

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE



UNIVERSITATEA TEHNICĂ

DIN CLUJ-NAPOCA

**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE**

SISTEM PENTRU MANAGEMENTUL UNUI COMPLEX SPORTIV

LUCRARE DE LICENȚĂ

Absolvent:

Andra Lavinia ROMAN

Coordonator
științific:

Ș.I. Ing. Cosmina IVAN

2017



UNIVERSITATEA TEHNICĂ

DIN CLUJ-NAPOCA

**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE**

DECAN,
Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent: **Prenumele NUMELE**

TITLUL LUCRĂRII DE LICENȚĂ

1. **Enunțul temei:** *Proiectul își propune realizarea unui sistem pentru rapid managementul abonamentelor și al programărilor din cadrul unui complex sportiv, cât și managementul angajaților acestui complex.*
2. **Conținutul lucrării:** *Cuprins, Introducere, Obiectivele proiectului, Studiu Bibliografic, Analiză și Fundamentare Teoretică, Proiectare de Detaliu și Implementare, Testare, Validare și Evaluare, Manual de Instalare și Utilizare, Concluzii, Bibliografie, Anexe*
3. **Locul documentării:** Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare
4. **Consultanți:**
5. **Data emiterii temei:** 1 noiembrie 2015
6. **Data predării:** 14 Iulie 2017

Commented [c1]: se va completa data predării

Absolvent: _____

Coordonator științific: _____

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE



UNIVERSITATEA TEHNICĂ

DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE



UNIVERSITATEA TEHNICĂ

DIN CLUJ-NAPOCA

**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE**

**Declarație pe proprie răspundere privind
autenticitatea lucrării de licență**

Subsemnatul(a) Roman Andra Lavinia, legitimat(ă) cu cartea de identitate seria MM nr. 610109 CNP 2940526240016, autorul lucrării Sistem pentru managemnetul unui complex sportiv elaborată în vederea susținerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea Tehnologia Informației din cadrul Universității Tehnice din Cluj-Napoca, sesiunea Iulie 2017 a anului universitar 2016-1017, declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, pe baza cercetărilor mele și pe baza informațiilor obținute din surse care au fost citate, în textul lucrării, și în bibliografie.

Declar, că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în fața unei alte comisii de examen de licență.

În cazul constatării ulterioare a unor declarații false, voi suporta sancțiunile administrative, respectiv, *anularea examenului de licență*.

Data

Nume, Prenume

Semnătura

Cuprins

Capitolul 1. Introducere – Contextul proiectului.....	1
1.1. Contextul proiectului	1
1.2. Motivația	1
1.3. Conținutul lucrării	2
Capitolul 2. Obiectivele Proiectului	3
2.1. Obiectivul principal	3
2.2. Obiective secundare.....	3
2.2.1. Obiective generale	3
2.2.2. Managementul clienților	4
2.2.3. Managementul vizitatorilor	4
2.2.4. Managementul programărilor.....	5
2.2.5. Managementul angajaților.....	5
2.2.6. Vizualizarea unei galerii foto și a unei descrieri	5
2.2.7. Vizualizarea unui calendar.....	5
2.2.8. Trimiterea unor mail-uri de confirmare	5
2.2.9. Generarea unor rapoarte.....	5
2.2.10. Vizualizarea unui dashboard	5
Capitolul 3. Studiu Bibliografic	6
3.1. Dezvoltarea aplicațiilor web	6
3.2. Arhitectura Model View Controller	6
3.3. Sisteme similare	7
3.3.1. Aplicația “Gestiune sală”	7
3.3.2. Aplicația “DashPlatform Recreational Management Software”	9
3.3.3. Aplicația “Virtuagym”	9
Capitolul 4. Analiză și Fundamentare Teoretică	12
4.1. Tehnologii și concept utilizate pentru dezvoltarea aplicație web.....	12
4.1.1. Java EE.....	12
4.1.2. Spring boot framework	12
4.1.3. Maven.....	13
4.1.4. Java Persistence with Hibernate	13
4.1.5. JavaServer Pages	15
4.1.6. RESTful Web Services	16

4.1.7. Baza de date.....	18
4.2. Cerințele sistemului	19
4.2.1. Cerințe funcționale.....	19
4.2.2. Cerințe non-funcționale.....	20
4.3. Cazuri de utilizare.....	22
4.3.1. Actorii sistemului.....	22
4.3.2. Descrierea detaliată a cazurilor de utilizare	25
Capitolul 5. Proiectare de Detaliu si Implementare.....	29
5.1. Arhitectura sistemului.....	29
5.1.1. Arhitectura aplicației web	29
5.2. Diagramele sistemului	38
5.2.1. Diagrame de navigare	38
5.2.2. Diagrama de pachete.....	40
5.3. Deployment	42
5.4. Proiectarea bazei de date.....	43
Capitolul 6. Testare și Validare.....	45
Capitolul 7. Manual de Instalare si Utilizare	47
7.1. Aplicația web.....	47
7.1.1. Instalarea și rutarea	47
Importul proiectului	47
Rularea proiectului	48
7.1.2. Manual de utilizare	48
Capitolul 8. Concluzii	52
8.1. Realizările sistemului.....	52
8.2. Dezvoltări ulterioare	52
Bibliografie.....	54
Anexa 1 (dacă este necesar).....	Error! Bookmark not defined.

Capitolul 1. Introducere – Contextul proiectului

1.1. Contextul proiectului

Jocurile si sportul trebuie sa joace un rol important in vietile noastre. Ele reprezintă moduri prin care ne putem contrui viața sociala si totodata un stil de viața sănătos. Jocurile sportive se pot împărți în două mari categorii, indoor games si outdoor games. În timp ce unii oameni prezintă un interes pentru jocurile de interior, alții preferă jocurile in aer liber, considerând sportul ca o activitate de recreere.

Având in vedere multitudinea jocurilor sportive existente, este vital ca un complex sportiv sa faca față unui numar mare de cereri si totodata să găsească un mod cât mai simplu prin care clienții pot să beneficieze de serviciile oferite.

Proiectul propus este o aplicație care oferă managementul unui complex sportiv, o experiență placută atât clientilor cât si să ușureze munca angajatilor si care implica un software rapid si performant, având in vedere concurența existentă pe piața software.

Luând in calcul multitudinea modurilor de organizare a sistemelor informatice, sistemul propus se incadreaza in categoria sistemelor informatice pentru management, asa numit MIS (Management information system). Această categorie de sisteme informatice este orientată spre a oferi suport și informații pentru manageri. Asigură suportul pentru necesitățile decizionale de la nivelul de vârf (strategic), la cel de mijloc (tactic) sau cel operațional. Sistemele informatice pentru management oferă o varietate de rapoarte specifice folosite în activitățile manageriale curente. Produsele informatice pot fi obținute la cerere, periodic sau când apar necesități excepționale.

Dacă analizăm la un nivel mai ridicat, sistemul propus se încadrează în ERP (Enterprise Resource Planning), o subcategorie a sistemelor informatice pentru management, și totodata un instrument software care facilitează integrarea tuturor informațiilor dintr-o organizație într-o platformă unică. ERP are scopul de a asigura transparența unor date în cadrul unei structuri sau organizații și facilitarea accesului aproape la orice tip de informație utilă pentru desfășurarea activității.

Software-ul sistemului de management gestionează activitățile întregului club sportiv și oferă funcționalități corespunzătoare pentru diferite tipuri de vizitatori (angajati/utilizatori/admin). Sistemul de management al clubului sportiv este construit ținând cont de diverse activități sportive de zi cu zi, atat sporturi de grup: handball, football etc. cat si activitati de aerobic si fitness. Software-ul automatizează toate aceste funcționalități ale clubului sportiv pentru operarea cat mai ușoară.

1.2 Motivația

Ținând cont de timpul necesar realizării tuturor activităților necesare pentru un complex sportiv, pentru un numar mare de clienți, apare necesitatea unui sistem software care sa cuprinda aceste activități într-un sistem cat se poate de compact, care sa ajuta la scurtarea timpului necesar pentru derularea acestor activități cât si să ofere o experiență plăcuta utilizatorilor.

Urmărind majoritatea platformelor existente pe piața software, am constatat ca ele constau in prezentarea complexelor sportive si a orarului acestora, nu și pentru

managementul diferitelor activități din cadrul complexului, cum ar fi programarea online, crearea unui cont de utilizator și comandarea abonamentelor online etc. Luând în considerare aceste detalii, am constatat că ar fi utilă o aplicație care să implementeze aceste funcționalități, să ajute la creșterea productivității angajaților și să gestioneze eficient activitățile destinate utilizatorilor.

1.3. Conținutul lucrării

În următoarele paragrafe se va prezenta structura lucrării pe capitole, precum și o scurtă descriere pentru fiecare din ele.

Capitolul 1 – Introducere – Acest capitol conține o scurtă descriere a contextului problemei și a motivației care au dus la dezvoltarea aplicației prezentate.

Capitolul 2 – Obiectivele Proiectului – În acest capitol sunt descrise și prezentate obiectivul principal și obiectivele secundare care au fost propuse spre implementare pentru sistemul prezentat.

Capitolul 3 – Studiu Bibliografic – În acest capitol sunt prezentate principalele aspecte și concepte ale dezvoltării aplicațiilor web și ale arhitecturii folosite. Tot în acest capitol, este prezentată o comparație a sistemului care s-a realizat și a sistemelor similare disponibile pe piață.

Capitolul 4 – Analiza și Fundamentare Teoretică – Acest capitol conține o descriere a tehnologiilor utilizate în dezvoltarea sistemului precum și motivele pentru care acestea au fost alese, și conceptele folosite în dezvoltare. Tot în acest capitol sunt prezentate cerințele funcționale, cerințele non-funcționale și cazurile de utilizare ale sistemului, cele mai semnificative fiind prezentate într-un mod mai detaliat.

Capitolul 5 – Proiectare de Detaliu și Implementare – În acest capitol va fi prezentat modul în care sistemul a fost proiectat. Se va prezenta schema arhitecturală a sistemului, arhitectura bazei de date și structura tabelor, diagrama de deployment și vor fi detaliate anumite componente ale aplicației considerate relevante în înțelegerea a felului în care sistemul este proiectat.

Capitolul 6 – Testare, Validare și Evaluare – Acest capitol conține o prezentare a principalele procese de testare care au fost realizate asupra sistemului propus.

Capitolul 7 – Manual de Instalare și Utilizare – În acest capitol sunt descriși pașii care trebuie urmați pentru instalarea cu succes a componentelor sistemului pe o mașină locală. Tot în acest capitol vor fi prezentate resursele software și hardware necesare rulării, precum și un manual de utilizare a aplicației.

Capitolul 8 – Concluzii – În acest capitol sunt aduse concluzii asupra sistemului dezvoltat. Se vor trece în revistă realizările și obiectivele care au fost atinse prin acest proiect, urmată de o descriere a posibilităților de dezvoltare ulterioară.

Capitolul 2. Obiectivele Proiectului

2.1. Obiectivul principal

Această lucrare se focusează pe furnizarea unei soluții software pentru un complex sportiv care deține atât o sală de fitness cât și terenuri pentru jocuri de echipă. Această soluție are rolul de a ajuta angajații complexului sportiv să gestioneze și să urmărească activitățile de zi cu zi ale complexului sportiv, să țină evidența înregistrărilor, a programărilor și programelor sportive, cât și clienții complexului sportiv prin dezvoltarea unui sistem ușor de utilizat, care să le ofere o soluție rapidă pentru crearea unui cont, programarea la activitățile sportive disponibile cât și o modalitate pentru plata online a abonamentelor.

Planificarea activităților sportive este una dintre activitățile provocatoare pentru angajați. Angajații trebuie să încerce să sincronizeze ora evenimentului cu programările planificate ale tuturor clienților complexului sportive. Datorită acestui lucru, în fiecare săptămână va fi afișat un orar pe pagina sistemului software cu activitățile disponibile pentru sălile de fitness, cât și o listă cu terenurile disponibile pentru sporturile de echipă

2.2. Obiective secundare

2.2.1. Obiective generale

În acest capitol sunt descrise câteva caracteristici generale pe care trebuie să le avem în vedere atunci când dezvoltăm un sistem software.

Un obiectiv important propus pentru acest sistem este reprezentat de utilizarea eficientă a sistemului software. Acest obiectiv este atins prin realizarea unei interfețe “user-friendly” care să fie ușor de înțeles și de utilizat. Pentru ca o interfață să fie “user-friendly” trebuie să îndeplinească o serie de caracteristici. Sistemul trebuie să fie intuitiv, interfața grafică trebuie să fie cel puțin la fel de bine gândită ca software-ul. Dacă interfața grafică nu este bine gândită și executată, clienții pot avea probleme cu utilizarea produsului. O interfață grafică bine concepută poate uneori să mascheze codificare mai puțin “prietenoasă”, dar, desigur, care funcționează cum era de așteptat. Software-ul nu trebuie să împiedice îndeplinirea unei sarcini și nici să nu creeze obstacole pentru utilizatori. Eficiența unei bucati de software este legată de intuitivitatea acesteia. O interfață “user-friendly” nu trebuie să fie complexă, trebuie să fie simplă, oferind acces rapid la funcționalitățile și comenzile sistemului. Trebuie să fie curată, bine organizată, facilitând localizarea instrumentelor și opțiunilor disponibile și să evite erorile, comportamentele neașteptate ale aplicației.

Un alt doilea obiectiv general și important realizat pentru sistemul propus este securitatea. Securitatea implică utilizarea de software, hardware și metode procedurale pentru a proteja aplicațiile de amenințările externe. În cadrul proiectului se dorește evitarea situațiilor în care un user neautorizat are acces la anumite resurse care trebuie protejate. Un exemplu bun ar fi accesarea paginilor la care au acces doar administratorul sau angajații, de către clienții complexului sportiv. Evitarea acestor situații o să se realizeze utilizând autorizarea și autentificarea.

Autentificarea se realizează pentru a constata că cineva este cu adevărat cel care pretinde că este, iar autorizarea se referă la reguli care determină cine are permisiunea de a face un anumit lucru, un exemplu bun este comparația dintre utilizatori și administrator. Utilizatorii pot să efectueze operații asupra programărilor, iar administratorii, pe lângă aceste operații, pot să gestioneze conturile utilizatorilor.

Sistemul propus trebuie să poată fi utilizat de diferite tipuri de utilizatori și anume: administrator, client, angajat și vizitator.

- **Administratorul**

Este acel tip de utilizator care o să aibă acces la toate funcționalitățile aplicației. El o să gestioneze conturile clienților, conturile angajaților, o să poată vizualiza toate programările și o să poată șterge anumite programări atunci când clienții o să încalce anumite reguli ale complexului sportiv.

- **Clientul**

Este acel tip de utilizator care are un cont creat și este logat în aplicație cu acel cont, poate să-și creeze o programare cât și să facă anumite modificări asupra programărilor făcute de el.

- **Angajatul**

Este acel tip de utilizator care poate să gestioneze conturile și programările clienților.

- **Vizitatorul**

Este acel tip de utilizator care nu are un cont creat și poate doar să vizualizeze pagina principală a sistemului și să se înregistreze.

Un al treilea obiectiv principal este extensibilitatea sistemului. Acest obiectiv reprezintă capacitatea unui sistem software de a permite și a accepta o extensie semnificativă a funcționalităților sale. Tema centrală este să asigure schimbarea, îmbunătățirea, minimizând impactul asupra funcțiilor existente ale sistemului. Un sistem extensibil este cel al cărui structură internă și fluxul de date nu sunt afectate de funcționalitatea nouă sau modificată. Un exemplu este recompilarea sau schimbarea codului sursă original, ar putea fi inutilă atunci când se schimbă comportamentul unui sistem, fie de cel care a creat sistemul, fie de alți programatori. Deoarece sistemele software sunt de lungă durată și vor fi modificate pentru caracteristici noi și funcționalități solicitate de utilizatori, extensibilitatea le permite dezvoltatorilor să se extindă sau să adauge funcționalități prin reutilizarea sistematică a codului.

2.2.2. Managementul clienților

Sistemul trebuie să ofere posibilitatea managerului și a angajaților complexului sportiv să efectueze operații CRUD asupra clienților, să trimită mail-uri de informare clienților în cazul în care o activitate sportivă este anulată din cauza unor condiții neprevăzute. Fiecare client trebuie să aibă un abonament pentru a putea participa la activitățile din cadrul complexului sportiv.

2.2.3. Managementul vizitatorilor

Având în vedere că vizitatorii au posibilitatea doar de a vizualiza pagina principală a complexului sportiv și de a-și crea un cont, administratorii și angajații pot doar să țină evidența vizitatorilor care și-au creat un cont.

2.2.4. Managementul programărilor

Fiecare client poate să efectueze operații CRUD asupra programărilor efectuate de el, iar orice angajat sau administratorul poate să efectueze operații CRUD asupra tuturor programărilor făcute de clienți. Fiecare programare trebuie să fie asignată unei activități din cadrul complexului sportiv.

2.2.5. Managementul angajaților

Administratorul este singurul utilizator care poate să efectueze operații asupra clienților.

2.2.6. Vizualizarea unei galerii foto și a unei descrieri

Pentru fiecare activitate sportivă disponibilă, utilizatorul va putea vizualiza o galerie foto și o mică descriere reprezentativă activității sportive alese. Prin această cale, utilizatorul își va putea face o idee despre fiecare activitate disponibilă în cadrul complexului sportiv.

2.2.7. Vizualizarea unui calendar

Fiecare utilizator va avea posibilitatea de vizualiza un orar cu sporturile disponibile în fiecare zi și intervalul orar pentru fiecare activitate sportivă.

2.2.8. Trimiterea unor mail-uri de confirmare

Fiecare utilizator după ce va efectua o programare sau va comanda un abonament va primi un mail de confirmare cu datele programării, respectiv a comenzii.

2.2.9. Generarea unor rapoarte

Administratorul are posibilitatea de a genera anumite rapoarte referitoare la clienți, angajați, programări sau abonamente.

2.2.10. Vizualizarea unui dashboard

Toți vizitatorii aplicației, atât cei cu cont cât și cei neînregistrați în sistem, au posibilitatea de a vizualiza un dashboard, pe pagina principală a aplicației, cu diferite date ale sistemului: numărul de utilizatori înscrși în sistem, numărul de abonamente care expiră în ziua curentă, numărul de abonamente create în ziua curentă și numărul de programări efectuate într-o zi.

Capitolul 3. Studiu Bibliografic

În acest capitol s-a realizat un studiu referitor la aplicațiile similare existente atât local cât și în alte țări. Acest studiu s-a realizat pentru a se putea identifica potențialele idei care stau la baza aplicației propuse, realizându-se o comparație a tuturor sistemelor similare și a sistemului propus pentru implementare.

3.1. Dezvoltarea aplicațiilor web

O aplicație web este un program care utilizează browsere web și tehnologii web pentru a efectua diferite task-uri pe internet. Un server pentru aplicații web publică conținutul unui fișier WAR sub o rădăcină URL definită (numită și context root), apoi direcționează cererile web către resursele corecte și returnează răspunsul web adecvat celui care a solicitat cererea. Anumite solicitări pot fi mapate la o resursă statică simplă, cum ar fi fișierele HTML și imaginile. Alte cereri, care sunt denumite resurse dinamice, sunt mapate la o anumită clasă JSP sau servlet. Prin aceste cereri, se inițiază logica Java pentru o aplicație web și se procesează apelurile către business logic-ul principal.

Atunci când se primește o solicitare web, serverul de aplicații examinează context root-ul adresei URL pentru a identifica pentru care WAR cererea este destinată și serverul examinează conținutul după root pentru a identifica ce resursă să trimită cererea. Această resursă ar putea fi o resursă statică (fișier HTML), conținutul acesteia fiind returnat sau o resursă dinamică (servlet sau JSP), unde prelucrarea cererii este trimisă către JSP sau servlet.

În fiecare fișier WAR, meta informația descriptivă, descrie aceste informații și ghidează web server-ul în implementarea și executarea servlet-urilor și JSP-urilor în cadrul aplicației web. Structura acestor elemente din fișierul WAR este standardizată și compatibilă între diferite servere de aplicații web.

3.2. Arhitectura Model View Controller

Conceptul de Model View Controller(MVC) este un model utilizat frecvent la descrierea și construirea aplicațiilor care conțin o interfață cu utilizatorul, inclusiv aplicațiile Java EE.

Ținând cont de conceptul MVC, o aplicație software sau un modul trebuie să aibă partea de business logic(modelul) separată de partea de presentation logic(view-ul). Această separare este importantă deoarece partea de presentation logic poate să se schimbe în timp și este posibil să nu fie necesară și schimbarea părții de business logic. De asemenea, multe aplicații pot avea mai multe view-uri ale aceluiași model. Dacă view-ul nu este separat, adăugarea unui view poate provoca perturbări considerabile și creșterea complexității componentei.

Această separare poate fi realizată prin organizarea componentei într-un layer pentru model(responsabil pentru implementarea părții de business logic) și un layer pentru view(responsabil pentru redarea interfeței pentru un anumit tip de client). În plus, layer-ul de controler se află între cele două layere, interceptând cererile din layer-ul view(sau prezentare) și mapându-le la apelurile către business logic, apoi returnând răspunsul pe baza unei pagini de răspuns selectate de layer-ul de controler. Avantajul

cheie oferit de layer-ul controler este acela că view-ul se poate concentra doar pe aspectele de prezentare ale aplicației și lasă layer-ul de controller să realizeze maparea.

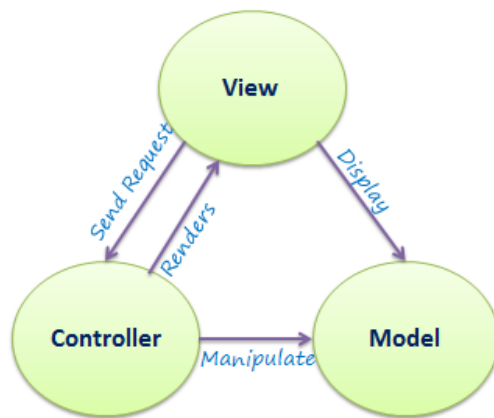


Figura 3.1 Arhitectura MVC

3.3. Sisteme similare

Ca și studiu bibliografic, am ales o serie de aplicații similare cu aplicația software care o să fie implementată. Aplicațiile alese vor evidenția cât mai bine ideea proiectului ales „managementul unui complex sportiv” și cu siguranță vor conține anumite specificații, considerate ca fiind cele mai importante pt client:

- Un design complex dar simplu de utilizat, care-i permite utilizatorul să acceseze cu ușurință funcționalitățile aplicației.
- Un mod cât mai simplu de înregistrare pt clienți
- Un mod cât mai simplu de a face programări

3.3.1. Aplicația „Gestiune sală”

<http://www.gestiunesala.ro>

Aplicația „Gestiune Sala” este perfectă atât pentru începători cât și pentru experți. Gestiune Sala este o **platforma cloud** pentru centre sportive, nu depinde de instalarea unui software și este compatibilă cu toate sistemele de operare (Windows, Linux, MacOS, Android, IOS, etc).

Pe pagina principală, aplicația are un dashboard personalizat care indică anumite statistici referitoare complexului sportiv, într-un timp real. Acesta este util, reprezentând un mod prin care clienții își pot face o idee despre complexul sportiv. Dashboardul indică următoarele statistici:

- numărul de abonați prezenți în sală
- totalul vânzărilor din abonamente și din produse, pentru ziua curentă
- numărul de abonați care au comenzi neachitate
- numărul de abonamente care expiră în ziua curentă.

Folosind orice device, administratorul se poate loga in contul de admin de unde poate efectua diferite operații:

- Adaugare de utilizatori
- Adăugarea diferitelor informații despre personalul de la recepție, cum ar fi: nume, adresă de e-mail sau telefon
- Limitarea accesului la diferite facilități, poate alege la ce facilități are acces fiecare utilizator in parte
- Trimiterea de notificări, poate porni sau opri trimiterea de notificări automate către abonați
- Personalizarea dashboard-ului în funcție de utilizator; poate limita vizibilitatea activităților zilnice pentru personalul de la recepție

Fiecare abonat poate efectua următoarele operații după logare:

- Adaugare unei fotografii. Abonatul poate să încarce o poză din calculatorul personal sau poate opta pentru modalitatea de a se fotografia instant folosind webcam-ul
- Adăugarea informațiilor de care are nevoie. Poate păstra date specifice (nume, telefon, e-mail, etc) sau observații utile pentru personalul de la recepție.
- Alocarea unui card de membru. Fiecare abonat își poate asocia un card de membru pe care îl poate personaliza cu poza și numele său.
- Setarea notificărilor individuale. Poate să decidă dacă vor fi trimise notificări prin e-mail sau nu.
- Adăugarea de abonamente noi, din lista predefinită, stabilirea perioadei de valabilitate și aplicarea unui discount, unde este cazul.
- Alegerea tipului de abonament dorit. Poate defini abonamente cu un număr finit sau nelimitat de ședințe.
- Selectarea rapidă a unei perioade de valabilitate dorită: o zi, o săptămână, o lună, 2 luni, 3 luni, 4 luni, sau un interval personalizat.
- Vizualizarea detaliilor abonamentelor pe care le-a cumpărat în trecut, dar și a abonamentelor active.
- Fiecare abonament poate fi asociat unui abonat printr-o scanare de card. Prin scanare, are acces la datele abonatului. Se pot adăuga abonamente noi, se poate stabili perioada de valabilitate sau se poate aplica un discount.
- Fiecare abonat poate să își vizualizeze comenzile, abonamentele active și coșul de cumpărături.

Pe lângă aceste funcționalități aplicația oferă și următoarele beneficii:

- Se pot construi promoții în funcție de vârstă, sex sau abonamentele pe care le cumpără.
- Număr nelimitat de utilizatori, abonați, abonamente și produse
- Statistici și sugestii care te ajută să-ți dezvolti afacerea
- Acces nelimitat și în timp real la datele tale
- Suport nelimitat 24 x 7

3.3.2. Aplicația “DashPlatform Recreational Management Software”

Am ales această aplicație datorită faptului că este folosită de utilizatori din alte părți ale lumii, spre deosebire de aplicația „Gestiune sală” care este o aplicație folosită doar local.

DashPlatform este o aplicație care gestionează mai multe activități sportive. Se împarte în 3 domenii:

- „Park & Recreation software”
- „Sports facility software”
- „Camp and classes software”.

Primele două domenii reprezintă o asemănare cu aplicația propusă, „Managementul unui complex sportiv”, datorită faptului că gestionează activități sportive practicate cu scopul de a te recrea, spre deosebire de „Camp and classes software” care este contruită pentru activitățile sportive în echipe, având la dispoziție un antrenor cu scopul de a face performantă.

Facând o comparație cu aplicația descrisă precedent, aplicația “DashPlatform” nu are o interfață atât de “user-friendly”, fiind greu de utilizat.

Aplicația DashPlatform este o soluție software, de încredere, all-in-one pentru parcuri și activități de recreere care dispune următoarele funcționalități:

- Membership: gestionarea vizitelor la sală, carduri de membru pentru fitness, familii și diferite grupuri. Cardurile pot să fie fizice sau digitale.
- Rezervări și închirieri: gestionarea resurselor, echipamentelor și facilităților.
- Gestionarea activităților: oferă conținut securizat care poate fi vizualizat pe tablete, telefoane și desktopuri. Permite accesul la diferite pagini în funcție de rolurile utilizatorilor.
- Înregistrarea la activitățile disponibile: oferă un set bogat de opțiuni de înregistrare pentru activitățile disponibile.
- Rapoarte: peste 60 de rapoarte standard și personalizate.
- Marketing: trimite mementouri automate pentru activitățile de recreere, email-uri sau mesaje text.
- Plata online: se pot plăti online activitățile sau comisioanele pentru membrii unui club, pentru unul sau mai multe conturi, utilizând reduceri și planuri de plată.
- Conturi clienți: se pot crea sau actualiza profiluri, adăuga membri de familie în conturile unui client și se pot vizualiza înregistrările anterioare.
- Înregistrarea clienților: clienții pot căuta cu ușurință diferite activități de recreere pentru care prezintă interes, folosind opțiunea “Program Finder” și se pot înregistra la una sau mai multe activități selectate.
- Accesul de pe mobil: aplicația mobilă PlatformDash permite clienților să acceseze conturile lor și să se înregistreze cu ușurință pentru diferite activități de recreere în timp ce se află în mișcare.

3.3.3. Aplicația “Virtuagym”

<https://virtuagym.com/>

Aplicația “Virtuagym” este soluția unică care permite trainerilor, cluburilor sportive și antrenorilor să gestioneze operațiunile zilnice ale cluburilor prin intermediul platformei de management online. Inițial a fost o platformă care oferea planuri nutriționale și de antrenament, care ulterior au fost combinate cu servicii de management și programare oferite trainerilor.

Cu Virtuagym, profesioniștii din domeniul fitness se pot concentra pe coaching-ul de fitness eficient, în timp ce conduc o afacere de succes. Aplicațiile personalizate de marca Virtuagym și comunitatea online reprezintă o modalitate perfectă de a promova brandul unui club sau al unui trainer și de a stimula implicarea clienților.

Opțiunile flexibile de aderare și integrare hardware oferă oportunități ample pentru o creștere rapidă și susținută a veniturilor. Odată cu aceasta, Virtuagym le permite companiilor să accepte plăți sub diferite forme, cum ar fi cardurile de numerar sau de debit direct, cu moduri automate de plată online. Acest lucru se adaugă la ușurința cu care membrii pot folosi Virtuagym.

Virtuagym a modernizat modul prin care cluburile de fitness ajung la membrii lor. În primul rând, oferă o mare asistență în gestionarea membrilor, membrilor activi și inactivi. Dashboardul oferă întregul raport care poate ajuta întreprinderile să-și definească modul de acțiune.

Permite o vizibilitate deosebită managerilor în aproape toate procedurile manageriale, financiare și operaționale ale afacerii lor. De asemenea, acesta dispune de un sistem de check-in automatizat, care poate fi utilizat de către companii pentru a monitoriza și urmări accesul membrului la site-ul web și la o aplicație integrată cu capabilități de scanare cu carduri RDIF. Acesta poate fi un instrument excelent pentru eficientizarea intrării și ieșirii membrilor către pagina de checkout, indicând cât de bine funcționează afacerea în diferite momente ale unei zile.

Aplicația Virtuagym, oferă soluții pentru un club de fitness, pe lângă alte soluții (Personal Trainers, Corporate Wellness, Fitness Chains etc.). Aplicația care ofera soluții pentru un club de fitness dispune de următoarele funcționalități:

- Automatizează procesele de business, de la programarea cursurilor la facturarea lor, ducând la eliberarea timpului angajaților și totodată a clienților.
- Managementul membership-urilor. Gestionează toate informațiile membrilor clubului și le stochează în siguranță pe cloud.
- Plata și facturarea online. Facturile sunt generate automat și trimise membrilor care pot opta pentru modalitatea de plată, ex. Debit direct, card de credit, PayPal etc.
- Controlul accesului. Există diferite metode pentru accesul în incinta complexului sportive, benzi magnetice, coduri de bare, RFID și amprente digital

În următorul tabel este schițată o comparație a sistemelor similar descrise mai sus

Funcționalitate /Sistem	Gestiune Sala	DashPlatform	Virtuagym	Sistemul propus
Aplicație web	X	x	x	x
Aplicație desktop	Nu are	Nu are	x	
Aplicație mobile	Nu are	x	x	
Dashboard	x	Nu are	x	x
Adăugare/Ștergere/ModificareProgramări	x	x	x	x
Adăugare/Ștergere/utilizatori	x	x	x	x
Adăugare abonamente	x	x	x	x
Membership		x	x	
Plată online	x	x	x	
Card de membru		x	x	
Trimitere notificări	x			x
Vizualizare comezi	x	x	x	x
Vizualizare abonamente active	x	Nu are	Nu are	
Orarul activităților	Nu are	Nu are	Nu are	x
Promoții	x	Nu are	Nu are	
Rezervări	Nu are	x	Nu are	
Rapoarte	Nu are	x	Nu are	x
Marketing	Nu are	x	Nu are	
Acces pe baza de card RFID	Nu are	Nu are	x	

Tabel 3.1 Comparăția sistemelor similare

Având în vedere comparația sistemelor similare făcută în tabelul de mai sus, Putem trage concluzia, că fiecare sistem are funcționalitățile lui specifice în funcție de zona în care se află și de cerințele clienților fiecărui complex sportiv.

Luând în considerare toate funcționalitățile acestor sisteme, sistemul ce urmează a fi dezvoltat își propune să cuprindă cele mai importante funcționalități ale unui complex sportiv, și să ofere clienților cât mai multe motive să folosească aplicația, oferindu-le o interfață intuitivă și o aplicație care o să le ușureze procesul de zi cu zi petrecut în incinta complexului.

Capitolul 4. Analiză și Fundamentare Teoretică

În acest capitol vor fi prezentate tehnologiile care au contribuit la implementarea aplicației web. Printre acestea se numără: SpringBoot Framework, Maven, Hibernate cu JavaPersistence, JavaServer Pages și baza de date pentru s-a folosit Microsoft SQL Server Management Studio.

4.1. Tehnologii și concept utilizate pentru dezvoltarea aplicație web

4.1.1. Java EE

Java EE este un framework de dezvoltare a aplicațiilor, cel mai popular standard pentru construirea și implementarea aplicațiilor web în Java. Două dintre tehnologiile cheie care stau la baza construirii componentelor web ale aplicațiilor Java EE sunt servlet-urile și JSP-urile. Servlet-urile sunt clase Java care oferă punctul de intrare în logica pentru gestionarea request-urilor web și pentru returnării unei reprezentări Java a răspunsului web. JSP reprezintă un mecanism de combinare a codului HTML cu logica scrisă în Java. După ce sunt compilate și implementate, JSP-urile rulează ca un servlet, unde, de asemenea, ia un web request și returnează un obiect Java care reprezintă pagina de răspuns. În mod tipic, într-un proiect mare, JSP și servlet-urile fac parte din layer-ul de prezentare al aplicației și includ logica pentru a invoca metodele de business de nivel superior. Funcțiile principale de business sunt separate într-un set clar definit de interfețe, astfel încât aceste componente să poată fi utilizate și modificate.

Am folosit Java EE pentru implementarea aplicației datorită faptului că pune la dispoziție o serie de avantaje administrative (GUI pentru gestionarea bazelor de date, logare, monitorizare, tranzacții etc.) . Cu ajutorul altor framework-uri aceste configurări ar trebui făcute manual.

4.1.2. Spring boot framework

Spring Boot folosește un model de dezvoltare complet nou pentru a ușura dezvoltarea programelor folosind Java, evitând niște pași de dezvoltare și configurație. Principalul obiectiv al acestui framework este de a reduce timpul de dezvoltare, de testare precum și de a ușura dezvoltarea aplicațiilor web spre deosebire de Spring framework care necesită mai mult timp.

SpringBoot poate fi descris, pe scurt, ca în figura de mai jos:



Figura 4.1 Structura framework-ului Spring Boot

Am folosit acest framework în dezvoltarea aplicației propuse datorită faptului că evită complet configurările XML, evită definirea mai multor configurații de adnotări, combinarea diferitelor adnotări existente în Spring cu o adnotare simplă și unică, evitarea

scrierii unui numar mare de importuri și furnizează unele setări implicite pentru a crea proiecte noi în cel mai scurt timp.

Ca un scurt rezumat a ceea ce oferă acest framework, există patru “trucuri” de baza pe care o sa le folosim în dezvoltarea sistemului propus.

- Configurarea automată: poate furniza automat configurări pentru funcționalitățile comune ale aplicației la mai multe aplicații Spring.
- Dependențe de început: se asigura ca toate librăriile necesare sunt adăugate
- Interfața “command-line” (CLI- Command Line Interface)
- Actuator

4.1.3. Maven

Apache Maven este un tool pentru managementul proiectelor și pentru înțelegere, bazat pe conceptul de POM (Project Object Model). Maven poate gestiona construirea, raportarea și documentarea unui proiect dintr-o informație centrală.

Maven abordează două aspecte legate de construirea software-ului: în primul rând, descrie modul în care software-ul este construit și, în al doilea rând, descrie dependențele acestuia.

Am ales Maven pentru dezvoltarea sistemului propus datorită faptului că acesta urmează un set de standarde, include un “life-cycle” al proiectului, un sistem de gestionare a dependențelor și o logică pentru executarea obiectivelor plugin-ului la diferite faze dintr-un “life-cycle”. Oferă configurație simplă a proiectului, care utilizează cele mai bune practici, proiectele urmând o structură consistentă. Prin impunerea unei structuri consecvente, face mai ușoare modificările în viitor, când noi dezvoltatori sunt introduși în proiect. De asemenea, se asigură că programatorii primesc întotdeauna cea mai recentă versiune de compilatoare .

Majoritatea proiectelor Java se bazează pe alte proiecte și framework-uri open source, pentru a funcționa corect. Maven oferă o modalitate convenabilă de a declara aceste dependențe de proiect, într-un fișier separat, extern POM.XML, descarcă automat aceste dependențe și permite utilizarea lor în proiect . Acest lucru simplifică foarte mult managementul dependențelor de proiect.

4.1.4. Java Persistence with Hibernate

Hibernate este o soluție ORM (Object Relational Mapping) pentru Java, mapează clasele Java la tabelele din baza de date, tipurile de date din Java la tipurile de date din SQL și reduce sarcinile programatorului legate de persistența datelor, cu 95%.

Maparea claselor Java la tabelele din baze de date este implementată prin configurarea unui fișier XML sau prin utilizarea adnotărilor Java. Când se utilizează un fișier XML, Hibernate poate genera cod sursă “skeleton” pentru clasele de persistență. Hibernate poate folosi fișierul XML sau adnotările Java pentru a menține schema bazei de date. Acesta furnizează facilități pentru a organiza relațiile “one-to-many” și “many-to-one” dintre clase . În plus, față de gestionarea asocierilor între obiecte, Hibernate poate gestiona asocieri reflexive în care un obiect are o relație “one-to-many” cu alte instanțe ale aceluși tip de clasă.

Am ales să folosesc Hibernate, datorită faptului că acesta acceptă maparea tipurilor de valori personalizate. Acest lucru face posibile următoarele scenarii:

- Înlocuirea tipului SQL implicit la maparea unei coloane către o proprietate
- Maparea Java Enum-urilor la coloane ca și cum ele ar fi proprietăți obișnuite.
- Maparea unei singure proprietăți la mai multe coloane.

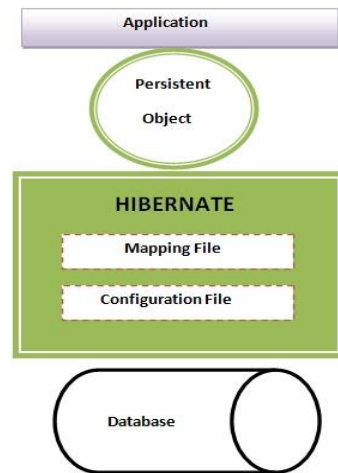


Figura 4.2 Arhitectura Hibernate

JPA (Java Persistence API) este doar o specificație care descrie interfețele cu care clientul operează și metadatele standard de mapare "object-relational" (adnotări Java sau descriptori XML). Dincolo de definiția API, JPA explică modul în care aceste specificații ar trebui să fie implementate.

Hibernate era deja o implementare full-featured Java ORM până în momentul în care specificația JPA a fost lansată pentru prima dată. Deși implementează specificația JPA, Hibernate își păstrează API-ul nativ atât pentru compatibilitate "backward", cât și pentru a acoperi caracteristicile non-standard. Chiar dacă este cel mai bine să adere la standardul JPA, în realitate, mulți furnizori JPA oferă funcții suplimentare care vizează cerințele de înaltă performanță la nivel data-access.

JPA se ocupă de modul în care obiectele Java ("entități persistente") sunt stocate în baza de date relațională, cum pot fi accesate și modul în care starea obiectului poate fi stocată astfel încât să poată fi restaurată la reluarea aplicației.

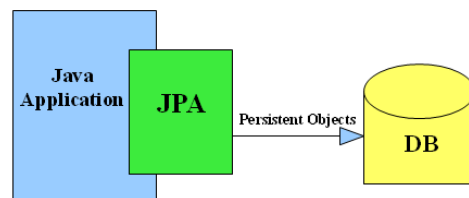


Figura 4.3 Java Persistence API

4.1.5. JavaServer Pages

Tehnologia JavaServer Pages (JSP) permite amestecarea unui cod HTML static, cu conținut generat dynamic.

Am folosit JSP pentru dezvoltarea paginilor web, datorită faptului ca în aceste fișiere JSP se pot include orice fragmente valide de cod Java și se pot amesteca cu fragmentele de cod HTML. În fișierul JSP, codul Java este marcat cu `<% și %>`. La implementare (sau uneori primul request către pagina, în funcție de configurație), JSP este compilat într-o clasă de servlet. Acest proces combină codul HTML și scripturile în fișierul JSP în metoda `_jspService` care populează variabila `HttpServletResponse`. Combinarea multor coduri Java complexe cu HTML poate duce la un fișier JSP complicat. O modalitate pentru a evita această situație este de a folosi bibliotecile personalizate ale JSP-urilor. Biblioteca de etichete cea mai răspândită este JavaServer Pages Standard Tag Library (JSTL), care oferă tag-uri pentru a gestiona operațiile simple necesare în majoritatea sarcinilor de programare JSP, inclusiv looping, globalizarea, manipularea XML și chiar procesarea seturilor de rezultate SQL.

Putem să spunem ca servlet-urile reprezintă cod Java în care include HTML, iar jsp-urile reprezintă cod HTML în care includem fragment de cod Java. În ciuda diferențelor aparente mari dintre paginile JSP și servlets, putem spune că ele sunt același lucru, datorită faptului ca paginile JSP sunt traduse în servleturi, servlet-urile sunt compilate, iar la un request se execută servlet-urile compilate. Deci, scrierea paginilor JSP este într-adevăr doar o altă modalitate de a scrie servlet-uri.

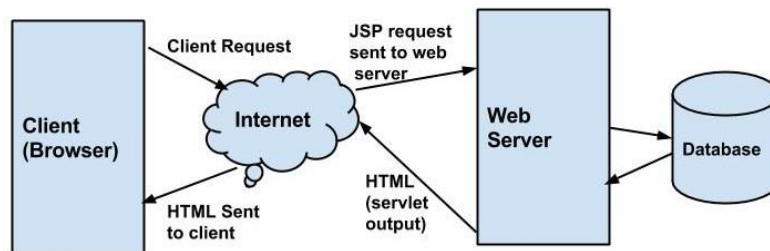


Figura 4.4 Arhitectura JSP

Paginile JSP sunt traduse în servlets. Deci, în mod fundamental, toate sarcinile JSP care pot fi executate pot fi realizate și prin servlets. Cu toate acestea, această echivalență de bază nu asigură faptul că servlet-urile și paginile JSP sunt la fel de adecvate în toate scenariile.

Problema nu este puterea tehnologiei, ci confortul, productivitatea și mentenabilitatea unuia sau a celuilalt. JSP oferă următoarele avantaje numai față de servlets:

- Este mai ușor să scriem și să menținem codul HTML. Codul static este HTML obișnuit: nu există backslash-uri uplimentare, nu există ghilimele duble și nu există sintaxă Java ascunsă.
- Puteți utiliza instrumentele standard de dezvoltare a site-urilor Web.

- Se poate împărți munca pe echipe. Programatorii pot lucra la codul dinamic iar dezvoltatorii web se pot concentra pe presentation layer. Pe proiecte mari, această organizare este foarte importantă. În funcție de dimensiunea echipei și complexitatea proiectului, se poate impune o mai slabă sau mai puternică separare între codul HTML static și conținutul dinamic.

4.1.6. RESTful Web Services

Un serviciu web (Web Service) este o aplicație web de tip client-server, în care un server furnizor de servicii (numit “Service Endpoint”) este accesibil unor aplicații client pe baza adresei URL a serviciului. Principalul merit al serviciilor web este acela că asigură interoperabilitatea unor aplicații software implementate pe platforme diferite și cu instrumente diferite. În același timp, aplicațiile sunt slab cuplate (“loosely coupled”), adică mesajele schimbate sunt standard (“self-contained”).

Diferența dintre o aplicație web clasică și un serviciu web constă în principal în formatul documentelor primite de client și modul în care acestea sunt folosite: într-o aplicație web, clientul primește documente HTML transformate de un browser în pagini afișate, iar clientul unui serviciu web primește un document XML / JSON folosit de aplicația client, dar care nu se afișează direct pe ecran.

Servicii SOAP (Simple Object Access Protocol)

SOAP este un protocol bazat pe XML (Extensible Markup Language) ce realizează schimbul de informații între calculatoare, într-un mediu distribuit. Acest protocol este alcătuit din trei componente:

- Un înveliș care definește conținutul unui mesaj și cum se procesează acesta
- Un set de reguli pentru codificare instanțelor definite de aplicație
- convenție pentru reprezentarea apelurilor și răspunsurilor procedurilor apelate la distanță

SOAP este independent de platformă și limbaj, iar astfel reprezintă o cale de comunicație între sisteme de operare diferite care au limbaje de operare diferite. Mesajele SOAP sunt codificate în documente XML, fiind alcătuite din SOAP Envelope, SOAP Body și SOAP Header.

Acest protocol poate fi unul destul de lent din cauza codificării informației folosind fișiere XML. Folosind această codificare, există un număr suplimentar de informații care trebuie transmise între capetele de comunicare.

Aceste servicii sunt descrise cu ajutorul unor documente WSDL (Web Service Description Language) sau Limbajul de Descriere a Serviciilor Web care descrie interfețele serviciilor web.

În următoarea figură, este prezentat fluxul trimerii unei cereri folosind serviciile SOAP.

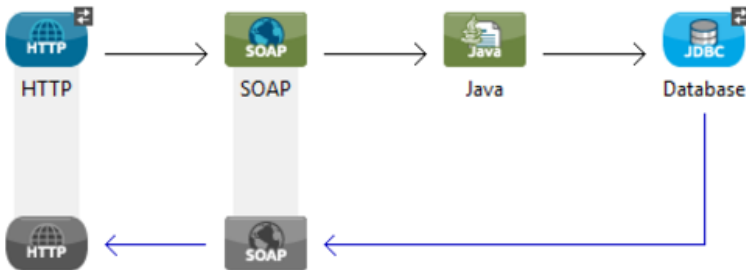


Figura 4.5 Fluxul transmiterii unei cerei SOAP

Servicii REST (Representational State Transfer)

REST reprezintă un model arhitectural pentru crearea serviciilor web. Acesta descrie o arhitectură orientată pe resurse. Motivul realizării unei astfel de arhitecturi a pornit de la faptul că serviciile de tip RPC și cele care folosesc SOAP aduc o oarecare complexitate. Astfel, simplitatea web a reprezentat sursa de inspirație a serviciilor web de tip REST. Serviciile web de tip REST folosesc toate componentele ce au făcut web-ul un mare succes. Deși nu este un standard, el se folosește de următoarele standarde:

- HTTP (Hypertext Transfer Protocol)
- URI (Uniform Resource Identifier)
- XML / HTML / GIF / JPEG

REST este un set de reguli la care o arhitectură ar trebui să se conformeze, astfel putem spune ca este un stil arhitectural.

“Intefăța uniformă” este principiul de bază al REST. HTTP furnizează patru metode de bază ce descriu operațiile cele mai comune:

- Crearea unei noi resurse: HTTP POST
- Extragerea unei resurse: HTTP GET
- Actualizarea unei resurse existente: HTTP PUT
- Ștergerea unei resurse: HTTP DELETE

Toate aceste operații sunt suficiente pentru a realiza o arhitectură de tip REST.

Următoarea figură, Fig x.x., reprezintă fluxul trimiterii unei cereri folosind serviciile REST.

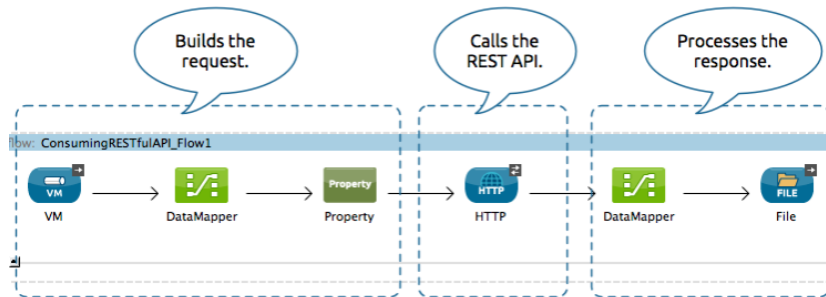


Figura 4.6 Fluxul transmiterii unei cereri REST

Comparație SOAP vs. REST

Fiecare protocol are avantajele și dezavantajele sale. În continuare, se vor prezenta anumite avantaje și dezavantaje pentru fiecare tip de serviciu.

Avantajele folosirii serviciilor REST sunt prezentate în continuare:

- Este mai simplu și are o documentație mai ușor de înțeles decât SOAP
- Permite mai multe formate de date, iar SOAP permite doar XML
- În general, folosește JSON, acesta fiind procesat mult mai rapid, lucru care duce la o navigare mai rapidă pentru client
- Are o performanță și o scalabilitate mai bună
- Serviciile REST sunt folosite de către mari companii, cum ar fi: Yahoo, Flickr, Amazon, Ebay, etc. Google folosea servicii SOAP până în anul 2006, când au început să folosească arhitectura REST.

Avantajele serviciilor SOAP sunt prezentate în continuare:

- Suportă WS-Security care oferă o anumită securitate
- Are o logică de “successful/retry” integrată (WS-ReliableMessaging), pe când REST nu are un sistem standard de trimitere a mesajelor și clienții trebuie să se confrunte cu eșecurile comunicației prin reîncercări.
- Oferă suport pentru tranzacții prin două standarde: WS-AtomicTransactions și WS-BusinessActivity

În concluzie, pentru sistemul implementat, cea mai potrivită alegere este arhitectura REST datorită numărului mare de avantaje ale acesteia

4.1.7. Baza de date

O bază de date reprezintă o modalitate de stocare a unor informații și date pe un suport extern (un dispozitiv de stocare), cu posibilitatea extinderii ușoare și a regăsirii rapide a acestora.

De obicei o bază de date este memorată într-unul sau mai multe fișiere. Bazele de date sunt manipulate cu ajutorul sistemelor de gestiune a bazelor de date.

Cel mai răspândit tip de baze de date este cel relațional, în care datele sunt memorate în tabele. Pe lângă tabele, o bază de date relațională mai poate conține: indecși, proceduri stocate, declanșatori, utilizatori și grupuri de utilizatori, tipuri de date, mecanisme de securitate și de gestiune a tranzacțiilor etc.

4.1.7.1. Microsoft SQL Server

Microsoft SQL Server este un sistem de management al bazelor de date relaționale, sau RDBMS, care suportă o mare varietate de aplicații de procesare a tranzacțiilor, de business intelligence și de analiză în mediile IT corporative.

Microsoft SQL Server este un sistem de management al bazelor de date relaționale, sau RDBMS, care suportă o mare varietate de aplicații de procesare a tranzacțiilor, de business intelligence și de analiză în mediile IT corporative.

Ca și alte programe RDBMS, Microsoft SQL Server este construit pe lângă SQL, un limbaj de programare standardizat pe care administratorii de baze de date (DBA) și alți profesioniști IT le folosesc pentru a gestiona bazele de date și pentru a interoga datele

pe care le conțin. SQL Server este legat de Transact-SQL (T-SQL), o implementare a SQL de la Microsoft care adaugă un set de extensii de programare la limbajul standard.

Arhitectura MS SQL Server este împărțită în trei componente:

- SQLOS care implementează serviciile de bază necesare serverului SQL, incluzând managementul thread-urilor, al memoriei și a I/O
- Motorul relațional (relational engine), care implementează componentele relaționale ale bazei de date, incluzând suport pentru bazele de date, tabele, interogări și proceduri stocate
- Protocol layer care expune funcționalitatea serverului SQL.

4.2. Cerințele sistemului

Cerințele sistemului reprezintă descrierile la nivel înalt referitoare la anumite servicii de sistem, constrângeri sau la o specificație detaliată care este generată în timpul procesului de colectare a cerințelor. Acestea sunt împărțite în două categorii:

- Cerințe funcționale: descriu serviciile sistemului, modul în care sistemul ar trebui să răspundă la anumite date de intrare și modul în care sistemul ar trebui să se comporte în situații concrete;
- Cerințe non-funcționale: descriu atributele sistemului, specifică criteriile care pot fi folosite pentru a analiza funcționare unui sistem în anumite condiții;

4.2.1. Cerințe funcționale

Cerințele funcționale reprezintă principalele lucruri la care un user se așteaptă de la un sistem software. Cerințele funcționale ale sistemului propus sunt următoarele:

- Crearea unui cont de utilizator pentru clienți sau logarea folosind gmail.
- Progrămarile se fac pe baza unui cont de utilizator.
- Afișarea unui calendar prin care utilizatorul poate să vizualizeze sporturile disponibile în fiecare zi și intervalul orar
- Utilizatorul poate să facă o programare la activitatea sportivă dorită în funcție de disponibilitatea locurilor.
- Programările se pot face pe anumite perioade de timp, în funcție de cerințele utilizatorului (1 săptămână/1 lună/plată pe activitate sportivă)
- Posibilitatea de a vizualiza utilizatorii deja înscrși pentru sporturile de echipă
- Utilizatorii pot anula programarea cu o perioadă de timp(definită) înainte de începerea activității sportive, în caz contrar utilizatorii pot fi penalizați.
- Vizualizarea unor galerii foto pentru fiecare activitate sportivă.
- Vizualizarea unei descrieri pentru fiecare activitate sportivă, cu scopul de a informa utilizatorul despre activitatea respectivă.
- Achiziționarea abonamentelor online, printr-un sistem de plată online, primirea dovezii și confirmarea plății prin email.
- Crearea, actualizarea și ștergerea profilului de utilizator de către admin sau angajați.
- Utilizatorii au posibilitatea de a da rating activităților sportive la care au participat.

- Admin ul poate crea/sterge/updata conturile angajaților.

Identificator	Descrierea cerinței funcționale	Utilizator care beneficiază de funcționalitate
CF 1	Autentificare	Admin, Angajat, Client
CF 2.1	Administrare conturi angajați	Admin
CF 2.2	Administrare conturi clienți	Angajați, Admin
CF 2.3	Administrare conturi admin	Admin
CF 3.1	Crearea unei programări	Admin, Angajat, Client
CF 3.2	Ștergerea unei programări	Admin, Angajat, Client
CF 3.3	Editarea unei programări	Admin, Angajat, Client
CF 4	Vizualizarea galerii foto	Toți utilizatorii
CF 5	Vizualizarea descrierilor	Toți utilizatorii
CF 6	Vizualizarea abonamentelor posibile	Admin, Angajat, Client
CF 7	Comandarea abonamentelor online	Client
CF 8	Creare cont utilizator	Toți utilizatorii
CF 9	Generare rapoarte	Admin, Angajat
CF 10	Trimite notificări	Admin, Angajat
CF 11	Vizualizare DashBoard	Toți utilizatorii
CF 12	Vizualizare calendar	Toți utilizatorii

4.2.2. Cerințe non-funcționale

Cerințele non-funcționale descriu cum funcționează sistemul. Acestea acoperă aproape toate cerințele pe care cerințele funcționale nu le acoperă.

Utilizabilitatea reprezintă ușurința cu care un sistem este utilizat și învățat, astfel sistemul propus își dorește ca timpul în care un client își face programare, comandă un abonament sau vizualizează detalii referitoare la activitățile sportive pentru care prezintă interes, să fie relativ mic. De asemenea, timpul în care un angajat face diferite modificări sau adaugări sa fie relativ mic.

Utilizabilitatea sistemului face referire și la design-ul sistemului. Modul în care elementele sunt așezate pe pagină reprezintă un criteriu important, design-ul trebuie sa fie intuitiv, astfel încât user-ul să poată folosi aplicația fără prea mult efort. Chiar dacă o interfață bine organizată nu ține locul unor funcționalități importante, în cele mai multe cazuri o aplicație care are o interfață „user-friendly” și care cât de cât cuprinde

funcționalitățile de bază, este mult mai apreciată de client, decât o aplicație a cărei interfețe lasă de dorit.

Utilizabilitatea este strâns legată și de eficiența cu care utilizatorii dispun de resursele sistemului.

Disponibilitatea reprezintă aptitudinea sistemului de a fi în stare de funcționare în anumite condiții date, la un moment dat sau un interval de timp, presupunând ca furnizarea mijloacelor externe este asigurată.

De asemenea, poate fi exprimată în termeni de durată medie de întrerupere pe săptămână, lună sau an sau ca durată totală de întrerupere pentru o anumită săptămână, lună sau an. Uneori, disponibilitatea este exprimată în termeni calitativi, indicând măsura în care un sistem poate continua să funcționeze atunci când o componentă semnificativă sau un set de componente scade

Tinând cont că sistemul propus trebuie să înlocuiască atât munca angajaților cât și să ofere suport pentru efectuarea unei rezervări sau comandarea unui abonament online, sistemul trebuie să fie disponibil utilizatorilor 24 din 24 de ore.

Performanța unui sistem se rezumă în cele mai multe cazuri la timpul maxim de răspuns care este pus la dispoziție sistemului pentru a oferi feedbackul necesar în urma unei acțiuni declanșate de utilizator. Depinzând de context, performanța poate implica una sau mai multe din următoarele aspecte:

- Timp de răspuns mic pentru un request
- Rata de procesare a informației
- Utilizarea redusă a resurselor calculatorului
- Compresarea/decompresarea sau codificare/decodificarea rapidă a datelor
- Timp scurt de transmitere a datelor

Sistemul trebuie să fie interactiv, iar întârzierile trebuie să fie cât mai puțin. Astfel, în fiecare "acțiune-răspuns" a sistemului, nu există întârzieri imediate. În cazul deschiderii pop-up-urilor, al mesajelor de eroare de afișare și al salvării setărilor sau sesiunilor există o întârziere cu mult mai mică de 2 secunde.

Securitatea acestui sistem a fost dezvoltată prin utilizarea framework-ului „Spring Security”.

Spring Security se focusează pe furnizarea serviciilor de autentificare și autorizare pentru aplicațiile dezvoltate în Java.

Autentificarea în cadrul acestui sistem, se realizează printr-o metoda de login, care verifică credențialele furnizate de utilizator cu cele din baza de date de pe sistemul de operare local. Dacă credențialele corespund, procesul este finalizat și utilizatorului i se acordă autorizație de acces.

Autorizarea este procesul prin care un administrator acordă drepturi și verifică permisiunile de acces la resurse ale unui utilizator. Privilegiile și preferințele acordate pentru un cont autorizat depind de permisiunile utilizatorului, care sunt stocate local. În sistemul propus, există trei tipuri de utilizatori, caracterizați pe baza rolurilor: Role_Admin, Role_Client, Role_Angajat. În funcție de rolul pe care îl are utilizatorul logat, el o să fie redirecționat pe o anumită pagină pe care are drepturi de acces.

Mentenabilitatea este definită ca ușurința cu care se pot face modificări unui sistem software. Aceste modificări pot fi necesare pentru corectarea defecțiunilor, adaptarea sistemului la îndeplinirea unei noi cerințe, adăugarea de noi funcționalități, eliminarea funcționalității existente sau corectarea atunci când apar erori sau a deficiențe care pot fi perfecționate, adaptate sau acțiuni luate pentru a reduce în costurile de întreținere.

În sistemul propus mentenabilitatea este reprezentată de ușurință prin care anumite părți ale sistemului pot fi eliminate fără a afecta restul funcționalităților. Spre exemplu, plata abonamentelor se poate elimina fără a afecta funcționalitatea corespunzătoare programărilor.

4.3. Cazuri de utilizare

Cazurile de utilizare au sarcina de a identifica, clarifica și organiza cerințele sistemului. Acestea sunt alcătuite dintr-un set de posibile secvențe de interacțiuni între sistem și utilizatori, într-un anumit mediu și legate de un anumit scop. Cazul de utilizare ar trebui să conțină toate activitățile sistemului care au o semnificație pentru utilizatori. Un caz de utilizare poate fi considerat ca o colecție de scenarii posibile legate de un anumit scop, într-adevăr, cazul de utilizare și scopul pot uneori considerate a fi sinonime.

Un caz de utilizare sau un set de cazuri, au următoarele caracteristici:

- Organizarea cerințelor funcționale
- Modelarea interacțiunilor dintre obiectivele sistemului și utilizatori (actori)
- Înregistrează căile de la declanșarea unui eveniment, la atingerea obiectivelor.
- Descrie un flux principal de evenimente, numit și “curs de bază” și eventual alte fluxuri excepționale de evenimente numite și “cursuri de acțiune alternative”
- Este “multi-level”, un caz de utilizare poate folosi funcționalitatea altuia.

4.3.1. Actorii sistemului

Tipurile de utilizatori ale sistemului sunt următoarele:

Administratorul: are acces la toate funcționalitățile sistemului;

Vizitatorul: are acces la pagina de “home” de unde are posibilitatea să se înregistreze, și la galeria foto a aplicației unde sunt descrise sporturile care pot fi practicate în cadrul complexului sportive;

Clientul: vizitatorul care și-a creat un cont. El are acces la pagina de programări și poate să-și comande un abonament online.

Angajatul: poate să gestioneze abonamentele, programările, conturile utilizatorilor cât și să trimită mail-uri clienților.

În următoarele figuri o să fie prezentate diagramele „use-case” pentru fiecare dintre utilizatorii sistemului descriși mai sus pentru a oferi o idee generală a modului în care va fi utilizat sistemul, pentru a furniza o privire de ansamblu a funcționalităților ce se doresc a fi oferite de sistemul propus și pentru descrierea modului în care sistemul interacționează cu diferite tipuri de actori.

În figura următoare sunt prezentate toate cazurile de utilizare, în care actorul principal este administratorul sistemului.

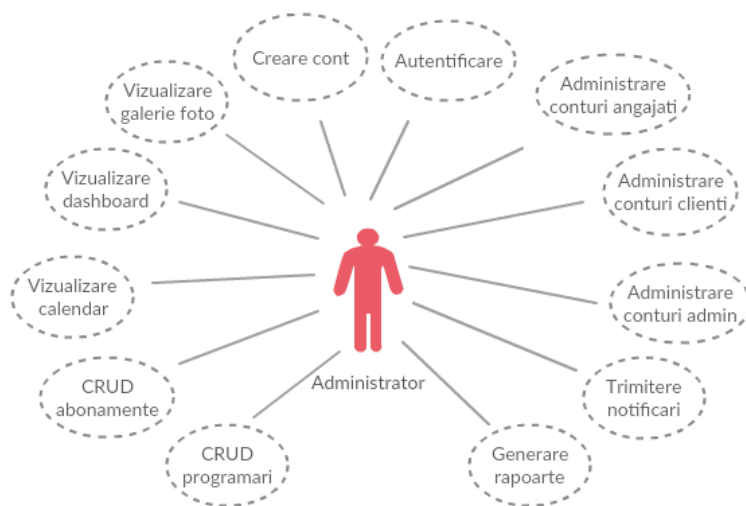


Figura 4.7 Cazuri de utilizare pentru administrator

În figura următoare sunt prezentate toate cazurile de utilizare, în care actorul principal este angajatul sistemului.

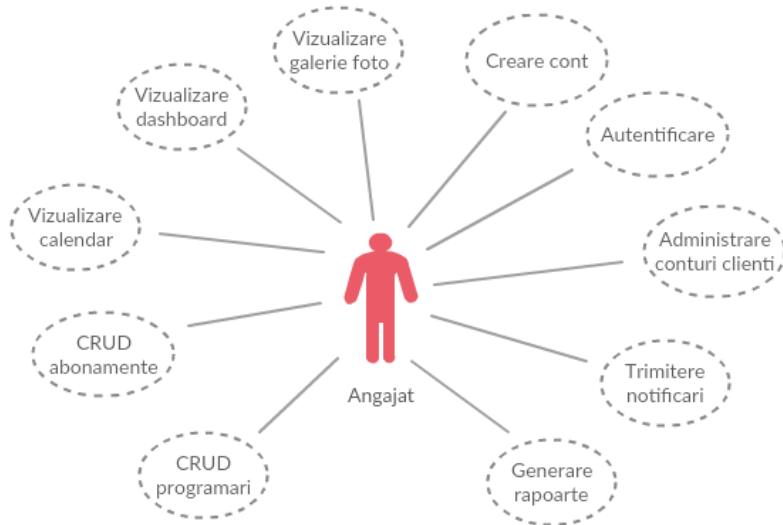


Figura 4.8 Cazuri de utilizare pentru angajat

În figura următoare sunt prezentate toate cazurile de utilizare, în care actorul principal este clientul sistemului.

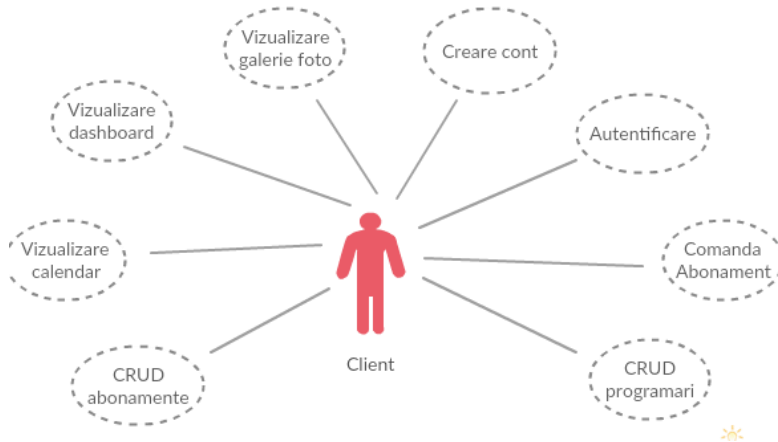


Figura 4.9 Cazuri de utilizare pentru client

În figura următoare sunt prezentate toate cazurile de utilizare, în care actorul principal este vizitatorul sistemului.

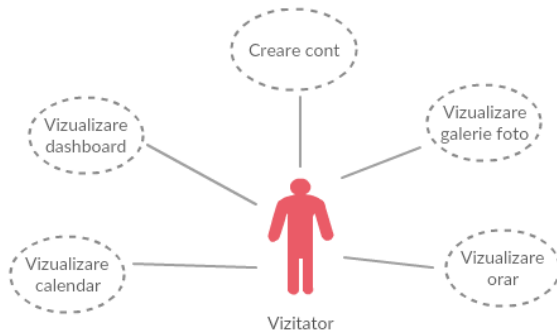


Figura 4.10 Cazuri de utilizare pentru vizitator

4.3.2. Descrierea detaliată a cazurilor de utilizare

În acest subcapitol sunt reprezentate așa numitele diagrame “flowchart” folosite pentru a descrie funcționalitățile unor anumite componente separate ale sistemului.

CU1 (Caz de utilizare 1)

Numele cazului de utilizare: Adăugarea unei programări

Actor principal: Clientul

Părți interesate :

Clientul: dorește să poată adăuga o programare la unul dintre sporturile disponibile în cadrul complexului sportive.

Administratorul: dorește să gestioneze programările adăugate de clienți, să trimită mail-uri de confirmare a programării și să genereze diferite rapoarte, cum ar fi numărul de programări create în ziua respectivă.

Angajatul: dorește să aibă acces la toate programările adăugate de clienți.

Precondiții: Clientul trebuie să aibă un cont creat și să fie logat în sistem în momentul adăugării unei programări

Postcondiții: După adăugarea unei programări, programarea trebuie să se stocheze în baza de date pentru a se ține evidența tuturor programărilor și să se evite cazul în care două programări se pot suprapune.

Scenariul de succes :

- Clientul accesează pagina de “Home”.
- Clientul se loghează în aplicație.
- Clientul accesează orarul
- Clientul adaugă o programare pentru o activitate sportivă aleasă
- Clientul primește pe mail notificarea pentru adăugarea programării cu detaliile necesare.

Diagrama următoare furnizează o reprezentare vizuală, în detaliu, a evenimentelor declanșate de cazul de utilizare în care actorul “Client” adaugă o programare.

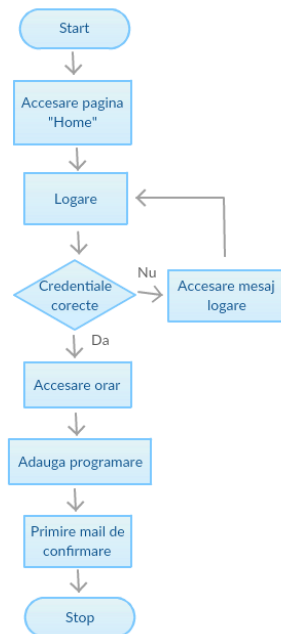


Figura 4.11 Diagramă flow-chart pentru adăugarea unei programări

Diagrama următoare furnizează o reprezentare vizuală, în detaliu, a evenimentelor declanșate de cazul de utilizare în care actorul “Client” comandă un abonament.

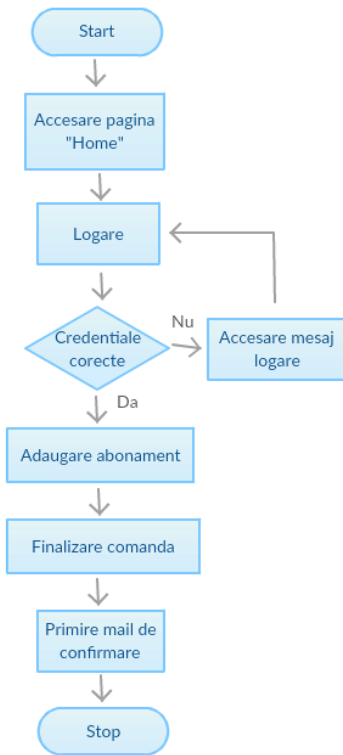


Figura 4.12 Diagramă flow-chart pentru comandarea unui abonament

CU3 (Caz de utilizare 3)

Numele cazului de utilizare: Crearea unui cont

Actor principal: Vizitatorul

Părți interesate:

Vizitatorul: dorește să se înregistreze pentru a putea comanda abonamente și a efectua programări.

Administratorul: dorește să gestioneze conturile tuturor utilizatorilor.

Angajatul: dorește să gestioneze conturile clienților.

Precondiții: Vizitatorul trebuie să se afle pe pagina principală.

Postcondiții: După crearea unui cont, datele utilizatorului trebuie să se stocheze în baza de date pentru a se ține evidența tuturor utilizatorilor sistemului și a se putea efectua ulterior logarea.

Scenariul de succes:

- Clientul accesează pagina de "Home".
- Clientul se selectează opțiune de "Register".
- Clientul își adaugă datele necesare pentru crearea contului.

- Clientul primește pe mail notificarea cu toate detaliile contului creat.

Diagrama următoare reprezintă diagrama de secvență o evenimentelor declanșate de cazul de utilizare în care actorul “Client” comandă un abonament.

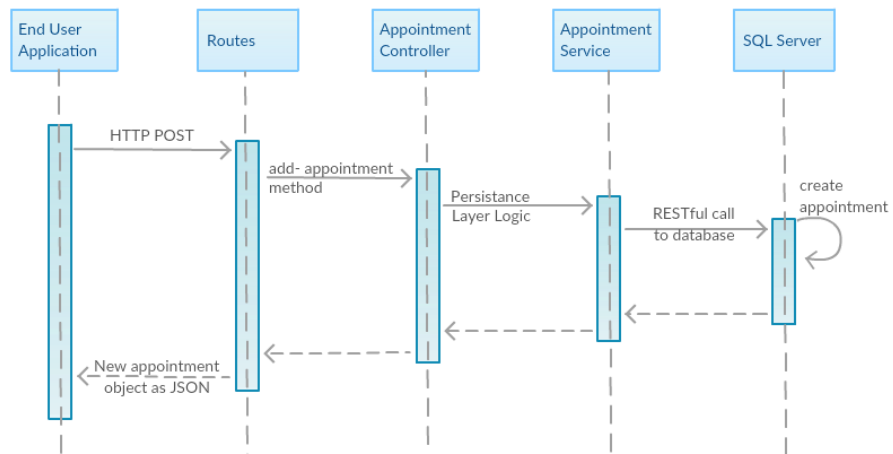


Figura 4.13 Diagramă de secvență pentru adăugarea unei programări

Capitolul 5. Proiectare de Detaliu si Implementare

În acest capitolul o sa fie descrisă comportarea sistemului pentru aplicația web dezvoltată. Va fi prezentată diagrama arhitecturii alese pentru implementarea celor două aplicații, diagrama de dezvoltare, arhitectura bazei de date și structura tabelor și vor fi detaliate componente considerate relevante.

5.1. Arhitectura sistemului

5.1.1. Arhitectura aplicației web

Arhitectura aplicației web dezvoltate respectă modelul arhitecturii pe trei nivele. O arhitectură pe trei nivele este o arhitectură client-server în care logica procesului funcțional, accesul la date, stocarea de date și interfața cu utilizatorul sunt dezvoltate și menținute ca module independente. Arhitectura pe trei niveluri este un model de design software și o arhitectură software bine stabilită.

Arhitectura folosită pentru sistemul propus permite oricărui dintre cele trei niveluri să fie modernizate sau înlocuite independent. Interfața cu utilizatorul este implementată pe un PC, desktop, și utilizează o interfață grafică standard cu diferite module care rulează pe serverul de aplicații. Sistemul de gestionare a bazelor de date relaționale de pe serverul de bază de date conține logica de stocare a datelor computerizate.

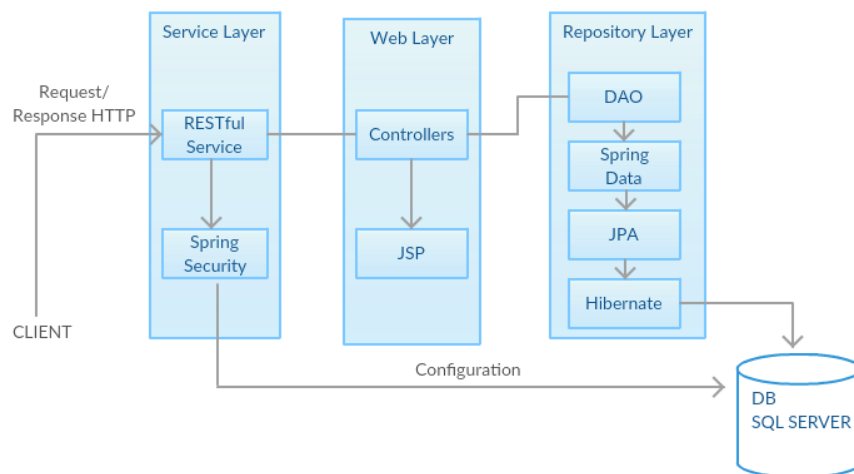


Figura 5.1 Diagrama arhitecturală a sistemului

5.1.1.1. Web Layer

Web layer-ul reprezintă layer-ul superior al sistemului. Acesta răspunde de prelucrarea

datelor introduse de utilizator și de returnarea răspunsului corect către utilizator. Layer-ul web se ocupă, de asemenea, de excepțiile aruncate de celelalte layer-uri. Deoarece acest layer este punctul de intrare al unei aplicații, trebuie să aibă în vedere autentificarea și să acționeze ca o primă linie de apărare împotriva utilizatorilor neautorizați.

În cadrul sistemului propus, acest layer conține paginile JSP (Java Server Page) ale aplicației. De asemenea, acest serviciu pune la dispoziție interfața cu utilizatorul, o interfață de acces pentru datele stocate în baza de date relațională. Această componentă a sistemului interacționează direct cu utilizatorul, oferindu-i posibilitatea de a vizualiza date, de a adăuga date noi sau de a le modifica date existente printr-o interfață interfață prietenoasă, intuitivă și ușor de utilizat.

Paginile JSP sunt împărțite și ele pe componente, în acest mod sistemul o să fie ușor de menținut, iar design-ul unei pagini o să poată fi schimbat fără a afecta restul paginilor.

Pagina `welcome.jsp` este pagina principală a aplicației care poate fi vizualizată de orice utilizator. Ea conține un dashboard în care sunt afișate date în timp real referitoare complexului sportiv și anume: numărul de utilizatori înregistrați în sistem, numărul de abonamente create în ziua curentă, numărul de abonamente care expiră în ziua curentă și numărul de programări efectuate în ziua curentă. De pe această pagină, utilizatorul poate naviga pe pagina de „register” pentru crearea unui cont, pe pagina de „Login” pentru autentificare, pe pagina unde se află galeria foto a aplicației și poate să vizualizeze tipurile generale de abonamente existente în cadrul complexului sportiv.

Pagina „`Login.jsp`” este pagina care-i oferă utilizatorului posibilitatea de a se loga în sistem. Dacă utilizatorul introduce credențiale greșite, o să fie afișat un mesaj de eroare pentru a-l informa că username-ul sau parola introdusă sunt greșite. Dacă utilizatorul o să introducă credențialele corecte o să fie redirecționat pe una dintre paginile corespunzătoare rolului său.

Pagina „`appointment.jsp`” este pagina în care utilizatorul o să poată efectua operații asupra programărilor. Pe această pagină utilizatorul o să aibă posibilitatea să aleagă una dintre operațiile următoare: vizualizarea tuturor programărilor efectuate, modificarea unei programări existente în cazul în care dorește o anumită schimbare a informațiilor corespunzătoare, o să poată șterge o programare în cazul în care au apărut anumite schimbări în programul lui și o să poată adăuga o nouă programare. La această pagină o să aibă acces și utilizatorul cu rolul de administrator în cazul în care dorește să vizualizeze programările adăugate sau să efectueze anumite modificări asupra lor.

Pagina „`pass.jsp`” este pagina în care utilizatorul o să poată efectua operații asupra abonamentelor. Pe această pagină utilizatorul o să aibă posibilitatea să aleagă una dintre operațiile următoare: vizualizarea tuturor abonamentelor comandate, modificarea unui abonament existent în cazul în care dorește o anumită schimbare a duratei abonamentului, o să poată comanda un nou abonament și o să-l adăuge în coșul de cumpărături. La această pagină o să aibă acces și utilizatorul cu rolul de administrator în cazul în care dorește să vizualizeze programările adăugate sau să efectueze anumite modificări asupra lor.

Pagina „`orar.jsp`” este pagina pe care este afișat orarul programărilor existente pe săptămâna curentă. Utilizatorul o să aibă posibilitatea să adăuge o programare în orarul respectiv.

Pagina „workout.jsp” o să conțină toate sporturile disponibile în cadrul acestui complex sportiv. Pe această pagina o să aiba acces doar utilizatorul cu rolul de administrator și utilizatorul cu rolul de angajat al sistemului pentru a putea șterge anumite sporturi sau pentru a adăuga unele noi.

Pagina „403.jsp”, este pagina pe care este redirecționat un utilizator în cazul în care încearcă să acceseze una dintre paginile accesibile doar după logarea în sistem. Pe această pagina o să fie afișat un mesaj de eroare „You do not have permission to access this page!”

Pagina „pictures.jsp” este doar o pagină de prezentare care poate fi accesată de pe pagina de welcome, de orice tip de utilizator. Pe această pagină o să se afle o mică descriere a fiecărui sport din cadrul complexului sportiv cât și o imagine reprezentativă pentru acel sport.

În figura următoare este reprezentată arhitectura acestei componente, „web layer”, o arhitectura generală a sistemului propus, pe care se poate mapa fiecare dintre paginile JSP enumerate mai sus. „Page.jsp” poate fi înlocuită cu oricare dintre paginile prezentate, pe fiecare pagina se aplică un anumit design cu ajutorul limbajului CCS, limbaj de stilizare folosit pentru descrierea paginilor web. Utilizatorul accesează pagina JSP printr-un URL, și o să primească o pagina cu conținutul cerut, pe care au fost aplicate diferite stiluri de design.

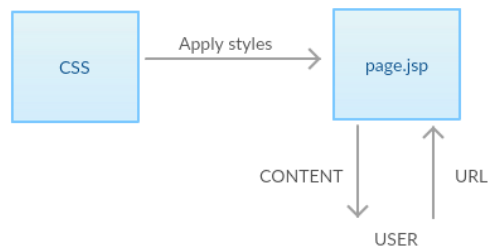


Figura 5.2 Arhitectura layer-ul Web

Din structura layer-ul web fac parte și controllerele sistemului dezvoltat. Controllerele sunt folosite pentru procesarea request-urilor transmise de utilizator și returnarea răspunsului corespunzător. Există șase clase de controller în sistemul implementat, specifice fiecărei funcționalitate existentă. Fiecare dintre aceste clase este marcată cu adnotarea `@Controller` iar fiecare metodă este marcată cu una dintre adnotările `@PostMapping` sau `@GetMapping`. Adnotarea `@Controller` indică faptul că acea clasă servește rolului de controller în sistemul implementat. `@GetMapping` este o adnotare compusă care acționează ca o comandă rapidă pentru `@RequestMapping` (method = RequestMethod.GET). Adnotarea este folosită pentru maparea cererilor HTTP GET pe anumite metode. `@PostMapping` este o adnotare compusă care acționează ca o comandă rapidă pentru `@RequestMapping` (method = RequestMethod.POST). Aceasta este o metodă de identificare a cererii și corespunde metodei HTTP denumite în mod similar. Metoda Java adnotată cu acest desemnator de metodă de request va procesa

cererile HTTP POST. Comportamentul unei resurse este determinat de metoda HTTP la care răspunde resursa. Clasele de controller specifice sistemului dezvoltat sunt următoarele:

`UserController` conține metodele specifice utilizatorului sistemului. În aceasta clasă este creat un obiect de tipul clasei `UserService` cu ajutorul căruia se apează metodele specifice. Fiecare metoda are un URL specific, prin care se va putea accesa metoda cu ajutorul browserului. Pentru adăugarea unui utilizator se folosește URL-ul `"/add-user"`, pentru ștergerea unui utilizator se folosește URL-ul `"/delete-user"`, pentru vizualizarea tuturor utilizatorilor se folosește URL-ul `"/all-users"`, iar pentru editarea informațiilor corespunzătoare unui user se folosește URL-ul `"/update-user"`. Tot în acest controller, există metoda care returnează numărul total al utilizatorilor înscrși în sistem pentru afișarea lui pe dashboard-ul principal. Fiecare metodă returnează pagina `„user.jsp"`. Pentru metodele specifice request-urilor de tip GET, se folosește metoda `“setAttribute"` care stochează un atribut într-un request pentru a-l putea folosi în pagina `user.jsp`.

`AppointmentController` conține metodele specifice programărilor. În aceasta clasă este creat un obiect de tipul clasei `AppointmentService` cu ajutorul căruia se apează metodele specifice. Fiecare metoda are un URL specific, prin care se va putea accesa metoda cu ajutorul browserului. Pentru adăugarea unei programări se folosește URL-ul `"/add-appointment"`, pentru ștergerea unei programări se folosește URL-ul `"/delete-appointment"`, pentru vizualizarea tuturor programărilor se folosește URL-ul `"/all-appointments"`, iar pentru editarea unei programări se folosește URL-ul `"/update-appointment"`. Fiecare metodă returnează pagina `„appointment.jsp"`. Pentru metodele specifice request-urilor de tip GET, se folosește metoda `“setAttribute"` care stochează un atribut într-un request pentru a-l putea folosi în pagina `appointment.jsp`.

`PassController` conține metodele specifice programărilor. În aceasta clasă este creat un obiect de tipul clasei `PassService` cu ajutorul căruia se apează metodele specifice. Fiecare metoda are un URL specific, prin care se va putea accesa metoda cu ajutorul browserului. Pentru comandarea unui abonament se folosește URL-ul `"/add-pass"`, pentru ștergerea unui abonament se folosește URL-ul `"/delete-pass"`, pentru vizualizarea tuturor abonamentelor se folosește URL-ul `"/all-passes"`, iar pentru editarea unui abonament se folosește URL-ul `"/update-pass"`. Pe lângă acestea, mai există două metode, o metoda care returnează numărul de abonamente create în ziua curentă, a doua metodă care returnează numărul de abonamente care expiră în ziua curentă. Fiecare dintre aceste metode returnează pagina `„pass.jsp"`. Pentru metodele specifice request-urilor de tip GET, se folosește metoda `“setAttribute"` care stochează un atribut într-un request pentru a-l putea folosi în pagina `„pass.jsp"`.

`WorkoutController` conține metodele specifice antrenamentelor sportive. În aceasta clasă este creat un obiect de tipul clasei `WorkoutService` cu ajutorul căruia se apează metodele specifice. Fiecare metoda are un URL specific, prin care se va putea accesa metoda cu ajutorul browserului. Pentru adăugarea unui workout, admin-ul folosește URL-ul `"/add-workout"`, pentru ștergerea unui antrenament se folosește URL-ul `"/delete-workout"`, pentru vizualizarea tuturor antrenamentelor se folosește URL-ul `"/all-workouts"`, iar pentru editarea unui workout se folosește URL-ul `"/update-workout"`. Fiecare metodă returnează pagina `„workout.jsp"`. Pentru metodele specifice

request-urilor de tip GET, se folosește metoda “setAttribute” care stochează un atribut într-un request pentru a-l putea folosi în pagina workout.jsp.

MailController conține metoda specifică pentru trimiterea mail-urilor. În aceasta clasă este creat un obiect de tipul clasei MailService cu ajutorul căruia se apelează metoda sendMail(), folosind URL –ul „/mail”.

ReportController conține metoda specifică pentru generarea rapoartelor. În aceasta clasă este creat un obiect de tipul clasei ReportService cu ajutorul căruia se apelează metoda createReport(), folosind URL –ul „/report”.

Toate aceste clase de controllere sunt reprezentate în figura următoare:

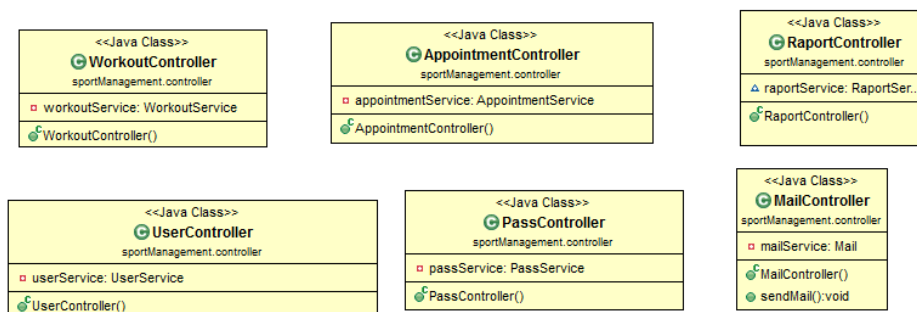


Figura 5.3 Diagrama claselor “Controller”

5.1.1.2. Service Layer

Service Layer este reprezentat de layer-ul se află sub layer-ul web. Acționează ca o limită a tranzacțiilor și conține atât serviciile aplicației, cât și servicii de infrastructură.

Serviciile aplicației sunt folosite de consumatorii externi pentru a comunica cu sistemul, mai concret, de utilizatorii care accesează aplicația dezvoltată pentru a dispune de beneficiile complexului sportiv. Acestea sunt reprezentate de operațiile tipice, CRUD-uri.

Exista patru clase de servicii: UserService, AppointmentService, WorkoutService și PassService. Fiecare dintre aceste clase este marcată cu adnotările @Service și @Transactional.

Adnotarea @Service este folosită pentru a indica faptul că această clasă este componenta layer-ul de servicii, iar adnotarea @Transactional este folosită pentru ca framework-ul Spring sa execute implicit tranzacțiile de commit/rollback. În fiecare dintre aceste clase, este definit un obiect corespunzător de tipul claselor de repository, iar pentru fiecare operație se creează câte o metoda în care, folosind acel obiect, se apelează metodele din repository, pentru a fi posibilă transmiterea datelor către următorul layer al aplicației.

Toate aceste clase de servicii sunt reprezentate în figura următoare:

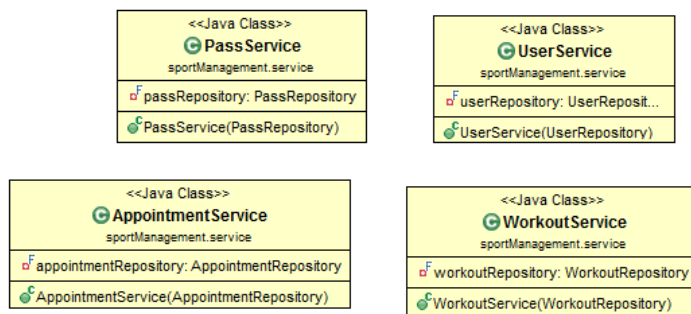


Figura 5.4 Diagrama claselor “Service”

UserService este clasa de servicii care conține metodele corespunzătoare unui utilizator. În această clasă sunt reprezentate operațiile CRUD pentru utilizator cât și o metoda care returnează numărul utilizatorilor înscrși în sistem pentru a putea fi afișat pe dashboard-ul de pe pagina principală a sistemului.

AppointmentService este clasa de servicii care conține metodele corespunzătoare unei programări. În această clasă sunt reprezentate operațiile CRUD pentru programări cât și o metoda care returnează numărul programărilor efectuate în ziua curentă pentru a putea fi afișat pe dashboard-ul de pe pagina principală a sistemului.

PassService este clasa de servicii care conține metodele corespunzătoare unui abonament. În această clasă sunt reprezentate operațiile CRUD pentru abonamente cât, o metoda care returnează numărul abonamentelor comandate în ziua curentă cât și numărul abonamentelor care expiră în ziua curentă pentru a putea fi afișate pe dashboard-ul de pe pagina principală a sistemului.

WorkoutService este clasa de servicii care conține metodele corespunzătoare unui abonament. În această clasă sunt reprezentate operațiile CRUD pentru abonamente cât, o metoda care returnează numărul abonamentelor comandate în ziua curentă cât și numărul abonamentelor care expiră în ziua curentă pentru a putea fi afișate pe dashboard-ul de pe pagina principală a sistemului.

Serviciile de infrastructură sunt folosite pentru serviciile de mail și generarea rapoartelor.

Serviciul de mail a fost realizat utilizând protocolul SMTP (Simple Mail Transfer Protocol), folosit pentru trimiterea și primirea email-urilor. În figura de mai jos este reprezentată metoda implementată pentru trimiterea mail-urilor. Metoda are trei parametrii, „to” – reprezentând destinatarul la care se trimite mail-ul, parametrul „subject” reprezentat de subiectul mail-ului și parametrul „content” reprezentat de conținutul mail-ului. În prima parte a metodei, este creat un obiect de tipul clasei „Session”, care colectează împreună proprietățile și valorile implicite utilizate de API-urile de mail. Obiectul „props” este un declarat la începutul clasei în care este conținută metoda de mail de mai jos, și este reprezentat de un set persistent de proprietăți:

- portul protocolului de mail SMTP
- serviciul de mail setat, în cazul proiectului propus gmail
- setarea proprietății “mail.smtp.starttls.enable” pe true, pentru o conexiune sigură.
- Alegerea modului de autentificare, dacă autentificarea trebuie să se facă sau nu înainte de trimiterea mail-ului.

După ce obiectul de tip “Session” a fost creat, este implementată crearea mesajului. Pentru crearea mesajului se transmite obiectul session în constructorul clasei “MimeMessage”. Odată ce obiectul mesajului este creat, trebuie să stocăm informațiile în el. Clasa de mesaje implementează interfața javax.mail.Part în timp ce javax.mail.internet.MimeMessage implementează javax.mail.internet.MimePart. În continuarea metodei, am utilizat patru metode necesare pentru crearea mesajului:

- setFrom(Address address) – folosită pentru setarea câmpului din antet
- addRecipients(Message.RecipientType type, String addresses) – folosită pentru a adăuga adresa dată destinatarului
- setSubject(String subject) – folosită pentru a seta câmpul antetului subiect
- setText(String textmessage) – folosită pentru a seta textul ca mesaj de conținut.

La sfârșitul metodei, se folosește clasa “Transport” care implementează metodă “send” pentru trimiterea mail-ului creat.

```
public void sendMail(String to, String subject, String content) {
    Session session = Session.getInstance(props,
        new javax.mail.Authenticator() {
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username, password);
            }
        });
    try {
        Message message = new MimeMessage(session);
        message.setFrom(new InternetAddress(username));
        message.setRecipients(Message.RecipientType.TO,
            InternetAddress.parse(to));
        message.setSubject(subject);
        message.setText(content);

        Transport.send(message);

        System.out.println("Mail sent.");
    } catch (MessagingException e) {
        e.printStackTrace();
    }
}
```

Figura 5.5 Metoda pentru trimiterea unui mail

Serviciul de creare rapoarte a fost realizat utilizând API-ul open source IText folosit pentru generarea pdf-urilor în Java. În figura de mai jos este reprezentată o parte importantă a metodei implementate pentru generarea rapoartelor. Clasa com.itextpdf.text.Document în IText reprezintă un document PDF. Este una dintre clasele de bază din IText. Pentru generarea unui document PDF de la zero, am folosit clasa Document. Mai întâi trebuie am crea o instanță de document. După crearea instanței am folosit metoda pentru deschiderea documentului, iar după asta am adăugat conținut în document. În cele din urmă, am închis instanța Document. Clasa com.itextpdf.text.Chunk

În IText reprezintă cea mai mică "bucată" de text posibil. O bucată poate conține de la un singur caracter, până la câteva propoziții. Am folosit „chunk” pentru trecerea la o linie nouă în document, pentru a nu se confunda titlul cu textul din raport. Ultima clasă folosită, este clasa com.itextpdf.text.Paragraph din IText care reprezintă un "paragraf" de text. Într-un paragraf putem seta alinierea paragrafelor, indentarea și spațierea înainte și după paragraf.

```
Document document = new Document();
List unorderedList = new List(List.UNORDERED);
for(Appointment appointment : appointments){
    unorderedList.add(new ListItem(appointment.toString()));
}

PdfWriter writer = PdfWriter.getInstance(document, new FileOutputStream(file+nameFile));
document.open();
document.add(new Paragraph(title + " " + sum,font3));
document.add( Chunk.NEWLINE );
document.add( Chunk.NEWLINE );
document.add(unorderedList);
document.close();
writer.close();
} catch (DocumentException e)
{
    e.printStackTrace();
} catch (FileNotFoundException e)
{
    e.printStackTrace();
}
```

Figura 5.6 Metoda pentru generarea rapoartelor

În acest layer a fost dezvoltată securitatea aplicației. Securitatea a fost realizată folosind framework-ul Spring Security. Acest framework oferă servicii complete de securitate, suportând atât autentificarea cât și autorizarea.

Pentru dezvoltarea securității au luat în considerare următorii pași:

- Am declarat un obiect datasource adnotat cu @Autowired. Aceasta caută definiția datasource-ului în toate clasele din același pachet, în cazul nostru am definit-o în clasa de configurări MvcConfig.java.
- Am creat două interogări pentru AuthenticationManagerBuilder. Unul pentru autentificare în usersByUsernameQuery și celălalt pentru autorizare în authoritiesByUsernameQuery ca și în figura de mai jos. Aceste query-uri selectează datele unui utilizator din baza de date, și verifică acel utilizator atunci când se face logarea.

```
public void configAuthentication(AuthenticationManagerBuilder auth) throws Exception {
    auth.jdbcAuthentication().dataSource(dataSource)
    .usersByUsernameQuery(
        "select username,password, enabled, email, nume, prenume, id_user from USERS where username=?")
    .authoritiesByUsernameQuery(
        "select username, role from USER_ROLES where username=?");
}
```

Figura 5.7 Metoda pentru configurarea autentificării

- Am configurat HttpSecurity pentru a defini ce pagini trebuie să fie securizate, autorizate, neautorizate, neasigurate, pagina de conectare, pagina de logout, pagina de acces refuzată etc. Un lucru important este de a observa aici ordinea configurației. Configurația specifică anumitor pagini sau url-uri trebuie să fie plasată înaintea configurațiilor care sunt comune în majoritatea url-urilor. Aceste detalii sunt reprezentate în codul de mai jos.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable();
    http.authorizeRequests().antMatchers("/appointment/**", "/appointment").access("hasRo
    .antMatchers("/user/**", "/user").access("hasRole('ROLE_ADMIN')")
    .antMatchers("/appointment/**", "/appointment").access("hasRole('ROLE_USER')")
    .anyRequest().permitAll()
    .and()
    .formLogin().loginPage("/login")
    .usernameParameter("username").passwordParameter("password")
    .and()
    .logout().logoutSuccessUrl("/login?logout")
    .and()
    .exceptionHandling().accessDeniedPage("/403");
}
```

Figura 5.8 Metoda pentru configurarea autorizării

5.1.1.3. Repository layer

Repository layer este reprezentat de layer-ul cel mai de jos layer. Acest nivel este reprezentat de server-ul de management bazei de date, având legătură direct cu baza de date relațională a sistemului. În acest modul se efectuează operațiile de adăugare, ștergere, modificare asupra bazei de date. Datele primite de la baza de date sunt modelate și sunt transmise către layer-ul de servicii sub forma unor colecții de date.

În cadrul sistemului propus, acest layer a fost organizat utilizând interfața ”JPA Repository”, care este o extensie a interfeței ”Repository”. Această interfață ne furnizează cele mai relevante metode CRUD pentru accesul la date.

Exista patru clase de Repository în sistemul dezvoltat. UserRepository, AppointmentRepository, PassRepository și WorkoutRepository.

Toate aceste clase de servicii sunt reprezentate în figura următoare:



Figura 5.9 Diagrama de clase ”Repository”

Fiecare dintre aceste clase are structura următoare :

```
public interface AppointmentRepository extends JpaRepository<Appointment, Integer>{
}
```

Clasele UserRepository si AppointmentRepository conțin în plus metodele folosite la crearea dashboardului de pe pagina principală. Pentru crearea acestor metode s-a folosit adnotarea @Query. Această adnotare este utilizată pentru a scrie interogarea de preluare a datelor într-un mod mai flexibil. De exemplu, pentru metoda din UserRepository s-a folosit următoarea interogare:

```
@Query("SELECT COUNT(*) FROM USERS")
public int nrUsers();
```

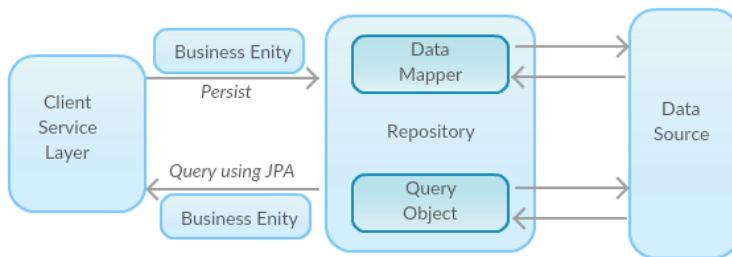


Figura 5.10 Arhitectura layerului Repository

5.2. Diagramele sistemului

5.2.1. Diagrame de navigare

În acest subcapitol vor fi prezentate diagramele de navigare ale aplicației propuse. O diagram de navigare, este o diagram de arhitectura, “low-level”, care documentează modul de navigare în jurul componentelor de prezentare ale unei aplicații.

În figura următoare vor fi descrise toate cazurile de navigare pentru tipurile de actori ai aplicației. În aplicație exista un număr de pagini care pot fi accesate de fiecare utilizator. Aceste cazuri vor fi prezentate în figura următoare.

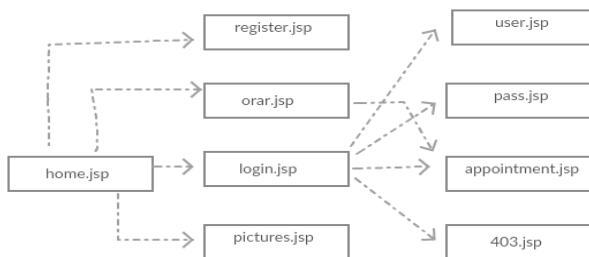


Figura 5.11 Pagini web comune rolurilor.Navigarea

În cazul în care utilizatorul introduce credențiale valide unui utilizator, logarea se face cu succes, iar utilizatorul este redirectionat la una din paginile aplicației în funcție de rolul utilizatorului logat. Pagina de home este disponibilă atât din pagina de start a aplicației, cât și după logare, din restul paginilor existente. Aceasta reprezintă un “design” general al aplicației disponibil inclusive tipului de utilizator fără cont (vizitator), pentru a-și face o idee despre aplicație, și pentru a putea vizualiza anumite detalii generale despre complexul sportiv. În cazul în care un utilizator nu introduce parola sau username-ul corect, o să fie afișat un mesaj de eroare: “Invalid username or password”. În cazul în care un utilizator care nu este logat în aplicație încearcă să acceseze una din paginile disponibile doar după logare, utilizatorul o să fie redirectionat către pagina de eroare 403.jsp, pe care o să fie afișat mesajul: „You do not have permission to access this page!”

Administratorul: are acces la toate paginile reprezentate în figura de mai sus. El o să aibă acces atât la programările și abonamentele efectuate de clienți, cât și la conturile create de utilizatori. În cazul în care unul dintre utilizatori încalcă anumite reguli, admin-ul o să aibă posibilitatea să-i elimine din sistem. Diagrama de navigare pentru utilizatorul cu rolul de administrator este identică cu diagrama de navigare din figura 4.12. Pagina web comuna rolurilor.

Clientul: În figura următoare o să fie reprezentate cazurile de navigare pentru utilizatorul cu rolul de client. Pentru acest tip de utilizator, navigarea este posibilă pe paginile comune tuturor utilizatorilor, pe paginile necesare pentru adaugare unei programări și pe paginile necesare pentru comandarea unui abonament.

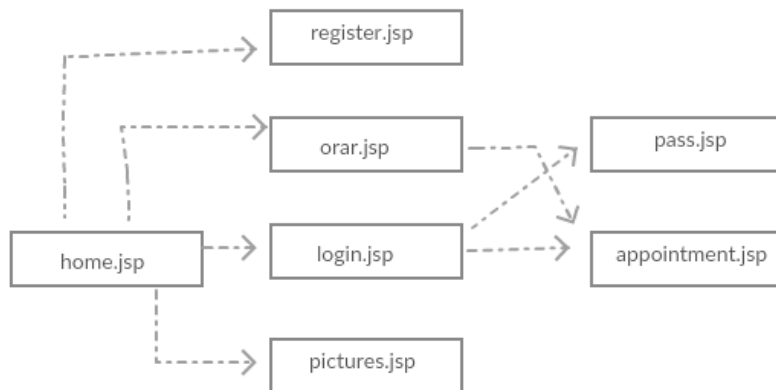


Figura 5.12 Diagramă de navigare pentru utilizatorul client

Anagajat: În figura următoare o să fie reprezentate cazurile de navigare pentru utilizatorul cu rolul de anagajt. Pentru acest tip de utilizator, navigarea este posibilă pe paginile comune tuturor utilizatorilor, cât și pe paginile pentru gestionarea utilizatorului cu rolul de „client”.

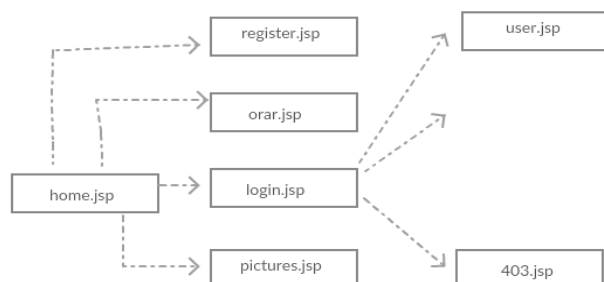


Figura 5.13 Diagramă de navigare pentru utilizatorul cu rolul de angajat

Vizitator: În figura următoare o sa fie reprezentate cazurile de navigare pentru utilizatorul cu rolul de vizitator. Pentru acest tip de utilizator, navigarea este posibilă doar pe paginile comune tuturor utilizatorilor.

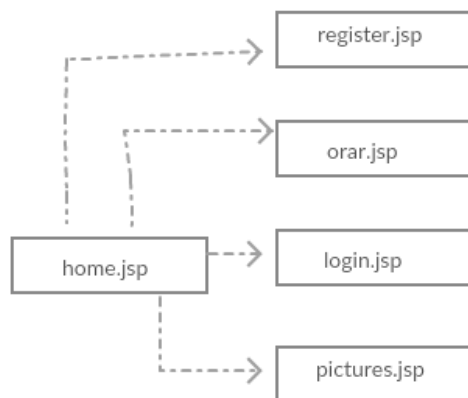


Figura 5.14 Diagramă de navigare pentru utilizatorul cu rolul de vizitator

5.2.2. Diagrama de pachete

În figura de mai jos este reprezentată diagrama de pachete a sistemului dezvoltat. În această diagramă sunt reprezentate dependențele dintre pachetele sistemului.

Pachetul SportManagement este pachetul care conține clasa Application.Java. Această clasă conține metoda “run”, de tipul clasei “SpringApplciation” care oferă o modalitate simplă de a porni aplicația dintr-o clasă principală a proiectului.

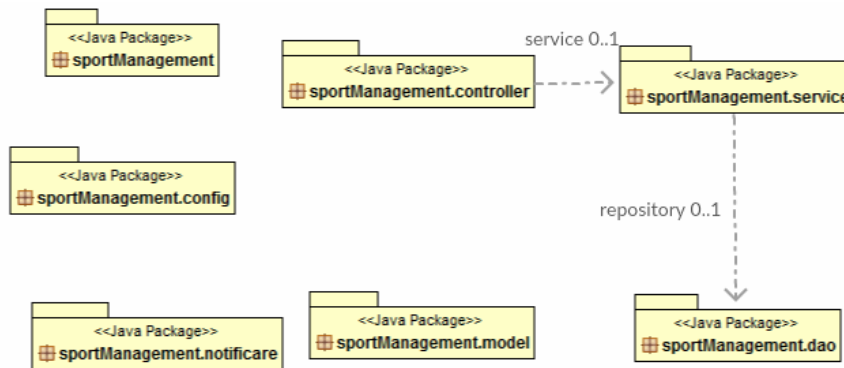


Figura 5.15 Diagrama de pachete a sistemului

```

public static void main (String[] args)
{
    SpringApplication.run(Application.class, args);
}
    
```

Clasa în care este conținută metoda de mai sus, este adnotată cu `@SpringBootApplication`. Această încapsulează următoarele adnotări:

- `@Configuration` etichetează clasa ca o sursă de definiții ale "bean-urilor" pentru contextul aplicației.
- `@EnableAutoConfiguration` face ca Spring Boot pentru să înceapă adăugarea bean-urilor pe baza setarilor clasei, a altor bean-uri și a diferitelor setări ale proprietăților.
- În mod normal, am adăuga adnotarea `@EnableWebMvc` pentru o aplicație Spring MVC de primăvară, dar Spring boot adaugă automat atunci când găsește spring-webmvc în configurările clasei. Aceasta cataloghează aplicația ca aplicație web și activează anumite comportamente, cum ar fi configurarea unui `DispatcherServlet`.
- `@ComponentScan` face ca Spring să caute alte componente, configurații și servicii în pachetul principal, permițându-i să găsească controlerele adăugate în sistem.

Pachetul `sportManagement.config` conține clasele de configurare a Spring Security, `MvcConfig` și `WebSecurityConfig`, clase descrise mai sus, în cadrul layer-ului de servicii.

Pachetul `sportManagement.notification`, conține clasele: "MailService" și "ReportService". În aceste clase se află metode pentru trimiterea mail-ului, respectiv pentru generarea rapoartelor de catre administrator.

Pachetul `sportManagement.model` conține clasele de entități ale sistemului. Aceste clase sunt reprezentate în figura de mai jos.

Pachetul `sportManagement.controller` se află toate controlerele aplicației, acestea fiind descrise în detaliu în acest capitol, la layer-ul web.

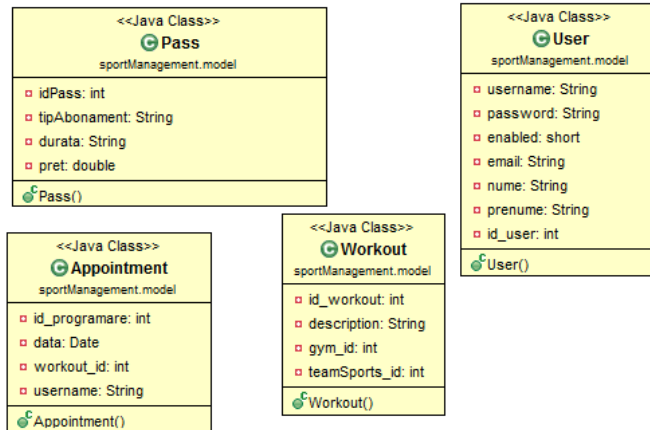


Figura 5.16 Diagrama de clase “Model”

5.3. Deployment

Diagrama de deployment este reprezentată de componentele hardware care sunt necesare pentru a rula componentele software ale sistemului propus și modul în care aceste componente sunt interconectate. Componentele diagramei mai sunt numite artefacte ale sistemului. Scopul unei diagrame de deployment este de a prezenta structura hardware necesară rulării sistemului.

Aplicația web poate fi accesată de pe un sistem desktop. Componenta fizică a sistemului pe care rulează această aplicație este serverul, server pe care se găsește și baza de date a sistemului, baza de date realizată în SQL Server.

În următoare figură este reprezentată diagrama de deployment a aplicației. Componenta din stanga diagramei este reprezentată de clienți, consumatorii aplicației, serverul web la mijloc, iar în dreapta serverul bazei de date, SQL Server.

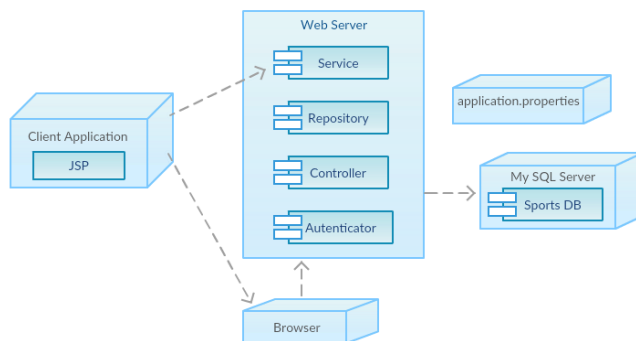


Figura 5.17 Diagrama de deployment

5.4. Proiectarea bazei de date

Proiectarea bazei de date a fost realizată utilizând limbajul SQL Server. Baza de date este structurată în șapte tabele. Trei dintre tabele sunt folosite pentru comandarea abonamentelor, iar celelalte tabele pentru restul operațiilor dezvoltate în cadrul aplicației.

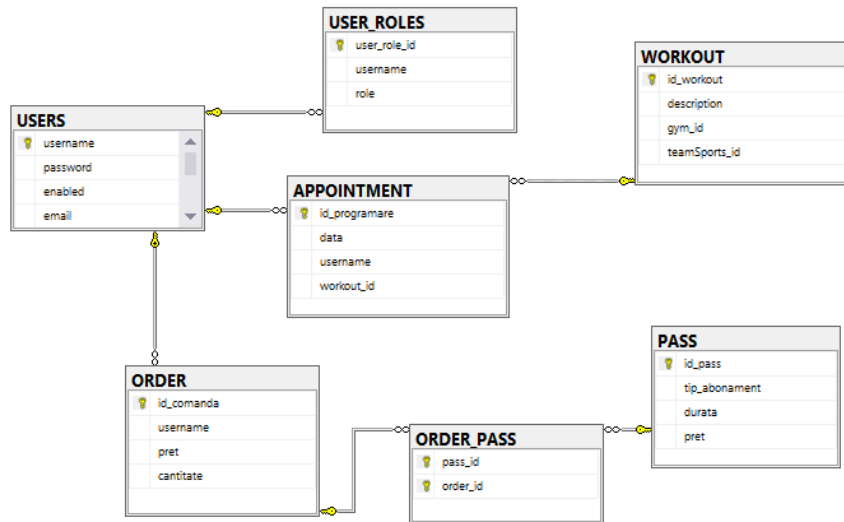


Figure 5.1 Diagrama bazei de date

În proiectarea bazei de date pentru a se diminua posibilitatea de a ajunge la informații eronate s-a dorit reprezentarea cât mai corectă a datelor. Pentru a ajunge la acest scop, s-a normalizat a bazei de date. În continuare o să fie descrise formele normale pe care baza de date le respectă.

Forma normală 1, presupune ca ne trebuie sa existe rânduri de date care să conțină grupuri de informații repetate, adică fiecare set de coloane trebuie să aibă o valoare unică, astfel încât să nu poată fi utilizate mai multe coloane pentru a prelua același rând. Fiecare tabelă ar trebui să fie organizată pe rânduri și fiecare rând ar trebui să aibă o cheie primară care o deosebește ca fiind unică. Cheia primară este, de obicei, o coloană singură, dar uneori pot fi combinate mai mult de o coloane pentru a crea o singură cheie primară.

Forma normală 2, dacă fiecare atribut care nu aparține cheii primare, este total dependent funcțional de cheia primară și este îndeplinită dacă FN1 este îndeplinită și Această formă trebuie verificată la relațiile care au cheie compusă din mai multe atribute pe post de cheie primară. În baza de date a sistemului, pentru toate relațiile cheia este compusă dintr-un singur atribut deci putem concludem ca baza de date a sistemului se află în această formă normală.

Forma normală 3 este îndeplinită dacă primele doua forme normale sunt îndeplinite, dacă nu există niciun atribut care să nu aparțină cheii principale și care să fie

tranzitiv dependent de cheia principal. În general, astfel de dependențe există doar dacă tabelul stochează date din mai multe domenii ceea ce nu este cazul în sistemul propus.

Forma normală Boyce-Codd este îndeplinită dacă orice determinant din relație este cheia candidat.

Baza de date a sistemului conține următoarele **tabele**:

Tabela Users: În această tabela sunt stocate toate datele unui utilizator. Pe lângă datele generale, username și parola, tabela mai conține un camp numit "enabled". Dacă acest camp este setat la valoarea 1, utilizatorul nu mai poate să se logheze în sistem. Acest lucru poate fi considerat o măsură de securitate pentru ca utilizatorii să nu se mai poată loga în sistem atunci când utilizatorul considera că au încălșcat anumite drepturi.

Tabela User_Roles este folosită pentru salvarea rolului fiecărui utilizator. În sistem există 3 roluri posibile:

- ROLE_ADMIN: rolul pe care îl are administratorul complexului
- ROLE_ANGAJAT: rolul pe care îl are angajatul complexului
- ROLE_CLIENR: rolul pe care îl are un client al complexului sportiv.

Această tabela și tabela Users sunt tabele folosite și pentru autentificare și autorizare, țin funcție de rolurile stocate.

Tabela Appointment este folosită pentru informațiile necesare programărilor. Aceste informații sunt folosite zilnic de clienți, pentru programarea la una dintre activitățile complexului sportiv.

Tabela workout este folosită doar de utilizatorul cu rolul de administrator, pentru a adăuga informațiile necesare sporturilor din cadrul complexului sportive, și pentru a șterge/adauga un anumit sport.

Tabela Order este tabela în care sunt stocate comenzile abonamentelor.

Tabela Pass este tabela în care sunt stocate toate detaliile abonamentelor, datele din această tabela fiind folosite în general de către clienți.

Relațiile dintre tabele, folosite în structura bazei de date, sunt de două de două tipuri:

- One-to Many: între tabelele Users-UserRoles, User-Appointment, User-Order și Appointment-Workout.
- Many-to-many între tabelele Order-Pass, relație care a fost împărțită în one-to many și many-to-one folosind tabela intermediară Order_Pass.

Capitolul 6. Testare și Validare

În acest capitol o să fie prezentate principalele metode de testare pentru sistemul dezvoltat. Pentru testare am ales doua funcționalități importante ale sistemului propus, comandarea unui abonament și adăugarea unei programări. Pentru fiecare dintre acest funcționalități o sa creăm câte un caz de test corespunzător, un set de condiții sau variabile prin care vom determina dacă sistemele propuse testării, satisfac cerințele și funcționează corect. Procesul de dezvoltare a cazurilor de testare ne poate ajuta, de asemenea, să găsim probleme în cerințele sau proiectarea aplicației.

Pentru crearea cazurilor de test trebuie să avem în vedere următorii termeni.

- Caz de testare: funcționalitatea sistemului pe care o supunem testării
- Preconții: orice premise sau condiții prealabile care trebuie îndeplinite înainte de a efectua testul
- Acțiuni: Pașii care trebuie urmați pentru a efectua cazul de test
- Rezultat așteptat: rezultatul pe care trebuie sa-l primim dupa fiecare acțiune.

În tabelul următor o să fie reprezentat cazul de test pentru adăugare unei programări:

Caz de testare: Adăugarea unei programări

Preconții: Utilizatorul trebuie sa aibă creat un cont

Acțiune	Rezultat așteptat
1.Utilizatorul acceseaza pagina principală a aplicației.	Pagina principală este deschisă cu toate butoanele și opțiunile care ar trebui să se regăsească.
2. Utilizatorul accesează butonul de login.	Utilizatorul este redirecționat pe pagina de login unde își poate trece credențialele.
3. Utilizatorul își introduce credențialele si apasă butonul de login.	Utilizatorul este redirecționat pe pagina destinată tipului de utilizator cu rolul de client.
4. Utilizatorul alege opțiunea “New Appointment” din bara de meniu	Pe pagina accesată de utilizator, exista form-urile necesare pentru introducerea informațiilor corespunzătoare unei programări, si buton-ul de save, folosit pentru salvarea datelor introduse
5. Utilizatorul apasă butonul de Save.	Datele sunt salvate cu success iar pe ecran este afișat un mesaj cu textul. Programarea a fost efectuată cu succes. Un mail de confirmare a fost trimis.”.
6.Utilizatorul verifică mail-ul.	Mail-ul a fost primit și conține toate informațiile importante despre programare.
7.Utilizatorul se delogheaza din sistem	Utilizatorul este delogat și redirecționat către pagina principală.

Tabel 6.1 Cazul de test pentru adăugarea unei programări

În urma efectuării cazului de test de mai sus, rezultatele au fost cele așteptat, adăugarea unei programări a fost efectuată cu success.

În tabelul următor o să fie reprezentat cazul de test pentru adăugare unei programări:

Caz de testare: Adăugarea unei programări

Preconțiții: Utilizatorul trebuie sa aibă creat un cont

Acțiune	Rezultat așteptat
1.Utilizatorul acceseaza pagina principală a aplicației.	Pagina principală este deschisă cu toate butoanele și opțiunile care ar trebui să se regăsească.
2. Utilizatorul accesează butonul de login.	Utilizatorul este redirecționat pe pagina de login unde își poate trece credențialele.
3. Utilizatorul își introduce credențialele si apasă butonul de login.	Utilizatorul este redirecționat pe pagina destinată tipului de utilizator cu rolul de client.
4. Utilizatorul alege opțiunea “New Pass” din bara de meniu	Pe pagina accesată de utilizator, exista form-urile necesare pentru introducerea informațiilor corespunzătoare unui abonament, si buton-ul de save, folosit pentru salvarea datelor introduse
5. Utilizatorul apasă butonul de Save.	Datele sunt salvate cu success iar pe ecran este afișat un mesaj cu textul. „Abonamentul a fost comandat cu succes. Un mail de confirmare a fost trimis.”.
6.Utilizatorul verifică mail-ul.	Mail-ul a fost primit și conține toate informațiile importante despre abonament.
7.Utilizatorul se delogheaza din sistem	Utilizatorul este delogat și redirecționat către pagina principală.

Tabel 6.2 Cazul de test pentru comandarea unui abonament

Capitolul 7. Manual de Instalare si Utilizare

În acest capitol sunt descriși, sub forma unui tutorial, pașii care trebuie urmați pentru realizarea cu succes a instalării componentelor sistemului pe o mașină. Tot în acest capitol vor fi prezentate resursele software și hardware necesare și un scurt manual de utilizare.

7.1. Aplicația web

Utilizatorii aplicație, mai precis clienții complexului sportiv, trebuie să dispună doar de resursele solicitate de un browser și nicio alta resursă hardware.

7.1.1. Instalarea și rutarea

Este recomandat ca sistemul pe care se instalează și se rulează aplicația web să fie windows, datorită faptului că este cel mai utilizat sistem de operare, și o persoană care nu are cunoștințe în domeniul ingineriei ar putea urma tutorialul.

- Instalarea utilitarului Eclipse
 - Este nevoie de software-ul Java instalat
 - Instalarea Java Development Kit
 - Java Runtime Environment
 - Vom deschide un browser și accesăm pagina www.eclipse.org.
 - Selectăm versiunea de Eclipse
 - Selectăm sistemul de operare, în cazul nostru windows
 - Dezarhivăm fișierele descărcate și pornim executabilul eclipse
 - La prima accesare selectăm un folder unde se vor salva proiectele create
- Instalarea utilitarului Microsoft Sql Server Management Studio
 - În primul rând trebuie să descărcăm fișierul de instalare SQL Server Management Studio (SSMS) (de pe pagina de descărcare SQL Server în funcție de tipul de server (x64, x86))
 - Odată ce am descărcat fișierul respectiv pentru tipul de server corespunzător, trebuie să-l executăm. Acesta ne va duce la primul ecran și anume SQL Server Centrul de instalare.
 - Trebuie să selectăm „New SQL stand-alone installation”
 - Următorul pas este alegerea opțiunii “Install Setup File”, după SQL Server Management Studio (SSMS) va începe descărcarea, extragerea și instalarea tuturor fișierelor de configurare necesare. După acest pas, un folder cu numele utilizatorului va fi prezent în “Program Files”

Importul proiectului

După deschiderea utilitarului Eclipse, selectăm *File-Import>Maven->Existing Maven Project* de unde putem alege fișierul arhivat cu proiectul pe care dorim să-l importăm.

Rularea proiectului

După importul proiectului, pentru a utiliza aplicația, trebuie să o rulăm. Pentru rularea aplicației, se dă click dreapta pe proiect, Run As-> Java Application. Pentru deschiderea aplicației, accesăm link-ul “localhost:port_number/welcomepage”.

7.1.2. Manual de utilizare

După accesarea URL-ului scris la rularea proiectul, utilizatorul o să fie redirecționat pe următoarea pagina principală

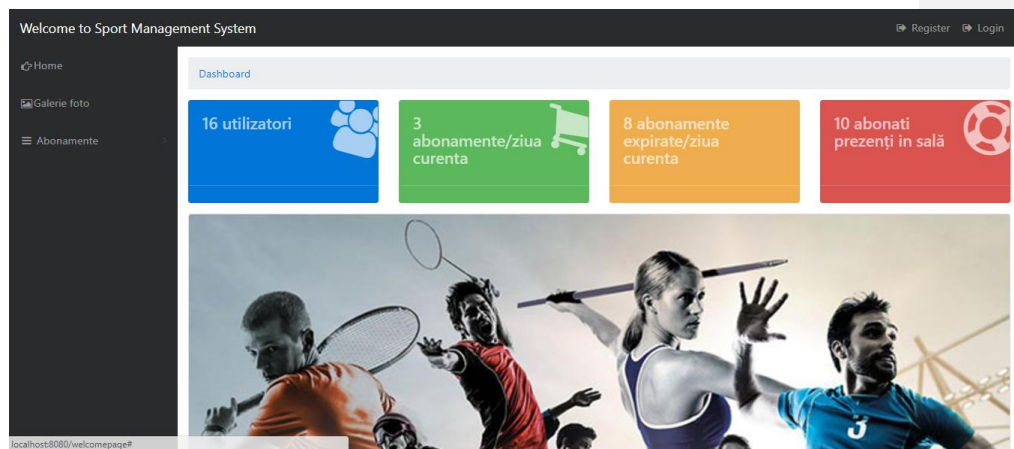


Figura 7.1 Pagina principală a aplicației

Pe această pagina, utilizator poate vizualiza un dashboard cu date informative despre complexul sportiv: numărul de utilizatori, numărul de abonamente create în ziua curentă, numărul de abonamente care expiră în ziua curentă și numărul de abonați prezenți în sală.

În cazul în care utilizatorul nu are cont în aplicație, poate alege opțiunea de a-și crea un cont, folosind opțiunea de “Register” de lângă butonul de login.

Tot de pe această pagina utilizatorul poate vizualiza o galerie foto cu imaginii reprezentative pentru sporturile din cadrul complexului sportiv.

Din meniul din partea stângă a paginii, utilizatorul poate extinde secțiunea de abonamente, pentru a vizualiza abonamentele disponibile în cadrul complexului sportiv.

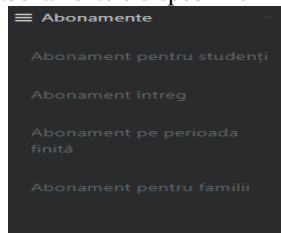


Figura 7.2 Tipurile de abonamente

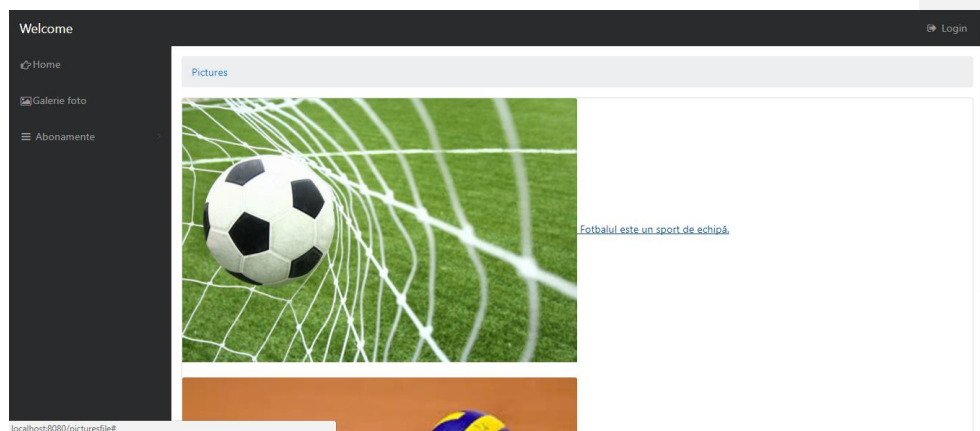


Figura 7.3 Pagina cu galeria foto a aplicației

De pe pagina principală, descrisă mai sus, utilizatorul poate să se logheze în aplicație. Pagina de login este reprezentată în figura de mai jos.

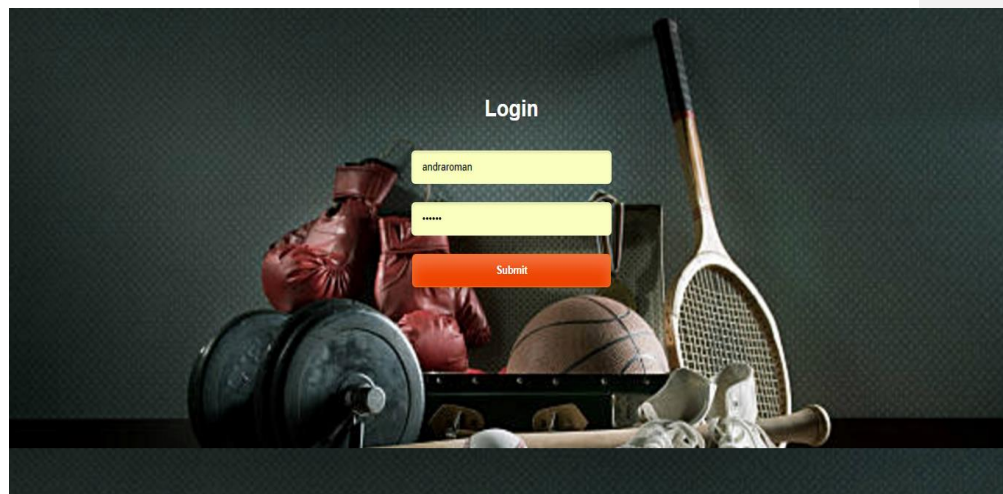


Figura 7.4 Pagina de login a aplicației

Pe această pagină de login, utilizatorul trebuie să-și introducă credențialele, și în funcție de rolul său în cadrul complexului sportiv, o să fie redirecționat pe o pagină.

În cazul în care utilizatorul nu are un cont creat, el nu o să poată accesa nicio pagină a aplicației în afara de pagina principală

Dacă utilizatorul are rolul de client, el o să fie redirecționat pe pagina cu programările sau cu abonamentele, în funcție de dorința sa.

În figurile de mai jos, sunt descrie operațiile pe care utilizatorul le poate face asupra abonamentelor.

Pentru opțiunea “All passes”, utilizatorul pot vizualiza toate abonamentele, ca în figura de mai jos. În cazul în care utilizatorul dorește ștergerea unui abonament, el trebuie sa apese pe “coșul” din stânga abonamentului respective, iar acesta o să fie șters.

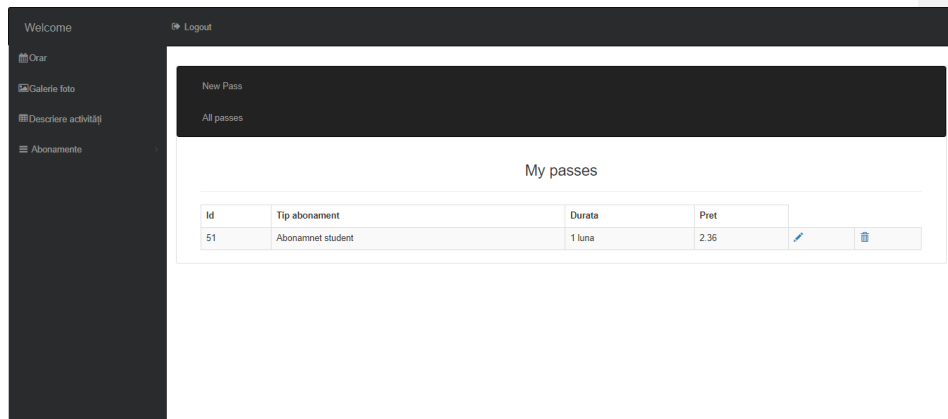


Figura 7.5 Vizualizarea tuturor abonamentelor

În cazul în care utilizatorul dorește să editeze abonamnetul respectiv, spre exemplu pentru prelungirea perioadei de valabilitate, el trebuie să apese pe „creionul” din dreapta abonamentului pe care vrea să-l editeze. Pentru adăugarea unui nou abonament, utilizatorul va alege opțiunea “New pass” din bara de meniu. Design-ul aplicației pentru adăugare și ștergerea unui abonament, este similar, singura diferență fiind că la editarea unui abonament, câmpurile cu informațiile abonamentul rămân pline. Design-ul operației de adăugare este reprezentat în figura de mai jos

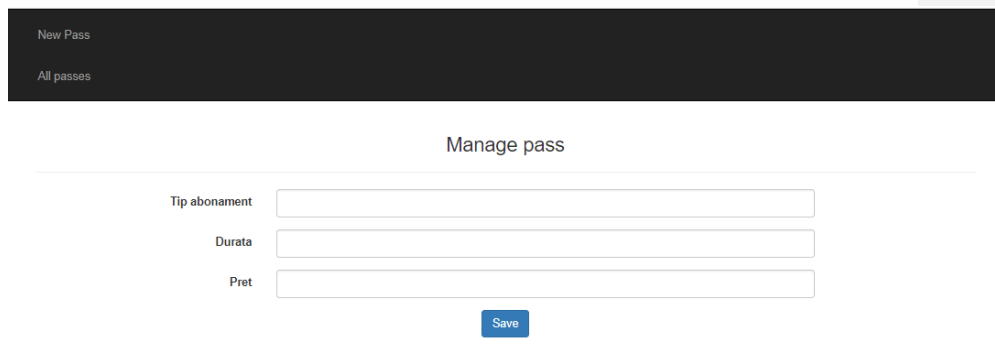
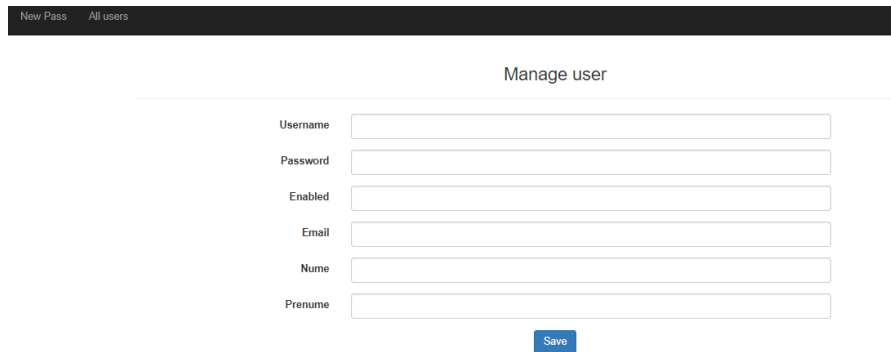


Figura 7.6 Adăugarea unui abonament

Administratorul, după logare este redirecționat pe pagina pentru gestionarea utilizatorilor. Pagina generală pe care există opțiunile de gestionarea utilizatorilor este descrisă mai jos, paginile specifice pentru fiecare operație în parte, sunt similare cu cele de mai sus.



The screenshot shows a web application interface for managing users. At the top, there is a dark navigation bar with two links: 'New Pass' and 'All users'. Below this, the main content area has a title 'Manage user' centered at the top. Underneath the title is a form with several input fields arranged vertically. The fields are labeled 'Username', 'Password', 'Enabled', 'Email', 'Nume', and 'Prenume'. Each label is positioned to the left of its corresponding input field. At the bottom of the form, there is a blue button labeled 'Save'.

Manage user	
Username	<input type="text"/>
Password	<input type="password"/>
Enabled	<input type="checkbox"/>
Email	<input type="text"/>
Nume	<input type="text"/>
Prenume	<input type="text"/>
<input type="button" value="Save"/>	

Figura 7.7 Pagina aplicației pentru gestiunea utilizatorilor

Capitolul 8. Concluzii

În acest capitol vor fi selectate câteva concluzii referitoare sistemului implementat. Se vor analiza realizările obținute și se vor trece în revistă obiectivele care au fost atinse prin acest proiect, iar apoi posibilităților de dezvoltare ulterioară

8.1. Realizările sistemului

Sistemul realizat reușește să își atingă scopul, eficientizând cu succes procesele de gestiune din cadrul complexul sportiv.

Obiectivele propuse pentru acest proiect au fost realizate, astfel sistemul oferă capabilitatea de creare a programărilor și de comandare a abonamentelor în cadrul complexului sportiv. Posibilitatea de gestionare atât a angajaților cât și a clienților complexului sportiv. Vizitatorii, de asemenea, au acces la o pagina principală în care pot vizualiza date informative referitoare complexului sportiv. Un alt obiect propus și realizat cu succes, a fost dashboar-ul de pe aceasta pagină, oferind utilizatorilor posibilitatea de a-și face o idee generală despre complex.

Utilizatorii sunt clasificați pe roluri, fiecare având meniuri și resurse specifice pe care le pot accesa. A fost asigurată și securitatea aplicației care folosește autentificarea și autorizarea pe bază a Spring Security. În acest mod, pe lângă logare, utilizatorii nu v-or putea accesa paginile destinate utilizatorilor logați.

Sistemul reușește să livreze o interfață grafică intuitivă și ușor de învățat, în acest mod ușurând atât activitatea angajaților cât și oferind o experiență plăcută utilizatorilor acestui sistem.

8.2. Dezvoltări ulterioare

Ca și la orice sistem software, șansele ca un sistemul să trebuiască să fie supus unor schimbări sau dezvoltări ulterioare sunt mari, din această cauză sistemul a fost proiectat pentru a putea fi extins. În continuare o să fie descrisă o listă cu îmbunătățiri care pot fi aduse acestui sistem și posibilități de dezvoltare ulterioară.

- O primă îmbunătățire care ar putea fi adusă acestui sistem, este plata abonamentelor online. Cu ajutorul acestei funcționalități, clienții ar putea comanda și plăti abonamentele online, de acasă.
- O a doua îmbunătățire a acestui sistem, ar putea fi reprezentată de controlul accesului clienților în complexul sportiv, pe baza unui card de acces. În acest fel, utilizatorii o să aibă un bilanț al intrărilor în complexul sportiv, iar munca angajaților o să fie ușurată, nemaifiind nevoie de gestiunea clienților complexului sportiv.
- O a treia îmbunătățire a sistemului este reprezentată de crearea unui “membership”, prin care utilizatorii pot să se înscrie într-un grup împreună cu membrii familiei sau prieteni, împreună cu care pot participa la activitățile sportive.

- O a patra îmbunătățire a sistemului poate fi reprezentată de implementarea unei soluții pentru generarea diferitelor promoții pentru utilizatori. De exemplu, clientul să beneficieze de o reducere în luna în care este născut.
- O posibilitate de dezvoltare ulterioară a sistemului poate fi reprezentată de dezvoltarea unei aplicații mobile pentru complexul sportiv. Aplicația mobilă poate fi dezvoltată în așa fel încât să gestioneze kaloriile consumate pe perioada activităților sportive și să ofere un plan nutrițional zilnic, pentru fiecare client al complexului sportiv.

Bibliografie

- [1] Antonio Goncalves, Beginning Java EE 7, Manning, 2013.
Disponibil la: <http://pdf.th7.cn/download/files/1508/Beginning%20Java%20EE%207.pdf>
- [2] Craig Walls, Spring Boot in Action, Manning 2010
Disponibil la: <http://pdf.th7.cn/download/files/1603/Spring%20Boot%20in%20Action.pdf>
- [3] Christian Bauer and Gavin King, Java Persistence with Hibernate, Manning 2005
Disponibil la: <http://staff.ustc.edu.cn/~dingqing/teach/ssh/resource/Manning+-+Java+Persistence+with+Hibernate.pdf>
- [4] Qusay H. Mahmoud Servlets and JSP Pages Best Practices, March 2003
Disponibil la: <http://www.oracle.com/technetwork/articles/javase/servlets-jsp-140445.html>
- [5] Documentația oficială SpringBoot, <https://docs.spring.io/springboot/docs/current/reference/htmlsingle/>
- [6] Documentația oficială JSP de la Oracle, <http://docs.oracle.com/javaee/5/tutorial/doc/bnajo.html>
- [7] Documentația oficială Bootstrap Framework, <http://getbootstrap.com/getting-started/>
- [8] Ramez Elmasri, Shamkant B. Navathe, „Fundamentals of Database Systems”, 6th Edition, 2011, disponibil la: http://lms.uop.edu.jo/lms/pluginfile.php/2376/mod_resource/content/0/Fundamentals_of_Database_Systems%2C_6th_Edition.pdf
- [9] Documentația oficială de dezvoltare a paginilor web folosind JSP a IBM, disponibilă la: <http://www.redbooks.ibm.com/redpapers/pdfs/redp4880.pdf>
- [10] Savills World Research, „World Student Housing”, studiu de caz referitor la cazările studentești, disponibil la: <http://classof2020.nl/wpcontent/uploads/2016/06/Spotlight-World-Student-Housing-201516-Savills1.pdf>
- [11] Leonard Richardson, Sam Ruby, „RESTful Web Services”, O'Reilly Media, 2008, disponibilă la: https://www.crummy.com/writing/RESTful-WebServices/RESTful_Web_Services.pdf
- [12] Java EE <https://dzone.com/articles/java-ee-basics>
- [13] MVC Arhitecture <http://www.tutorialsteacher.com/mvc/mvc-architecture>
- [14] Spring Boot Structure <http://www.adeveloperdiary.com/java/spring-boot/an-introduction-to-spring-boot/>
- [15] Hibernate Arhitecture <https://www.javatpoint.com/hibernate-architecture>
- [16] Java Persistence API <https://aishwaryavaishno.wordpress.com/2013/06/07/hibernate-persistent-context-and-objects-lifecycle/comment-page-1/>
- [17] JSP Arhitecture <http://javabeat.net/jsp-architecture-lifecycle/>
<http://www.tutorialsteacher.com/mvc/mvc-architecture>

Bibliografie

- [18] Sistemul similar Gestiune Sala, disponibil la <http://www.gestiunesala.ro/>
- [19] Sistemul similar DashPlatform Recreational Management Software
disponibil la: <https://www.dashplatform.com/>
- [20] Sistemul similar VirtuaGym, dispobil la: <https://www.dashplatform.com/>

Anexa 1

Figura 3.1 Arhitectura MVC	7
Figura 4.1 Structura framework-ului Spring Boot	12
Figura 4.2 Arhitectura Hibernate.....	14
Figura 4.3 Java Persistence API	14
Figura 4.4 Arhitectura JSP	15
Figura 4.5 Fluxul transmiterii unei cereri SOAP.....	17
Figura 4.6 Fluxul transmiterii unei cereri REST	17
Figura 4.7 Cazuri de utilizare pentru administrator.....	23
Figura 4.8 Cazuri de utilizare pentru angajat	24
Figura 4.9 Cazuri de utilizare pentru client.....	24
Figura 4.10 Cazuri de utilizare pentru vizitator	25
Figura 4.11 Diagramă flow-chart pentru adăugarea unei programări	26
Figura 4.12 Diagramă flow-chart pentru comandarea unui abonament	27
Figura 4.13 Diagramă de secvență pentru adăugarea unei programări.....	28
Figura 5.1 Diagrama arhitecturală a sistemului.....	29
Figura 5.2 Arhitectura layer-ul Web	31
Figura 5.3 Diagrama claselor “Controller”	33
Figura 5.4 Diagrama claselor “Service”	34
Figura 5.5 Metoda pentru trimiterea unui mail	35
Figura 5.6 Metoda pentru generarea rapoartelor	36
Figura 5.7 Metoda pentru configurarea autentificării	36
Figura 5.8 Metoda pentru configurarea autorizării.....	37
Figura 5.9 Diagrama de clase “Repository”	37
Figura 5.10 Arhitectura layerului Repository	38
Figura 5.11 Pagini web comune rolurilor.Navigarea.....	38
Figura 5.12 Diagramă de navigare pentru utilizatorul client	39
Figura 5.13 Diagramă de navigare pentru utilizatorul cu rolul de angajat	40
Figura 5.14 Diagramă de navigare pentru utilizatorul cu rolul de vizitator	40
Figura 5.15 Diagrama de pachete a sistemului.....	41
Figura 5.16 Diagrama de clase “Model”.....	42
Figura 5.17 Diagrama de deployment.....	42
Figura 7.1 Pagina principală a aplicației.....	48
Figura 7.2 Tipurile de abonamente	48
Figura 7.3 Pagina cu galeria foto a aplicației	49
Figura 7.4 Pagina de login a aplicației.....	49
Figura 7.5 Vizualizarea tuturor abonamentelor.....	50
Figura 7.6 Adăugarea unui abonament	50
Figura 7.7 Pagina aplicației pentru gestiunea utilizatorilor.....	51
Tabel 3.1 Comparția sistemelor similare.....	11
Tabel 6.1 Cazul de test pentru adăugarea unei programări	45
Tabel 6.2 Cazul de test pentru comandarea unui abonament	46

Anexa 2 – Glosar de termeni

MIS	Management Information System
ERP	Enterprise Resource Plannig
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
RDBMS	Relational Database Management System
SQLOS	SQL Server Operationg System
JSP	JavaServer Pages
HMTL	HyperText Markup Language
SQL	Structured Query Language
API	Application Programming Interface
XML	eXtensible Markup Language
DOM	Document Object Model
CRUD	Create Read Update Delete
ORM	Object Relational Mapping