asemenea actualizată pagina web.



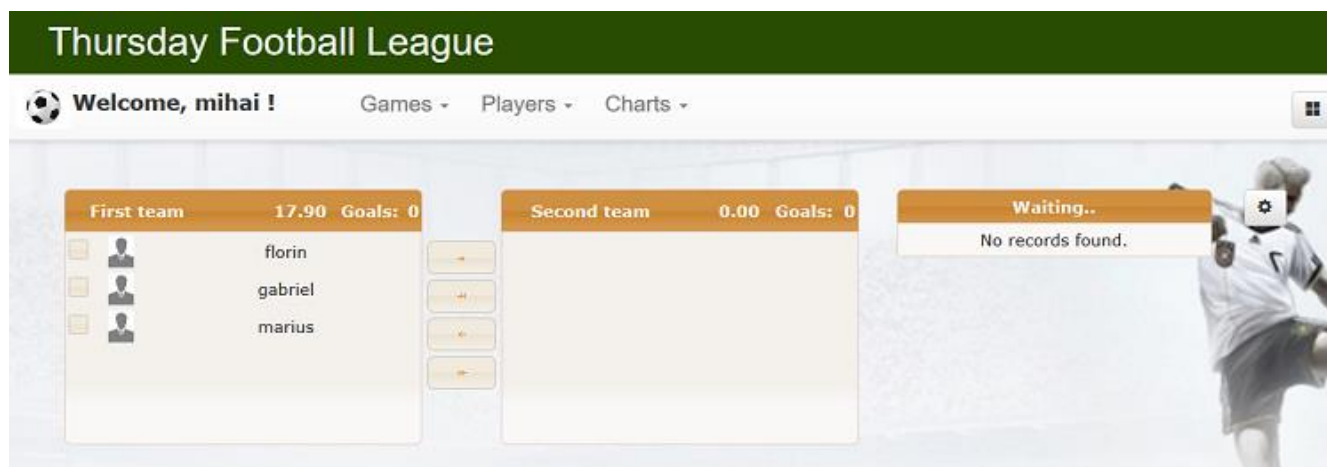
Figură 60. Informarea utilizatorului cu privire la acțiunea solicitată

Dacă se alege anularea unui joc, utilizatorului i se cere mai întâi o confirmare, pentru a avea siguranța că butonul nu a fost apăsat din greșeală. Așadar, meciul de fotbal este anulat numai în cazul în care este confirmat acest lucru, prin apăsarea butonului Yes. În caz contrar, nu are loc nicio acțiune.



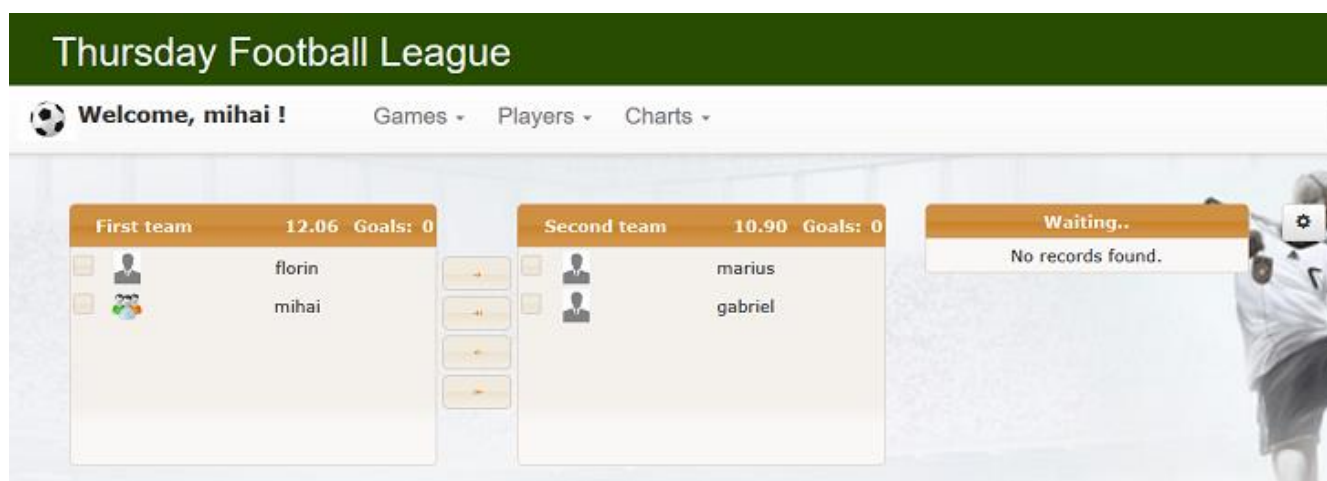
Figură 61. Confirmarea anulării unui joc

La apăsarea butonului 'Teams', utilizatorul poate vedea echipele formate până în momentul actual, echipe care pot fi modificate însă până la data jocului. Împărțirea jucătorilor pe echipe se face doar în momentul în care sunt minim 4 utilizatori înscriși la joc. De exemplu, la meciul din data de 26.06.2016 din Figura58, sunt înscriși 3 jucatori.



Figură 62. Vizualizarea meciului înainte de a se atinge numărul minim de jucători

Se poate observa faptul că echipele nu sunt formate încă. Dacă utilizatorul logat în acest moment se înscrie la acest joc, se face împărțirea jucătorilor în două echipe echilibrate.



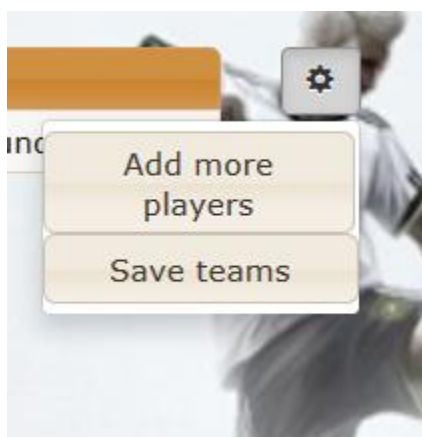
Figură 63. Echipele meciului după adăugarea unui nou jucător

Deoarece utilizatorul logat în acest moment are drepturi de administrator, el poate schimba configurația curentă a echipelor, scorul fiecărei echipe actualizându-se, și poate salva noua configurație. Scorul primei echipe este 12.06, iar a celei de-a doua 10.90. După interschimbarea jucătorilor Mihai și Marius, noul scor pentru fiecare echipă este: 12.50, respectiv 10.46.



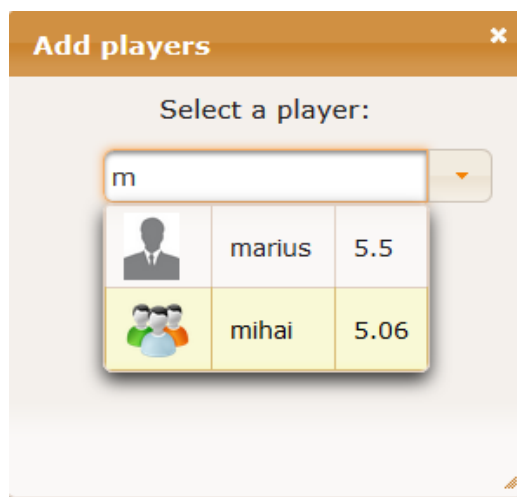
Figură 64. Interschimbarea de jucători în cadrul echipelor

Salvarea noii configurații se face prin click pe butonul din dreapta tabelului cu echipe, apoi 'Save teams'.



Figură 65. Salvarea configurației curente


Se pot adăuga și noi jucători echipelor curente, dacă nu s-a depășit numărul maxim de jucători permiși, prin click pe butonul 'Add more players'. Va apărea o fereastră, de unde se poate alege jucătorul ce se dorește a fi adăugat, prin selectarea acestuia din lista de jucători apărută.



Add players [X]

Select a player:

Search: m

	marius	5.5
	mihai	5.06

Figură 66. Adăugarea de noi jucători în cadrul echipelor

Prin selectarea opțiunii ‘Games history’ din meniu, se va afișa un istoric al jocurilor, utilizatorii având opțiunea de a vedea echipele și de a introduce rezultatul jocului, prin click pe butonul ‘Add result’.

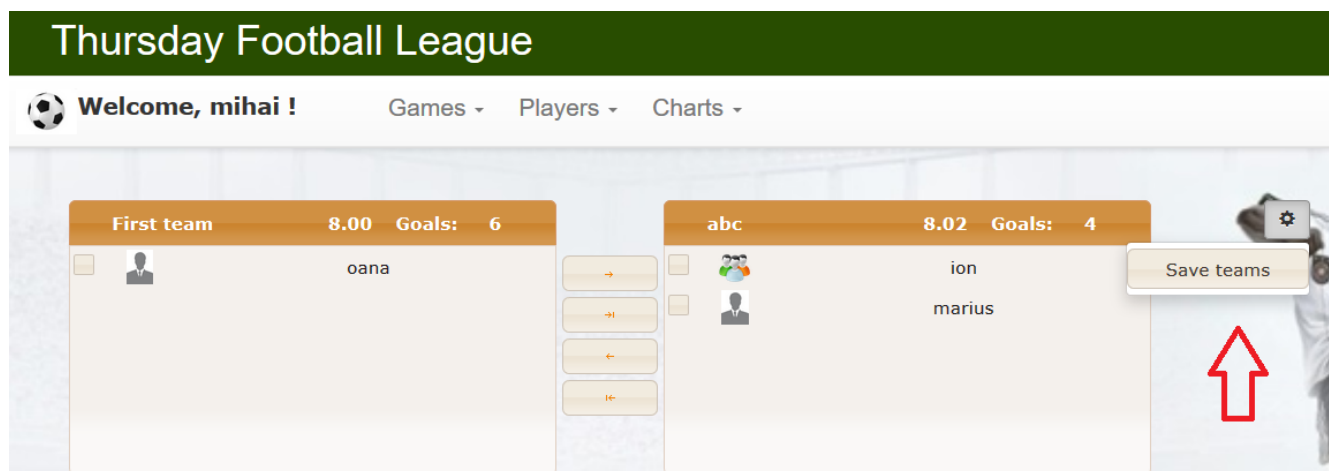


 **Welcome, mihai !**
Games ▾ Players ▾ Charts ▾

Games history		
Date	Difference	
2016-06-24	1	Add result
2016-06-25	0	Add result
2016-06-23	5	Add result

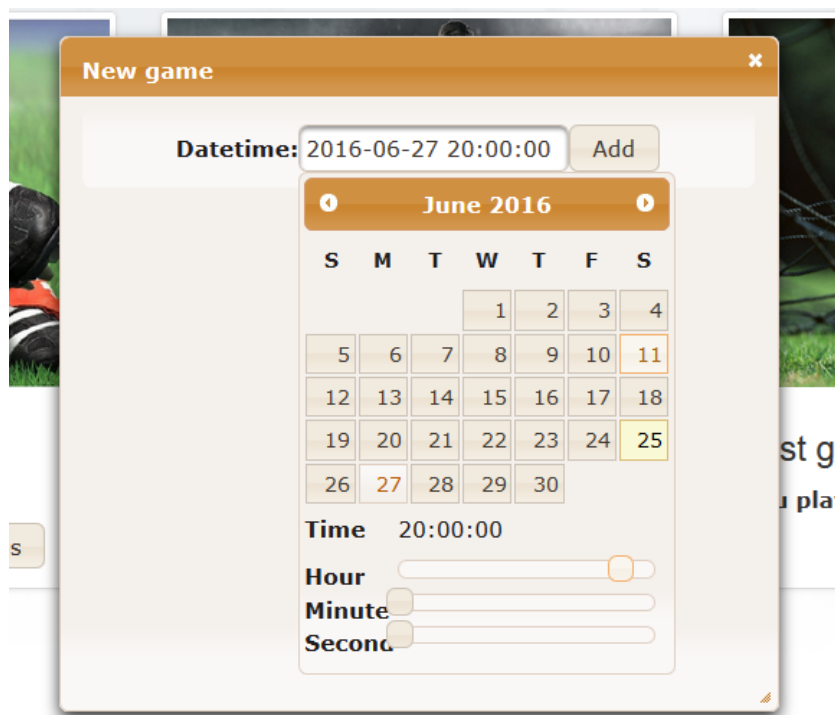
Figură 67. Opțiunile pentru un meci anterior

Se poate introduce numărul de goluri pentru fiecare echipă, se pot schimba numele echipelor, sau jucătorii între echipe, totate aceste schimbări persistându-se în baza de date, rating-ul jucătorilor fiind de asemenea recalculat în funcție de diferența de goluri. În exemplul din Figura64, am schimbat numele celei de-a doua echipe și am introdus numărul de goluri marcate de fiecare dintre echipe.



Figură 68. Exemplu introducere rezultat meci

Dacă se dorește stabilirea datei pentru următorul meci de fotbal, din meniu se alege opțiunea ‘New game’, iar în fereastra apărută se alege data și ora jocului. În Figura65, am stabilit un joc pentru data de 27.06.2016 la ora 20:00.



Figură 69. Stabilirea următorului meci de fotbal

Secțiunea destinată jucătorilor cuprinde două opțiuni: vizualizarea tuturor jucătorilor și adăugarea unui nou jucător, disponibilă numai pentru administratori. Pentru a vedea jucătorii, se face click pe ‘View players’, utilizatorul fiind redirectat către pagina ce va conține informații despre jucători.

Thursday Football League				
<div> Games ▾ Players ▾ Charts ▾ </div>				
Picture	Name	Rating		
	ion	2.52	Remove from players	Add to chart
	vasile	10.0	Remove from players	Add to chart
	florin	7.0	Remove from players	Add to chart
	oana	5.0	Remove from players	Add to chart
	feo	3.0	Remove from players	Add to chart

Figură 70. Vizualizarea tuturor jucătorilor

Pentru fiecare jucator, un user de tip admin poate efectua următoarele acțiuni, poate șterge (logic) din baza de date, jucătorul, sau îl poate adăuga la diagrama jucătorilor, pentru a vedea graficul evoluției acestuia. De exemplu, prin adăugarea jucătorului Mihai la diagrama, apoi accesarea diagramei din Charts->View chart, obținem următorul grafic:

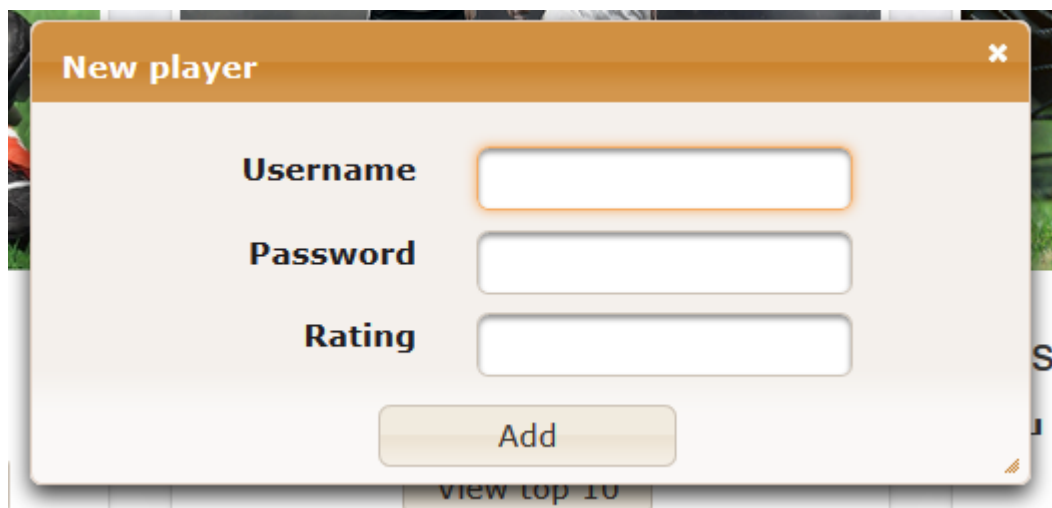


Figură 71. Exemplu grafic jucător

În diagramă se poate observa o evoluție a rating-ului jucătorului Mihai, de la data de 22 iunie până pe 24 iunie, apoi o scădere în data de 26 iunie.

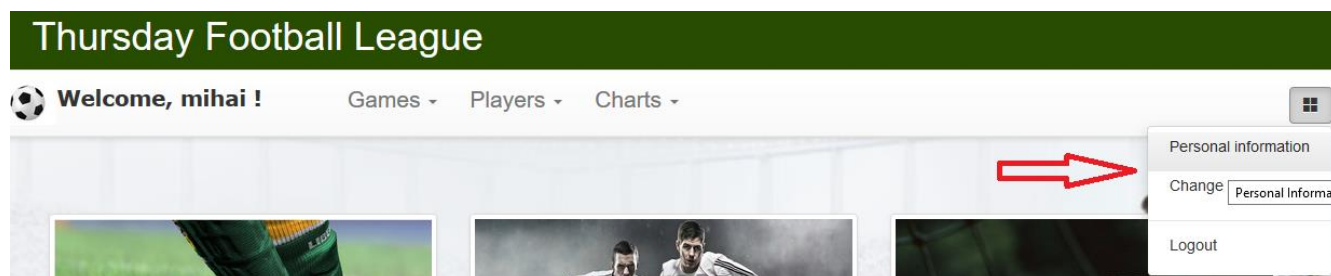
Dacă se dorește adăugarea unui nou jucător, după click pe tabul 'Players' din meniul și alegerea opțiunii 'Add new player', se vor completa câmpurile: nume, parola și rating. Câmpul rating nu este

obligatoriu, în cazul în care este lăsat liber, utilizatorul înregistrat va avea rating-ul 5, rating-ul maxim fiind în jurul valorii 10.

A modal window titled "New player" with a close button (X) in the top right corner. It contains three input fields labeled "Username", "Password", and "Rating". Below the fields is an "Add" button. The modal is overlaid on a blurred background of a website.

Figură 72. Exemplu adăugare jucător nou

Utilizatorii mai au opțiunile de a-și schimba parola și a-și vizualiza și modifica datele persoanele. Pentru a putea vizualiza datele sale personale, user-ul trebuie să aleagă din meniu, opțiunea 'Personal information', precum în Figura 69.




Figură 73. Accesarea informațiilor personale

Informațiile cuprinse în pagina încărcată sunt: numele jucătorului logat, rating-ul, numărul de jocuri jucate, numărul de jocuri câștigate, numărul de jocuri pierdute, precum și disponibilitatea acestuia. De asemenea, tot din aceasta pagină, jucatorii își pot schimba fotografia de profil, prin click pe butonul 'Choose'. Userii pot schimba și disponibilitatea acestora. Prin setarea opțiunii available 'false', un jucător nu se mai poate înscrie la meciuri, și nu poate fi inclus în echipe de alți utilizatori.

Thursday Football League

Welcome, mihai ! Games ▾ Players ▾ Charts ▾



Name: mihai

Rating: 5.06

Played games: 4

Winner: 2

Loser: 2

Available: ☐ off

Figură 74. Schimbarea pozei de profil si a disponibilității

Dacă se dorește schimbarea parolei curente, din meniu se alege ‘Change password’ și se completează câmpurile din formular, precum în Figura71.

Thursday Football League

Welcome, mihai ! Games ▾ Players ▾ Charts ▾

Current password:
 New password:
 Confirm password:

Figură 75. Schimbare parolă

În final, pentru ieșirea din aplicație se face click pe ‘Logout’, sesiunea curentă se închide și se face redirectarea către pagina de start a aplicației.

4.6 Index de figuri

Figură 1. Crearea unui proiect Maven	5
Figură 2. Conceptele Maven	5
Figură 3. Dependințe funcționale incluse cu Maven.....	6
Figură 4. Arhitectura MVC (Model-View-Controller)	9
Figură 5. Dependințele JSF	11
Figură 6. Crearea unui ManagedBean.....	11
Figură 7. Exemplu de folosire a JSF în HTML.....	12
Figură 8. Exemplu formular JSF	12
Figură 9. Adăugarea dependenței pentru temele Primefaces.....	13
Figură 10. Tema 'Humanity' Primefaces	13
Figură 11. Primefaces Panel.....	14
Figură 12. Primefaces Autocomplete.....	15
Figură 13. Primefaces Chart.....	16
Figură 14. Primefaces PickList	16
Figură 15. Interfața pgAdmin.....	19
Figură 16.Arhitectura JPA	21
Figură 17. Diagrama use-case utilizator fără drepturi de administrator	28
Figură 18. Diagrama use-case utilizator de tip administrator	29
Figură 19.Clasa Player	30
Figură 20. Clasa 'Game'	31
Figură 21. Pachetul 'DataAccessLayer'	31
Figură 22. Clasa PlayerDataAccess	32
Figură 23. Diagrama claselor din pachetul 'dataAccessLayer'.....	32
Figură 24. Pachetul 'Helpers'	33
Figură 25. Clasa RatingComparator.....	34
Figură 26. Clasa PlayerConverter	35
Figură 27. Sintaxa elementului Picklist cu atributul 'converter'	35
Figură 28. Reprezentarea echipelor cu ajutorul elementului PickList	35
Figură 29. Diagrama claselor pachetului 'helpers'	36

Figură 30. Pachetul 'views'	36
Figură 31. ManagedBean-ul autoCompleteView	37
Figură 32. Fereastra pentru adăugarea unui nou jucător	37
Figură 33. Diagrama claselor pachetului 'views'	38
Figură 34. Conținutul folderului 'WebContent'	39
Figură 36. Baza de date a aplicației	41
Figură 37. Crearea conexiunii la baza de date PostgreSQL	42
Figură 38. Alegerea driver-ului pentru conexiune	42
Figură 39. Verificarea existenței JPA în cadrul proiectului	43
Figură 40. Maparea entităților din baza de date	43
Figură 41. Clasele rezultate în urma mapării JPA a entităților	44
Figură 42. Formula generală combinari	44
Figură 43. Lista jucători pentru împărțirea în echipe	45
Figură 44. Rezultatul împărțirii în echipe e listei	45
Figură 45. Înscrierea unui jucător în cadrul unui meci	46
Figură 46. Pornirea serverului Tomcat din mediul de dezvoltare Eclipse	47
Figură 47. Pornirea serverului Tomcat din linia de comandă	48
Figură 48. Verificarea pornirii corecte a serverului Tomcat	48
Figură 49. Accesul la aplicație prin intermediul unui browser web	49
Figură 50. Înregistrarea unui nou jucător	49
Figură 51. Exemplu autentificare cu date greșite	50
Figură 52. Pagina către care este redirectat un utilizator după autentificare	50
Figură 53. Prezentarea informațiilor despre meciul următor	51
Figură 54. Prezentarea informațiilor despre ultimii jucători înscriși	51
Figură 55. Top 10 jucători	52
Figură 56. Revenirea la pagina Home a aplicației	52
Figură 57. Prezentarea informațiilor despre meciul anterior	53
Figură 58. Prezentarea meniului pentru un utilizator de tip admin	53
Figură 59. Prezentarea meniului pentru un utilizator fără drepturi de admin	54
Figură 60. Opțiunile pentru meciurile anterioare	54
Figură 61. Informarea utilizatorului cu privire la acțiunea solicitată	55

Figură 62. Confirmarea anulării unui joc.....	55
Figură 63. Vizualizarea meciului înainte de a se atinge numărul minim de jucători.....	56
Figură 64. Echipele meciului după adăugarea unui nou jucător	56
Figură 65. Interschimbarea de jucători în cadrul echipelor	57
Figură 66. Salvarea configurației curente	57
Figură 67. Adăugarea de noi jucători în cadrul echipelor.....	58
Figură 68. Opțiunile pentru un meci anterior.....	58
Figură 69. Exemplu introducere rezultat meci.....	59
Figură 70. Stabilirea următorului meci de fotbal	59
Figură 71. Vizualizarea tuturor jucătorilor	60
Figură 72. Exemplu grafic jucător	60
Figură 73. Exemplu adăugare jucător nou	61
Figură 74. Accesarea informațiilor personale	61
Figură 75. Schimbarea pozei de profil si a disponibilității	62
Figură 76. Schimbare parolă	62

5 Concluzii

În concluzie, Aplicația suport pentru constituirea de echipe echilibrate și personalizabile, are utilitate în lumea reală, putând fi folosită de grupul format din angajații unei companii, care doresc să se organizeze pentru a desfășura activități sportive, precum meciurile de fotbal. Reprezintă o modalitate interactivă și utilă de organizare. (sfdg)

Ca și perspective de viitor, se pot introduce mai multe criterii pe baza cărora să se facă împărțirea jucătorilor în cele două echipe. Un criteriu de care se mai poate ține cont ar putea fi preferințele jucătorilor pentru alți jucători. Astfel, la formarea echipelor se va ține cont de jucătorii pe care ar dori fiecare utilizator să îi aibă în echipă.

Pe o durată mai mare de timp, aplicația ar putea fi utilă chiar pentru organizare unor campionate de fotbal, participanții la campionate putând fi grupuri diferite de persoane.

6 Bibliografie

1. *Inițiere în programarea orientată pe obiecte din perspectivă C++ și Java*- Dorin Bocu
2. „*Thinking in Java*” -Bruce Eckel. Retrieved from „Thinking in Java” -Bruce Eckel
3. „*Data Structures and Algorithms with Object-Oriented-Design-Patterns in Java*” - Bruno R. Preiss.
4. <http://www.tutorialspoint.com/maven/>
5. <http://tutorials.jenkov.com/maven/maven-tutorial.html>
6. <https://maven.apache.org/guides/getting-started/>
7. <https://docs.jboss.org/hibernate/orm/3.6/quickstart/en-US/html/hibernate-gsg-tutorial-jpa.html>
8. <http://www.vogella.com/tutorials/ApacheMaven/article.html>
9. <http://www.roseindia.net/jsf/>
10. <http://www.tutorialspoint.com/maven/>
11. <http://docs.oracle.com/javaee/6/tutorial/doc/bnaph.html>
12. <http://www.oracle.com/technetwork/articles/javase/index-142890.html>
13. <http://whatis.techtarget.com/definition/model-view-controller-MVC>
14. <http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>
15. <http://alvinalexander.com/ooa-ood/java-model-view-controller-mvc-example-1>
16. http://www.tutorialspoint.com/design_pattern/mvc_pattern.html
17. <http://web.princeton.edu/sites/isapps/jasig/2004summerWestminster/Presentations/java%20server%20faces.pdf>
18. <http://www.tutorialspoint.com/jsf/>
19. http://profs.info.uaic.ro/~acf/tap/slides/jsf_slide.pdf
20. <http://www.coreservlets.com/JSF-Tutorial/primefaces/>
21. <http://www.journaldev.com/2990/primefaces-5-jsf-2-beginners-example-tutorial>
22. <http://www.primefaces.org/gettingStarted>
23. http://www.primefaces.org/docs/guide/primefaces_user_guide_5_3.pdf
24. https://en.wikipedia.org/wiki/Bootstrap_%28front-end_framework%29
25. http://www.w3schools.com/bootstrap/bootstrap_get_started.asp
26. <https://www.postgresql.org/docs/7.4/static/jdbc-binary-data.html>
27. <https://en.wikipedia.org/wiki/PostgreSQL>
28. *Suport de curs Java JPA* -Silviu, Dumitrescu.
29. <http://www.vogella.com/tutorials/JavaPersistenceAPI/article.html>

30. https://en.wikipedia.org/wiki/Java_Persistence_API

31. <http://www.tutorialspoint.com/jpa/>

32 <http://www.objectdb.com/java/jpa/persistence>